Individual_Assignment_apatula

February 16, 2024

0.0.1 Individual Assignnment

0.0.2 Adam Patula

0.0.3 Andrew ID: APATULA

Import Initial Libraries and Packages

[1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import altair as alt
from sklearn.preprocessing import MinMaxScaler
from pandas.tseries.holiday import USFederalHolidayCalendar
from pandas.tseries.offsets import CustomBusinessDay
from datetime import datetime, timedelta
from tensorflow.keras.models import load_model
```

```
WARNING:tensorflow:From C:\Users\patul\anaconda3\Lib\site-
packages\keras\src\losses.py:2976: The name
tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

1 Data Acquisition and Cleaning

Yahoo Finance Historical GameStop Stock Price Data

[2]:

```
# GME Stock Data 04JAN-31DEC21
#URL:
#https://finance.yahoo.com/quote/GME/history?
 ↪period1=1609718400&period2=1640908800&interval=1d&filter=history&frequency=1d&includeAdjust

df_stock = pd.read_csv("GME.csv")
```

[3]:

```
#convert date to datetime object
df_stock['Date'] = pd.to_datetime(df_stock['Date'])
```

EDA for Yahoo Finance Data

[4]:

```
df_stock.head()
```

```
[4]:          Date    Open    High     Low   Close  Adj Close    Volume
     0  2021-01-04  4.7500  4.7750  4.2875  4.3125     4.3125  40090000
     1  2021-01-05  4.3375  4.5200  4.3075  4.3425     4.3425  19846000
     2  2021-01-06  4.3350  4.7450  4.3325  4.5900     4.5900  24224800
     3  2021-01-07  4.6175  4.8625  4.5050  4.5200     4.5200  24517200
     4  2021-01-08  4.5450  4.5750  4.2700  4.4225     4.4225  25928000
```

```
[5]: df_stock.columns
```

```
[5]: Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'],
     dtype='object')
```

```
[6]: df_stock = df_stock.drop("Adj Close", axis=1)
```

```
[7]: df_stock.describe()
```

```
[7]:              Open         High          Low        Close        Volume
     count  252.000000   252.000000   252.000000   252.000000  2.520000e+02
     mean    42.761091    45.461538    40.083353    42.426686  5.732827e+07
     std     14.877450    16.198366    13.652308    14.391762  1.138750e+08
     min      4.335000     4.520000     4.270000     4.312500  3.272800e+06
     25%     38.930625    40.059375    36.928750    38.792500  8.615300e+06
     50%     44.261252    46.209999    42.885000    44.508751  1.706600e+07
     75%     50.973750    53.127501    48.940624    50.692500  4.798740e+07
     max     94.927498   120.750000    72.877502    86.877502  7.886316e+08
```

Visualizing Closing Prices

```
[8]: chart_GME_closing = alt.Chart(df_stock).mark_line().encode(
         x=alt.X('Date:T', axis=alt.Axis(titleFontSize=16)),
         y=alt.Y("Close:Q", axis=alt.Axis(titleFontSize=16)).title("Closing Price␣
       ↪($)"),
         #Source: https://altair-viz.github.io/gallery/scatter_tooltips.html
         tooltip=['Date', 'Close']
     ).properties(
         width=800,
         height=400,
         title='GME Closing Prices (04JAN-31DEC21)'
     ).configure_title(
         fontSize=24
     ).configure_axis(
         labelFontSize=16
     ).interactive()
     chart_GME_closing
```

```
[8]: alt.Chart(…)
```

2 Stock Data Feature Engineering

[9]:
```
#rolling average feature
df_stock_rolling = df_stock.copy()
#Source: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.rolling.
 ↪html
df_stock_rolling['7 Day Rolling Avg'] = df_stock_rolling['Close'].
 ↪rolling(window=7).mean()
```

[10]:
```
#Source: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.bfill.
 ↪html#pandas.DataFrame.bfill
# back fill initial rolling average window with the first available value
df_stock_rolling['7 Day Rolling Avg'].fillna(method='bfill', inplace=True)
```

[11]:
```
def move_close_to_end(data):
    #Source: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.pop.
 ↪html
    #Remove close so that it can be added as the last element to the dataframe␣
 ↪for convenience for Train-Test split
    column_to_move = data.pop('Close')
    #Source: https://www.geeksforgeeks.org/
 ↪insert-a-given-column-at-a-specific-position-in-a-pandas-dataframe/
    data.loc[:, 'Close'] = column_to_move
    return data
```

[12]:
```
df_stock_rolling = move_close_to_end(df_stock_rolling)
```

[13]:
```
df_stock_rolling.columns
```

[13]:
```
Index(['Date', 'Open', 'High', 'Low', 'Volume', '7 Day Rolling Avg', 'Close'],
dtype='object')
```

[14]:
```
df_stock_rolling.head()
```

[14]:
```
         Date    Open    High     Low    Volume  7 Day Rolling Avg   Close
0  2021-01-04  4.7500  4.7750  4.2875  40090000           4.594286  4.3125
1  2021-01-05  4.3375  4.5200  4.3075  19846000           4.594286  4.3425
2  2021-01-06  4.3350  4.7450  4.3325  24224800           4.594286  4.5900
3  2021-01-07  4.6175  4.8625  4.5050  24517200           4.594286  4.5200
4  2021-01-08  4.5450  4.5750  4.2700  25928000           4.594286  4.4225
```

[15]:
```
df_stock_rolling.head()
```

[15]:
```
         Date    Open    High     Low    Volume  7 Day Rolling Avg   Close
0  2021-01-04  4.7500  4.7750  4.2875  40090000           4.594286  4.3125
1  2021-01-05  4.3375  4.5200  4.3075  19846000           4.594286  4.3425
2  2021-01-06  4.3350  4.7450  4.3325  24224800           4.594286  4.5900
```

```
3 2021-01-07  4.6175  4.8625  4.5050  24517200           4.594286  4.5200
4 2021-01-08  4.5450  4.5750  4.2700  25928000           4.594286  4.4225
```

Reddit Dataset on Meme Stock: GameStop Han, Jing, 2022, “Reddit Dataset on Meme Stock: GameStop”, https://doi.org/10.7910/DVN/TUMIPC, Harvard Dataverse, V3, UNF:6:c9s1zhZLHH+k32UmoPZu7A== [fileUNF]

Subset used: rGME_dataset_features

[16]:

```python
df_rGME = pd.read_csv('rGME_dataset_features.csv', index_col=0)
```

```
C:\Users\patul\AppData\Local\Temp\ipykernel_21360\1996247414.py:1: DtypeWarning:
Columns (8) have mixed types. Specify dtype option on import or set
low_memory=False.
  df_rGME = pd.read_csv('rGME_dataset_features.csv', index_col=0)
```

[17]:

```python
df_rGME.head()
```

[17]:

```
       id                                              title  \
0  kqfajb              You NEED to see this about GME
1  kqjh2t                     Short Squeeze Incoming
2  kqvp7l  THIS CONVINCED ME TO ALL IN  GME (EXTREME PUMP…
3  krcwch  You already know what we must do brothers and …
4  krnthg                          ICR conference (11th Jan)

                                                 url  score            author  \
0  https://www.reddit.com/r/GME/comments/kqfajb/y…    1.0      TitsDownOnly
1  /r/wallstreetbets/comments/kqcwdo/gamestops_gr…    1.0       zoomermoney
2  https://www.reddit.com/r/GME/comments/kqvp7l/t…    1.0      TitsDownOnly
3  /r/wallstreetbets/comments/kr98ym/gme_gang_we_…    1.0  dontforgettolive
4  https://www.reddit.com/r/GME/comments/krnthg/i…    1.0           nicky94

   num_comments        date flair  compound    neg  …  _poss  _intj _appos  \
0           9.0  2021-01-04   NaN    0.9872  0.000  …      0      0      6
1           1.0  2021-01-04   NaN    0.9906  0.000  …      0      0      3
2           6.0  2021-01-05   NaN    0.5319  0.000  …      0      0      0
3           4.0  2021-01-05   NaN   -0.2960  0.115  …      1      0      0
4          10.0  2021-01-06   NaN    0.0000  0.000  …      0      0      0

  _npadvmod  _predet  _case  _expl  _oprd _dative  _nmod
0         0        0      0      0      0       0      0
1         0        0      0      0      0       0      0
2         0        0      0      0      0       0      0
3         0        0      0      0      0       0      0
4         1        0      0      0      0       0      0

[5 rows x 73 columns]
```

[18]:

```
df_rGME.columns
```

[18]:

```
Index(['id', 'title', 'url', 'score', 'author', 'num_comments', 'date',
       'flair', 'compound', 'neg', 'neu', 'pos', 'spacy_title', 'clean_text',
       'char_count', 'word_count', 'word_count_cln', 'stopword_count',
       'pos_tags', 'dep_tags', '_NOUN', '_VERB', '_ADP', '_ADJ', '_DET',
       '_PROPN', '_INTJ', '_PUNCT', '_NUM', '_PRON', '_ADV', '_PART', '_amod',
       '_ROOT', '_punct', '_advmod', '_auxpass', '_nsubjpass', '_ccomp',
       '_acomp', '_neg', '_nsubj', '_aux', '_agent', '_det', '_pobj', '_prep',
       '_csubj', '_nummod', '_attr', '_acl', '_relcl', '_dobj', '_pcomp',
       '_xcomp', '_cc', '_conj', '_mark', '_prt', '_compound', '_dep',
       '_advcl', '_parataxis', '_poss', '_intj', '_appos', '_npadvmod',
       '_predet', '_case', '_expl', '_oprd', '_dative', '_nmod'],
      dtype='object')
```

[19]:

```
df_rGME.describe()
```

[19]:

```
              score  num_comments      compound           neg           neu  \
count  1.033236e+06  1.033236e+06  1.033236e+06  1.033236e+06  1.033236e+06
mean   3.486193e+00  1.226942e+01  1.522685e-01  6.738097e-02  7.934561e-01
std    9.373280e+01  1.075906e+02  4.197483e-01  1.477525e-01  2.366692e-01
min    0.000000e+00  0.000000e+00 -9.963000e-01  0.000000e+00  0.000000e+00
25%    1.000000e+00  1.000000e+00  0.000000e+00  0.000000e+00  6.520000e-01
50%    1.000000e+00  5.000000e+00  0.000000e+00  0.000000e+00  8.520000e-01
75%    1.000000e+00  1.000000e+01  4.574000e-01  5.900000e-02  1.000000e+00
max    5.957800e+04  3.618900e+04  1.000000e+00  1.000000e+00  1.000000e+00

                pos    char_count    word_count  word_count_cln  \
count  1.033236e+06  1.033236e+06  1.033236e+06    1.033236e+06
mean   1.391690e-01  6.430340e+01  1.445187e+01    6.717018e+00
std    2.072161e-01  5.905740e+01  1.314803e+01    5.758246e+00
min    0.000000e+00  1.000000e+00  1.000000e+00    0.000000e+00
25%    0.000000e+00  2.600000e+01  6.000000e+00    3.000000e+00
50%    0.000000e+00  4.500000e+01  1.000000e+01    5.000000e+00
75%    2.310000e-01  7.700000e+01  1.700000e+01    9.000000e+00
max    1.000000e+00  3.210000e+02  1.590000e+02    1.590000e+02

       stopword_count  …         _poss         _intj        _appos  \
count    1.033236e+06  …  1.033236e+06  1.033236e+06  1.033236e+06
mean     5.965602e+00  …  2.562241e-01  3.902497e-02  1.717391e-01
std      6.800317e+00  …  5.142755e-01  2.188352e-01  7.734972e-01
min      0.000000e+00  …  0.000000e+00  0.000000e+00  0.000000e+00
25%      2.000000e+00  …  0.000000e+00  0.000000e+00  0.000000e+00
50%      4.000000e+00  …  0.000000e+00  0.000000e+00  0.000000e+00
75%      8.000000e+00  …  0.000000e+00  0.000000e+00  0.000000e+00
max      5.100000e+01  …  9.000000e+00  1.900000e+01  1.470000e+02
```

```
          _npadvmod       _predet         _case         _expl         _oprd  \
count  1.033236e+06  1.033236e+06  1.033236e+06  1.033236e+06  1.033236e+06
mean   1.465261e-01  2.496235e-02  3.924273e-02  6.202842e-03  4.940788e-03
std    4.274982e-01  1.573509e-01  2.008172e-01  8.162009e-02  8.132627e-02
min    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
25%    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
50%    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
75%    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
max    5.800000e+01  3.000000e+00  2.000000e+01  3.000000e+00  1.100000e+01

            _dative         _nmod
count  1.033236e+06  1.033236e+06
mean   1.413811e-02  1.451914e-01
std    1.214386e-01  9.251204e-01
min    0.000000e+00  0.000000e+00
25%    0.000000e+00  0.000000e+00
50%    0.000000e+00  0.000000e+00
75%    0.000000e+00  0.000000e+00
max    4.000000e+00  1.480000e+02

[8 rows x 63 columns]
```

Data Cleaning and Extraction of Desired Features

[20]:
```
# group by date and get mean of compound sentiment score to merge with stock␣
↪data
groups = df_rGME.groupby("date")['compound'].mean()
print(len(groups))
grouped_sentiment = pd.DataFrame(groups)
#Source: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.
↪reset_index.html
# reset index to allow for merge
grouped_sentiment = grouped_sentiment.reset_index()
# convert to datetime
grouped_sentiment['Date'] = pd.to_datetime(grouped_sentiment['date'])
grouped_sentiment = grouped_sentiment.drop('date', axis=1)
grouped_sentiment.head()
```

```
343
```

[20]:
```
   compound       Date
0  0.988900 2021-01-04
1  0.117950 2021-01-05
2  0.388850 2021-01-06
3  0.966700 2021-01-10
4  0.237925 2021-01-11
```

```
[21]: # group by date and get mean of positive sentiment score to merge with stock␣
       ↪data
      group2 = df_rGME.groupby("date")['pos'].mean()
      print(len(group2))
      grouped_sentiment2 = pd.DataFrame(group2)
      #Source: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.
       ↪reset_index.html
      # reset index to allow for merge
      grouped_sentiment2 = grouped_sentiment2.reset_index()
      # convert to datetime
      grouped_sentiment2['Date'] = pd.to_datetime(grouped_sentiment2['date'])
      grouped_sentiment2 = grouped_sentiment2.drop('date', axis=1)
      grouped_sentiment2.head()
```

```
343
```

```
[21]:        pos       Date
      0  0.86600 2021-01-04
      1  0.12800 2021-01-05
      2  0.18850 2021-01-06
      3  0.60800 2021-01-10
      4  0.14425 2021-01-11
```

```
[22]: # group by date and get mean of negative sentiment score to merge with stock␣
       ↪data
      group3 = df_rGME.groupby("date")['neg'].mean()
      print(len(group3))
      grouped_sentiment3 = pd.DataFrame(group3)
      #Source: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.
       ↪reset_index.html
      # reset index to allow for merge
      grouped_sentiment3 = grouped_sentiment3.reset_index()
      # convert to datetime
      grouped_sentiment3['Date'] = pd.to_datetime(grouped_sentiment3['date'])
      grouped_sentiment3 = grouped_sentiment3.drop('date', axis=1)
      grouped_sentiment3.head()
```

```
343
```

```
[22]:       neg       Date
      0  0.0000 2021-01-04
      1  0.0575 2021-01-05
      2  0.0535 2021-01-06
      3  0.0000 2021-01-10
      4  0.0000 2021-01-11
```

```
[23]: grouped_rGME_sentiment = grouped_sentiment.merge(grouped_sentiment2)
      grouped_rGME_sentiment = grouped_rGME_sentiment.merge(grouped_sentiment3)
```

```
grouped_rGME_sentiment.head()
```

[23]:

```
   compound        Date      pos     neg
0  0.988900 2021-01-04  0.86600  0.0000
1  0.117950 2021-01-05  0.12800  0.0575
2  0.388850 2021-01-06  0.18850  0.0535
3  0.966700 2021-01-10  0.60800  0.0000
4  0.237925 2021-01-11  0.14425  0.0000
```

[24]:

```
# disable altair max row limit
alt.data_transformers.disable_max_rows()
```

[24]:

```
DataTransformerRegistry.enable('default')
```

[25]:

```
# chart with Compound Sentiment Scores
chart_GME_sentiment = alt.Chart(grouped_sentiment).mark_line().encode(
    x=alt.X('Date:T').title("Date"),
    y=alt.Y('compound:Q').title("Sentiment Score (-1:1)"),
    #color='Sentiment:N',
    tooltip=['Date:T','compound:Q']
).properties(
    width=800,
    height=400,
    title="Sentiment Trends over Time"
).interactive()
chart_GME_sentiment
```

[25]:

```
alt.Chart(…)
```

[26]:

```
#### Stock and rGME cleaned data with Compound, Negative, and Positive␣
 ↪Sentiment Scores
```

[27]:

```
# merge with stock data and create new dataframe
df_rGME_sentiment_stock = df_stock_rolling.copy()
df_rGME_sentiment_stock = df_rGME_sentiment_stock.merge(grouped_rGME_sentiment,␣
 ↪how='left')
df_rGME_sentiment_stock['compound'] = df_rGME_sentiment_stock['compound'].
 ↪fillna(0)
df_rGME_sentiment_stock['pos'] = df_rGME_sentiment_stock['pos'].fillna(0)
df_rGME_sentiment_stock['neg'] = df_rGME_sentiment_stock['neg'].fillna(0)
df_rGME_sentiment_stock = move_close_to_end(df_rGME_sentiment_stock)
df_rGME_sentiment_stock.head()
```

[27]:

```
         Date    Open    High     Low    Volume  7 Day Rolling Avg  compound  \
0  2021-01-04  4.7500  4.7750  4.2875  40090000           4.594286   0.98890
1  2021-01-05  4.3375  4.5200  4.3075  19846000           4.594286   0.11795
2  2021-01-06  4.3350  4.7450  4.3325  24224800           4.594286   0.38885
```

```
3 2021-01-07  4.6175  4.8625  4.5050  24517200           4.594286   0.00000
4 2021-01-08  4.5450  4.5750  4.2700  25928000           4.594286   0.00000

      pos     neg   Close
0  0.8660  0.0000  4.3125
1  0.1280  0.0575  4.3425
2  0.1885  0.0535  4.5900
3  0.0000  0.0000  4.5200
4  0.0000  0.0000  4.4225
```

[28]:

```
df_rGME_sentiment_stock.isna().sum()
```

[28]:

```
Date                 0
Open                 0
High                 0
Low                  0
Volume               0
7 Day Rolling Avg    0
compound             0
pos                  0
neg                  0
Close                0
dtype: int64
```

[29]:

```
df_rGME_sentiment_melt = df_rGME_sentiment_stock.melt(
    id_vars=['Date', 'Close'],
    value_vars=['compound', 'pos', 'neg'],
    var_name='Sentiment Value',
    value_name='Score'
)
df_rGME_sentiment_melt.head()
```

[29]:

```
        Date   Close Sentiment Value    Score
0 2021-01-04  4.3125        compound  0.98890
1 2021-01-05  4.3425        compound  0.11795
2 2021-01-06  4.5900        compound  0.38885
3 2021-01-07  4.5200        compound  0.00000
4 2021-01-08  4.4225        compound  0.00000
```

[30]:

```
# Source for Layered Chart: https://altair-viz.github.io/gallery/
 ↪layered_chart_with_dual_axis.html
base = alt.Chart(df_rGME_sentiment_melt).encode(
    x=alt.X('Date:T'),
    tooltip=['Date:T', 'Close:Q']
).properties(
    width=800,
    height=400,
    title="GME Stock Price compared to rGME Sentiment Scores over Time"
).interactive()

line1 = base.mark_line().encode(
    y=alt.Y('Score:Q').title("Sentiment Score (-1:1)"),
    color='Sentiment Value'
)

line2 = base.mark_line(opacity=0.3, interpolate='monotone').encode(
    y='Close:Q'
)

alt.layer(line2, line1).resolve_scale(
    y='independent'
)
```

[30]:
```
alt.LayerChart(…)
```

Scraped Google News Archive Search “GameStop” via Selenium Search URL: https://www.google.com/search?q=GameStop&sca_esv=54b6785aa301342b&tbs=cdr:1,cd_min:1/4/2021,cd_ma

Data scraped includes news article headlines and publication dates from 04JAN2021-31DEC2021

[31]:
```
df_news = pd.read_csv('GME_news_data.csv')
```

[32]:
```
df_news.head()
```

[32]:
```
         Date                                              Title
0  2021-01-25                      The GameStop Game Never Stops
1  2021-01-26  The GameStop Fiasco Proves We're in a 'Meme St…
2  2021-01-27  The complete moron's guide to GameStop's stock…
3  2021-01-27  Steven Cohen's Point72 invests $750 million in…
4  2021-01-28  The Whole Messy, Ridiculous GameStop Saga in O…
```

[33]:
```
len(df_news)
```

[33]:
```
260
```

3 Sentiment Analysis

Initial source: https://realpython.com/python-nltk-sentiment-analysis/

*Help with starting Sentiment Analysis code received from ChatGPT, please see Appendix below

[34]:
```
import nltk
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\patul\AppData\Roaming\nltk_data…
[nltk_data]   Package vader_lexicon is already up-to-date!
```

[34]:
```
True
```

[35]:
```python
# import nltk sentiment intensity analyzer
from nltk.sentiment import SentimentIntensityAnalyzer

sia = SentimentIntensityAnalyzer()
```

[36]:
```python
def get_sentiment(text):
    '''
    Function takes a str input and returns a dictionary with nltk sentiment␣
 ↪intensity analyzer scores

    get_sentiment('some text') --> {'neg': 0.0, 'neu': 0.735, 'pos': 0.265,␣
 ↪'compound': 0.1139}
    '''
    return sia.polarity_scores(text)
```

[37]:
```python
# get sentiment scores of article titles
df_news['Sentiment'] = df_news['Title'].apply(get_sentiment)
df_news['Sentiment'][0]
```

[37]:
```
{'neg': 0.0, 'neu': 0.735, 'pos': 0.265, 'compound': 0.1139}
```

[38]:
```python
#Source: https://www.skytowner.com/explore/
 ↪splitting_dictionary_into_separate_columns_in_pandas_dataframe
#split sentiment scores into multiple columns
split_sentiment = df_news['Sentiment'].apply(pd.Series)
#Source: https://pandas.pydata.org/docs/reference/api/pandas.concat.html
# add new columns back to dataframe
df_news = pd.concat([df_news, split_sentiment], axis=1)
```

[39]:
```python
df_news.head(10)
```

[39]:
```
         Date                                              Title  \
0  2021-01-25                       The GameStop Game Never Stops
1  2021-01-26  The GameStop Fiasco Proves We're in a 'Meme St…
2  2021-01-27  The complete moron's guide to GameStop's stock…
3  2021-01-27  Steven Cohen's Point72 invests $750 million in…
4  2021-01-28  The Whole Messy, Ridiculous GameStop Saga in O…
5  2021-01-28  GameStop Stock Soars as Reddit Investors Take …
6  2021-01-28  Robinhood CEO speaks to Cuomo after GameStop s…
7  2021-01-28  Robinhood restricts trading in GameStop, other…
8  2021-01-28  GameStop stock run gives it a bigger market ca…
9  2021-01-28  Reddit's WallStreetBets is back, taking on Gam…
```

```
                                          Sentiment    neg    neu    pos  \
0  {'neg': 0.0, 'neu': 0.735, 'pos': 0.265, 'comp…  0.000  0.735  0.265
1  {'neg': 0.292, 'neu': 0.708, 'pos': 0.0, 'comp…  0.292  0.708  0.000
2  {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound…  0.000  1.000  0.000
3  {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound…  0.000  1.000  0.000
4  {'neg': 0.407, 'neu': 0.488, 'pos': 0.106, 'co…  0.407  0.488  0.106
5  {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound…  0.000  1.000  0.000
6  {'neg': 0.316, 'neu': 0.684, 'pos': 0.0, 'comp…  0.316  0.684  0.000
7  {'neg': 0.365, 'neu': 0.635, 'pos': 0.0, 'comp…  0.365  0.635  0.000
8  {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound…  0.000  1.000  0.000
9  {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound…  0.000  1.000  0.000

   compound
0    0.1139
1   -0.5106
2    0.0000
3    0.0000
4   -0.5719
5    0.0000
6   -0.5719
7   -0.5574
8    0.0000
9    0.0000
```

[40]:

```
# group sentiment scores by Date and calculate mean of compound sentiment scores
#Source: https://pandas.pydata.org/pandas-docs/version/0.25.0/reference/api/
 ↪pandas.core.groupby.GroupBy.mean.html
df_news_grouped = df_news.groupby('Date')['compound'].mean()
#Source: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.
 ↪reset_index.html
# reset dataframe index
df_news_grouped = df_news_grouped.reset_index()
df_news_grouped['Date'] = pd.to_datetime(df_news_grouped['Date'])
df_news_grouped.head()
```

[40]:

```
        Date  compound
0 2021-01-25  0.113900
1 2021-01-26 -0.510600
2 2021-01-27  0.000000
3 2021-01-28 -0.099862
4 2021-01-29 -0.000281
```

[41]:

```
# group sentiment scores by Date and calculate mean of compound sentiment scores
#Source: https://pandas.pydata.org/pandas-docs/version/0.25.0/reference/api/
 ↪pandas.core.groupby.GroupBy.mean.html
df_news_grouped2 = df_news.groupby('Date')['pos'].mean()
```

```
#Source: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.
 ↪reset_index.html
# reset dataframe index
df_news_grouped2 = df_news_grouped2.reset_index()
df_news_grouped2['Date'] = pd.to_datetime(df_news_grouped2['Date'])
df_news_grouped2.head()
```

[41]:

```
        Date       pos
0 2021-01-25  0.265000
1 2021-01-26  0.000000
2 2021-01-27  0.000000
3 2021-01-28  0.048308
4 2021-01-29  0.067769
```

[42]:

```
# group sentiment scores by Date and calculate mean of compound sentiment scores
#Source: https://pandas.pydata.org/pandas-docs/version/0.25.0/reference/api/
 ↪pandas.core.groupby.GroupBy.mean.html
df_news_grouped3 = df_news.groupby('Date')['neg'].mean()
#Source: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.
 ↪reset_index.html
# reset dataframe index
df_news_grouped3 = df_news_grouped3.reset_index()
df_news_grouped3['Date'] = pd.to_datetime(df_news_grouped3['Date'])
df_news_grouped3.head()
```

[42]:

```
        Date       neg
0 2021-01-25  0.000000
1 2021-01-26  0.292000
2 2021-01-27  0.000000
3 2021-01-28  0.122385
4 2021-01-29  0.072538
```

[43]:

```
grouped_google_sentiment = df_news_grouped.merge(df_news_grouped2)
grouped_google_sentiment = grouped_google_sentiment.merge(df_news_grouped3)
grouped_google_sentiment.head()
```

[43]:

```
        Date  compound       pos       neg
0 2021-01-25  0.113900  0.265000  0.000000
1 2021-01-26 -0.510600  0.000000  0.292000
2 2021-01-27  0.000000  0.000000  0.000000
3 2021-01-28 -0.099862  0.048308  0.122385
4 2021-01-29 -0.000281  0.067769  0.072538
```

[44]:

```
len(df_news_grouped)
```

[44]:

```
80
```

```
[45]: #Source: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.
        ↪html
      # create new dataframe for merged stock and news data
      df_join_stock_news = df_stock_rolling.copy()
      df_join_stock_news = df_join_stock_news.merge(grouped_google_sentiment,␣
        ↪how='left')
      df_join_stock_news = move_close_to_end(df_join_stock_news)
```

```
[46]: len(df_join_stock_news)
```

```
[46]: 252
```

```
[47]: # check for null values
      df_join_stock_news.isna().sum()
```

```
[47]: Date                    0
      Open                    0
      High                    0
      Low                     0
      Volume                  0
      7 Day Rolling Avg       0
      compound              185
      pos                   185
      neg                   185
      Close                   0
      dtype: int64
```

```
[48]: # fill null values with 0 to indicate completely neutral score for days without␣
        ↪articles
      #Source: https://www.geeksforgeeks.org/
        ↪replace-nan-values-with-zeros-in-pandas-dataframe/
      df_join_stock_news = df_join_stock_news.fillna(0)
      df_join_stock_news.head()
```

```
[48]:         Date    Open    High     Low    Volume  7 Day Rolling Avg  compound  \
      0 2021-01-04  4.7500  4.7750  4.2875  40090000           4.594286       0.0
      1 2021-01-05  4.3375  4.5200  4.3075  19846000           4.594286       0.0
      2 2021-01-06  4.3350  4.7450  4.3325  24224800           4.594286       0.0
      3 2021-01-07  4.6175  4.8625  4.5050  24517200           4.594286       0.0
      4 2021-01-08  4.5450  4.5750  4.2700  25928000           4.594286       0.0

         pos  neg   Close
      0  0.0  0.0  4.3125
      1  0.0  0.0  4.3425
      2  0.0  0.0  4.5900
      3  0.0  0.0  4.5200
      4  0.0  0.0  4.4225
```

[49]:

```
df_join_stock_news.isna().sum()
```

[49]:

```
Date                 0
Open                 0
High                 0
Low                  0
Volume               0
7 Day Rolling Avg    0
compound             0
pos                  0
neg                  0
Close                0
dtype: int64
```

[50]:

```
df_google_sentiment_melt = df_join_stock_news.melt(
    id_vars=['Date', 'Close'],
    value_vars=['compound', 'pos', 'neg'],
    var_name='Sentiment Value',
    value_name='Score'
)
df_rGME_sentiment_melt.head()
```

[50]:

```
        Date   Close Sentiment Value    Score
0 2021-01-04  4.3125        compound  0.98890
1 2021-01-05  4.3425        compound  0.11795
2 2021-01-06  4.5900        compound  0.38885
3 2021-01-07  4.5200        compound  0.00000
4 2021-01-08  4.4225        compound  0.00000
```

[51]:

```
# Source for Layered Chart: https://altair-viz.github.io/gallery/
 ↪layered_chart_with_dual_axis.html
base = alt.Chart(df_google_sentiment_melt).encode(
    x=alt.X('Date:T'),
    tooltip=['Date:T', 'Close:Q']
).properties(
    width=800,
    height=400,
    title="GME Stock Price compared to Google Sentiment Scores over Time"
).interactive()

line1 = base.mark_line().encode(
    y=alt.Y('Score:Q').title("Sentiment Score (-1:1)"),
    color='Sentiment Value'
)

line2 = base.mark_line(opacity=0.3, interpolate='monotone').encode(
    y='Close:Q'
)

alt.layer(line2, line1).resolve_scale(
    y='independent'
)
```

```
[51]: alt.LayerChart(…)
```

TDM Studio Sentiment Analysis 04JAN-31DEC21

Source: https://tdmstudio.proquest.com/analysis/viz/sa/apatulaandrewcmuedu-GameStop04JAN-31DEC21-1707689470601 TDM Studio GameStop Sentiment Analysis Data:

ProQuest TDM Studio is a text and data mining research tool at https://tdmstudio.proquest.com/, access for which can be provided through the CMU Library. This tool was used to search for mentions of "GameStop" and related fields in 1051 newspapers, interviews, editorials, dissertations, theses, and journals available in the ProQuest database from 17 different sources (e.g., The Times of India, New York Times, Washington Post, etc.). TDM Studio uses its own, BERT-based model to determine an emotion assignment for each sentence of an article in the search space. These probabilities are then averaged at the document level and then average again for all documents in a specific date range (January 4, 2021 to December 31, 2021 to collect any relevant data).

The resultant dataset contains 10 columns and 51 rows indexed by week, year and month with 9 distinct emotion attributes assigned to the documents within that timeframe:

1. Anger
2. Disgust
3. Fear
4. Sadnes
5. Happiness
6. Love
7. Surprise
8. Neutral
9. Other

Each data point is then an averaged affect probability for each of the nine classes indicating the likelihood that a document in one of the available date ranges is assigned to a specific emotion.

```
[52]: df_TDM = pd.read_csv('emotion_time.csv')
```

```
[53]: df_TDM.head()
```

```
[53]:             Date Range     Anger   Disgust      Fear   Sadness  Happiness  \
      0  2021-01-11/2021-01-17  0.064095  0.064161  0.039228  0.072372   0.172943
      1  2021-01-18/2021-01-24  0.080808  0.066206  0.085939  0.148403   0.154426
      2  2021-01-25/2021-01-31  0.118919  0.086057  0.070727  0.093651   0.128504
      3  2021-02-01/2021-02-07  0.109803  0.090097  0.066075  0.086506   0.137711
      4  2021-02-08/2021-02-14  0.099007  0.091747  0.063766  0.079963   0.147129
```

```
       Love  Surprise   Neutral     Other
0  0.032801  0.278547  0.204678  0.071174
1  0.023851  0.190875  0.181870  0.067623
2  0.031275  0.237286  0.161021  0.072560
3  0.032282  0.224728  0.175876  0.076922
4  0.036771  0.219848  0.181026  0.080744
```

[54]:
```
len(df_TDM)
```

[54]:
```
51
```

Functions to split weekly TDM Sentiment data by day, each week will have the same values as the week range.

[55]:
```
def create_date_ranges(row):
    '''
    This function takes a row in a dataframe as an input where the first row is␣
 ↪a string of the form: 021-01-11/2021-01-17
    This encompasses a week, which is split.
    A list of all the days within the range is then returned.
    **See code generated by ChatGPT for assistance in Appendix A
    '''
    # split date range
    start, end = row['Date Range'].split('/')
    # get start
    start = datetime.strptime(start, "%Y-%m-%d")
    # get end
    end = datetime.strptime(end, "%Y-%m-%d")
    # get dates in between
    date_range = [start + timedelta(days=x) for x in range(0, (end-start).days␣
 ↪+1)]
    return date_range
```

[56]:
```
def create_data_by_day(df):
    '''
    TDM Studio Sentiment Analysis tool conducted analysis on a weekly basis for␣
 ↪the selected date range. This function
    is designed to create a new dataframe that takes the weekly sentiment␣
 ↪values and assigns them to all of the days
    during a given week for simplicity. This does not follow what actually␣
 ↪occurred in terms of document publication
    during the weeks in question.

    Function borrowed by previous project for 95-885, function development␣
 ↪credit goes to Shivaani Krishnakumar
    '''
    df_emotions = df.drop(['Date Range'], axis=1)
    length = len(df)
    day_list = []
    # Source: https://www.geeksforgeeks.org/
↪how-to-add-one-row-in-an-existing-pandas-dataframe/
    for n in range(0, length):
        # get individual row
        row = df.iloc[n]
        #get date range
        date_range = create_date_ranges(row)
        # create a new row for each date, but add each week's sentiment seven␣
↪times
        for date in date_range:
            new_row = {col: row[col] for col in df_emotions.columns}
            new_row['Date'] = date
            day_list.append(new_row)

    df_daily = pd.DataFrame(day_list)
    return df_daily
```

[57]:
```
# create TDM Sentiment dataframe
df_TDM_daily = create_data_by_day(df_TDM)
df_TDM_daily.head()
```

[57]:
```
      Anger   Disgust      Fear   Sadness  Happiness      Love  Surprise  \
0  0.064095  0.064161  0.039228  0.072372   0.172943  0.032801  0.278547
1  0.064095  0.064161  0.039228  0.072372   0.172943  0.032801  0.278547
2  0.064095  0.064161  0.039228  0.072372   0.172943  0.032801  0.278547
3  0.064095  0.064161  0.039228  0.072372   0.172943  0.032801  0.278547
4  0.064095  0.064161  0.039228  0.072372   0.172943  0.032801  0.278547

    Neutral     Other       Date
0  0.204678  0.071174 2021-01-11
1  0.204678  0.071174 2021-01-12
2  0.204678  0.071174 2021-01-13
3  0.204678  0.071174 2021-01-14
4  0.204678  0.071174 2021-01-15
```

[58]:
```
# create melted emotion dataframe for EDA
data_melt = df_TDM_daily.melt(
    id_vars=['Date'],
    value_vars=['Anger', 'Disgust','Fear', 'Sadness', 'Happiness', 'Love',␣
↪'Surprise', 'Neutral', 'Other'],
    value_name='emotion_score',
    var_name='Emotion'
)
data_melt.head()
```

```
[58]:         Date Emotion  emotion_score
      0 2021-01-11   Anger       0.064095
      1 2021-01-12   Anger       0.064095
      2 2021-01-13   Anger       0.064095
      3 2021-01-14   Anger       0.064095
      4 2021-01-15   Anger       0.064095
```

Visualizing TDM Studio Sentiment

```
[59]: # chart with all Sentiment Scores
      chart_TDM_GME_sentiment = alt.Chart(data_melt).mark_line().encode(
          x=alt.X('Date:T'),
          y='emotion_score:Q',
          color='Emotion:N',
          tooltip=['Date:T', 'Emotion:N','emotion_score:Q']
      ).properties(
          width=800,
          height=400,
          title="Sentiment Trends over Time"
      ).interactive()
      chart_TDM_GME_sentiment
```

```
[59]: alt.Chart(…)
```

```
[60]: #chart with only Anger, Fear, Happiness, and Surprise
      # create melted emotion dataframe for EDA
      data_melt_select2 = df_TDM_daily.melt(
          id_vars=['Date'],
          value_vars=['Anger', 'Fear', 'Happiness', 'Surprise'],
          value_name='emotion_score',
          var_name='Emotion'
      )
      data_melt_select2.head()
```

```
[60]:         Date Emotion  emotion_score
      0 2021-01-11   Anger       0.064095
      1 2021-01-12   Anger       0.064095
      2 2021-01-13   Anger       0.064095
      3 2021-01-14   Anger       0.064095
      4 2021-01-15   Anger       0.064095
```

```
[61]: # chart with select Sentiment Scores
      chart_TDM_GME_sentiment = alt.Chart(data_melt_select2).mark_line().encode(
          x=alt.X('Date:T'),
          y='emotion_score:Q',
          color='Emotion:N',
          tooltip=['Date:T', 'Emotion:N','emotion_score:Q']
      ).properties(
          width=800,
          height=400,
          title="Sentiment Trends over Time"
      ).interactive()
      chart_TDM_GME_sentiment
```

```
[61]: alt.Chart(…)
```

Joining TDM Studio Sentiment Analysis and GME Stock data

```
[62]: # merge TDM Sentiment data and stock data
df_merged1 = pd.merge(df_stock_rolling, df_TDM_daily, on='Date', how='left')
```

```
[63]: # check for null values
df_merged1.isnull().sum()
```

```
[63]: Date                  0
Open                  0
High                  0
Low                   0
Volume                0
7 Day Rolling Avg     0
Close                 0
Anger                 5
Disgust               5
Fear                  5
Sadness               5
Happiness             5
Love                  5
Surprise              5
Neutral               5
Other                 5
dtype: int64
```

```
[64]: # back fill null values with most recent
df_merged1 = df_merged1.bfill()
```

```
[65]: df_merged1.isna().sum()
```

```
[65]: Date                  0
Open                  0
High                  0
Low                   0
Volume                0
7 Day Rolling Avg     0
```

```
Close                    0
Anger                    0
Disgust                  0
Fear                     0
Sadness                  0
Happiness                0
Love                     0
Surprise                 0
Neutral                  0
Other                    0
dtype: int64
```

[66]:
```
df_merged1 = move_close_to_end(df_merged1)
```

[67]:
```
df_merged1.columns
```

[67]:
```
Index(['Date', 'Open', 'High', 'Low', 'Volume', '7 Day Rolling Avg', 'Anger',
       'Disgust', 'Fear', 'Sadness', 'Happiness', 'Love', 'Surprise',
       'Neutral', 'Other', 'Close'],
      dtype='object')
```

[68]:
```
#Borrowed from my project for 95-885
# melt and create Tidy dataset for display
sentiment_melt = df_merged1.melt(
    id_vars=['Date', 'Open', 'High', 'Low', 'Volume', '7 Day Rolling␣
 ↪Avg','Close'],
    value_vars=['Anger', 'Disgust', 'Fear', 'Sadness', 'Happiness', 'Love',␣
 ↪'Surprise', 'Neutral', 'Other'],
    value_name='sentiment_score',
    var_name='Sentiment'
)
sentiment_melt.head()
```

[68]:
```
         Date    Open    High     Low    Volume  7 Day Rolling Avg   Close  \
0  2021-01-04  4.7500  4.7750  4.2875  40090000           4.594286  4.3125
1  2021-01-05  4.3375  4.5200  4.3075  19846000           4.594286  4.3425
2  2021-01-06  4.3350  4.7450  4.3325  24224800           4.594286  4.5900
3  2021-01-07  4.6175  4.8625  4.5050  24517200           4.594286  4.5200
4  2021-01-08  4.5450  4.5750  4.2700  25928000           4.594286  4.4225

  Sentiment  sentiment_score
0     Anger         0.064095
1     Anger         0.064095
2     Anger         0.064095
3     Anger         0.064095
4     Anger         0.064095
```

[69]:

```
# Source for Layered Chart: https://altair-viz.github.io/gallery/
 ↪layered_chart_with_dual_axis.html
base = alt.Chart(sentiment_melt).encode(
    x=alt.X('Date:T', axis=alt.Axis(titleFontSize=16)),
    tooltip=['Date:T', 'Close:Q','compound:Q']
).properties(
    width=800,
    height=400,
    title="BTC Closing Price and Emotion Trend Over Time"
).interactive()

line1 = base.mark_line().encode(
    y=alt.Y('sentiment_score:Q', axis=alt.Axis(titleFontSize=16)).
 ↪title("Sentiment Score (-1:1)"),
    color='Sentiment:N'
)

line2 = base.mark_line(opacity=0.3, interpolate='monotone').encode(
    y='Close:Q'
)

alt.layer(line2, line1).resolve_scale(
    y='independent'
)
```

[69]:

```
alt.LayerChart(…)
```

[70]:

```
#Borrowed from my project for 95-885
#Only keeping Anger, Happiness, Neutral, and Fear
sentiment_melt2 = df_merged1.melt(
    id_vars=['Date', 'Open', 'High', 'Low', 'Volume', '7 Day Rolling␣
 ↪Avg','Close'],
    value_vars=['Anger','Fear', 'Happiness', 'Neutral'],
    value_name='sentiment_score',
    var_name='Sentiment'
)
sentiment_melt2.head()
```

[70]:

```
         Date    Open    High     Low    Volume  7 Day Rolling Avg   Close  \
0  2021-01-04  4.7500  4.7750  4.2875  40090000           4.594286  4.3125
1  2021-01-05  4.3375  4.5200  4.3075  19846000           4.594286  4.3425
2  2021-01-06  4.3350  4.7450  4.3325  24224800           4.594286  4.5900
3  2021-01-07  4.6175  4.8625  4.5050  24517200           4.594286  4.5200
4  2021-01-08  4.5450  4.5750  4.2700  25928000           4.594286  4.4225

  Sentiment  sentiment_score
0     Anger         0.064095
```

```
1      Anger         0.064095
2      Anger         0.064095
3      Anger         0.064095
4      Anger         0.064095
```

[71]:
```
# Source for Layered Chart: https://altair-viz.github.io/gallery/
 ↪layered_chart_with_dual_axis.html
base = alt.Chart(sentiment_melt2).encode(
    x=alt.X('Date:T', axis=alt.Axis(titleFontSize=16)),
    tooltip=['Date:T', 'Close:Q','compound:Q']
).properties(
    width=800,
    height=400,
    title="BTC Closing Price and Emotion Trend Over Time"
).interactive()

line1 = base.mark_line().encode(
    y=alt.Y('sentiment_score:Q', axis=alt.Axis(titleFontSize=16)).
 ↪title("Sentiment Score (-1:1)"),
    color='Sentiment:N'
)

line2 = base.mark_line(opacity=0.3, interpolate='monotone').encode(
    y='Close:Q'
)

alt.layer(line2, line1).resolve_scale(
    y='independent'
)
```

[71]:
```
alt.LayerChart(…)
```

4 Time-Series Forecasting

4.0.1 Model Training and Testing

Code for scaling, creating training sequences, LSTM model, business days, predictions, and visualizing predictions borrowed from given Quick_Starter_Notebook

[72]:
```
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.optimizers import Adam
```

4.0.2 Model with Stock Closing Prices Only

[73]:

```
# input features
features = ['Close']
NUM_FEATURES = len(features)
features = df_stock[features].values
```

[74]:

```
# features scaling
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(features.reshape(-1,1))
```

[75]:

```
# Function to create sequences
def create_sequences(data, sequence_length):
    '''
    Function to create an array of sequences from provided time-series data.␣
 ↪This function takes a dataframe as input and a unit
    indicating the desired sequence length to include in the time-series␣
 ↪predictions. It splits the input and target features into
    X and y arrays, appending these pairs to lists that hold the X and y pairs␣
 ↪for predicting each new unit after the current sequence length time frame.
    '''
    xs, ys = [], []
    for i in range(len(data) - sequence_length):
        x = data[i:(i + sequence_length)]
        y = data[i + sequence_length]
        xs.append(x)
        ys.append(y)
    return np.array(xs), np.array(ys)
```

[76]:

```
SEQUENCE_LENGTH = 30  # You can adjust this value -> parameter tuning; set to␣
 ↪30 for approximately 1 month of observations
X, y = create_sequences(scaled_data, SEQUENCE_LENGTH)
```

[77]:

```
print(X.shape)
print(y.shape)
```

```
(222, 30, 1)
(222, 1)
```

[78]:

```
# Split the data into training and test sets (train on data until May 31st)
TRAIN_END_DATE = '2021-05-31'
train_indices = df_stock[df_stock['Date'] <= TRAIN_END_DATE].index
X_train, y_train = X[:train_indices[-1]-SEQUENCE_LENGTH], y[:
 ↪train_indices[-1]-SEQUENCE_LENGTH]
```

[79]:

```
print(X_train.shape, y_train.shape)
```

```
(71, 30, 1) (71, 1)
```

[80]:

```
# function to create LSTM model
def lstm_model(sequence_length, num_features):
    '''
    This function creates an LSTM model for use with different datasets and␣
 ↪input features. It requires the length of the input sequence
    and the number of input features.
    '''
    model = Sequential([
        # adjust input shape and add 10% dropout; add additional dense layer as␣
 ↪experiment
        # 50 LSTM units
        LSTM(50, return_sequences=True,␣
 ↪input_shape=(sequence_length,num_features), dropout=0.1),
        LSTM(50, return_sequences=False, dropout=0.1),
        Dense(25),
        Dense(1)
    ])

    return model
```

[81]:

```
# create Stock only model
# model = lstm_model(SEQUENCE_LENGTH, NUM_FEATURES)
```

[82]:

```
# load model from saved model
model = load_model('stock_model')
```

```
WARNING:tensorflow:From C:\Users\patul\anaconda3\Lib\site-
packages\keras\src\saving\legacy\saved_model\load.py:107: The name
tf.gfile.Exists is deprecated. Please use tf.io.gfile.exists instead.

WARNING:tensorflow:From C:\Users\patul\anaconda3\Lib\site-
packages\keras\src\layers\rnn\lstm.py:148: The name
tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.
```

[83]:

```
# compile model
# model.compile(optimizer='adam', loss='mean_squared_error')
```

[84]:

```
# fit model to training data; use 100 epochs; suppress output
# model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)
```

[85]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 30, 50)            10400

 lstm_1 (LSTM)               (None, 50)                20200

 dense (Dense)               (None, 25)                1275

 dense_1 (Dense)             (None, 1)                 26

=================================================================
Total params: 31901 (124.61 KB)
Trainable params: 31901 (124.61 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

5 Prediction Period 01JUN-31AUG2021

[86]:
```
# get the business date range for the desired testing window 01JUN-31AUG21
business_days = pd.date_range(start=pd.to_datetime(TRAIN_END_DATE) +␣
 ↪timedelta(days=1),
                              periods=66, freq='B')
```

[87]:
```
business_days
```

[87]:
```
DatetimeIndex(['2021-06-01', '2021-06-02', '2021-06-03', '2021-06-04',
               '2021-06-07', '2021-06-08', '2021-06-09', '2021-06-10',
               '2021-06-11', '2021-06-14', '2021-06-15', '2021-06-16',
               '2021-06-17', '2021-06-18', '2021-06-21', '2021-06-22',
               '2021-06-23', '2021-06-24', '2021-06-25', '2021-06-28',
               '2021-06-29', '2021-06-30', '2021-07-01', '2021-07-02',
               '2021-07-05', '2021-07-06', '2021-07-07', '2021-07-08',
               '2021-07-09', '2021-07-12', '2021-07-13', '2021-07-14',
               '2021-07-15', '2021-07-16', '2021-07-19', '2021-07-20',
               '2021-07-21', '2021-07-22', '2021-07-23', '2021-07-26',
               '2021-07-27', '2021-07-28', '2021-07-29', '2021-07-30',
               '2021-08-02', '2021-08-03', '2021-08-04', '2021-08-05',
               '2021-08-06', '2021-08-09', '2021-08-10', '2021-08-11',
               '2021-08-12', '2021-08-13', '2021-08-16', '2021-08-17',
               '2021-08-18', '2021-08-19', '2021-08-20', '2021-08-23',
               '2021-08-24', '2021-08-25', '2021-08-26', '2021-08-27',
               '2021-08-30', '2021-08-31'],
              dtype='datetime64[ns]', freq='B')
```

[88]:
```
#remove holiday, no stock data
business_days = business_days.drop('2021-07-05')
```

[89]:
```
# Get the last sequence from the training data
last_sequence = X_train[-1].reshape((1, SEQUENCE_LENGTH, 1))
```

[90]:
```
last_sequence.shape
```

[90]:
```
(1, 30, 1)
```

[91]:
```
# Create a list to hold predictions
predictions = []

# Predict future prices
for i in range(len(business_days)):
    # Get the prediction (scaled value)
    current_prediction = model.predict(last_sequence, verbose=0)[0]

    # Append the prediction
    predictions.append(current_prediction)

    # Update the sequence
    last_sequence = np.roll(last_sequence, -1, axis=1)
    last_sequence[0, -1, 0] = current_prediction

# Inverse transform the predictions to get actual values
predicted_prices = scaler.inverse_transform(np.array(predictions).reshape(-1,␣
 ↪1))

# Create a DataFrame with the predicted stock prices and dates
predictions_df = pd.DataFrame({
    'Date': business_days,
    'Predicted_Close': predicted_prices.flatten()
})

# Show the prediction results
print(predictions_df)
```

```
          Date  Predicted_Close
0   2021-06-01        47.842812
1   2021-06-02        48.881100
2   2021-06-03        49.395931
3   2021-06-04        49.524757
4   2021-06-07        49.399239
..         …               …
60  2021-08-25        44.527283
61  2021-08-26        44.573296
62  2021-08-27        44.618977
63  2021-08-30        44.662880
64  2021-08-31        44.703724

[65 rows x 2 columns]
```

[92]:

```
# Continue from the previous predictions_df creation code

# Ensure the 'Date' columns in both DataFrames are in the same format
df_stock['Date'] = pd.to_datetime(df_stock['Date'])
predictions_df['Date'] = pd.to_datetime(predictions_df['Date'])

# Merge the predictions with the actual closing prices from 'df'
predictions_with_actuals_df = predictions_df.merge(df_stock[['Date', 'Close']],␣
 ↪on='Date', how='left')

# Rename columns for clarity
predictions_with_actuals_df.rename(columns={'Close': 'Actual_Close'},␣
 ↪inplace=True)

# Show the DataFrame with predictions and actual closing prices
print(predictions_with_actuals_df)
```

```
         Date  Predicted_Close  Actual_Close
0  2021-06-01        47.842812     62.255001
1  2021-06-02        48.881100     70.559998
2  2021-06-03        49.395931     64.544998
3  2021-06-04        49.524757     62.090000
4  2021-06-07        49.399239     70.002502
..        …              …             …
60 2021-08-25        44.527283     49.912498
61 2021-08-26        44.573296     51.305000
62 2021-08-27        44.618977     51.237499
63 2021-08-30        44.662880     52.299999
64 2021-08-31        44.703724     54.560001

[65 rows x 3 columns]
```

[93]:

```
predictions_with_actuals_df.isna().sum()
```

[93]:

```
Date               0
Predicted_Close    0
Actual_Close       0
dtype: int64
```

Test which revealed holiday in business_days datetime index

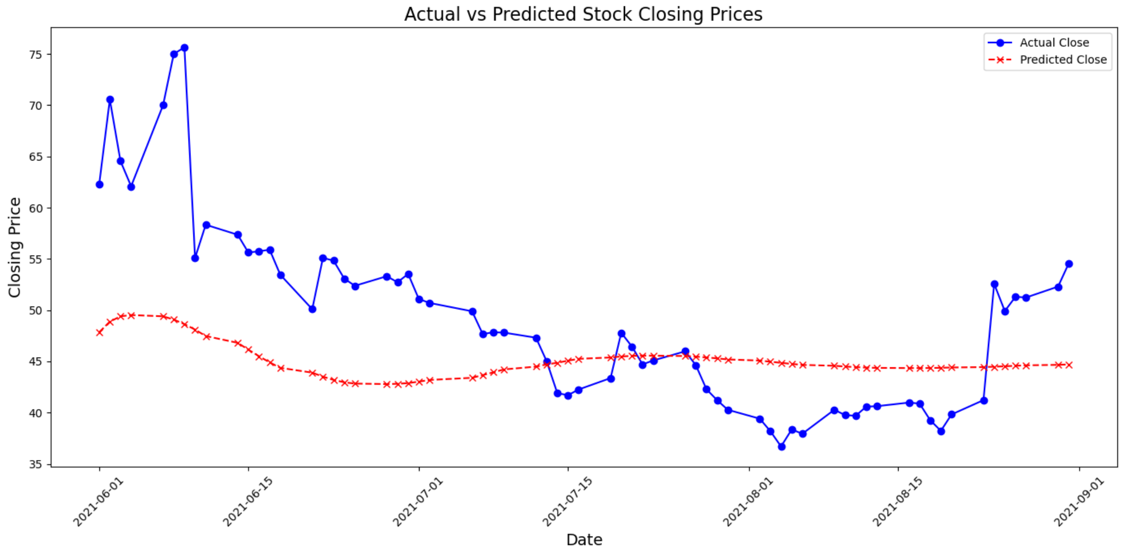
#Source: #https://saturncloud.io/blog/python-pandas-selecting-rows-whose-column-value-is-null-none-nan/#:~:text=The%20simplest%20way%20to%20select,dataframe%20is%20null%20or%20not. null_mask = predictions_with_actuals_df.isnull().any(axis=1) null_rows = predictions_with_actuals_df[null_mask]

print(null_rows) print("This was a federal holiday in the US")

Display Actual versus Predicted Stock Closing Prices

[94]:

```
import matplotlib.pyplot as plt

# Ensure the 'Date' column is in datetime format for proper plotting
predictions_with_actuals_df['Date'] = pd.
 ↪to_datetime(predictions_with_actuals_df['Date'])

# Setting the plot size for better readability
plt.figure(figsize=(14, 7))

# Plotting the actual closing prices
plt.plot(predictions_with_actuals_df['Date'],␣
 ↪predictions_with_actuals_df['Actual_Close'], label='Actual Close',␣
 ↪color='blue', marker='o')

# Plotting the predicted closing prices
plt.plot(predictions_with_actuals_df['Date'],␣
 ↪predictions_with_actuals_df['Predicted_Close'], label='Predicted Close',␣
 ↪color='red', linestyle='--', marker='x')

# Adding title and labels with font size adjustments
plt.title('Actual vs Predicted Stock Closing Prices', fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Closing Price', fontsize=14)

# Rotating date labels for better visibility
plt.xticks(rotation=45)

# Adding a legend to distinguish between actual and predicted values
plt.legend()

# Display the plot
plt.tight_layout()
plt.show()
```



[95]:

```
predictions_with_actuals_df.columns
```

[95]:

```
Index(['Date', 'Predicted_Close', 'Actual_Close'], dtype='object')
```

Display Model Evaluation Metrics: MAE, MSE, RMSE

[96]:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error

def model_evaluation(y_test, predictions):
    '''
    Function to easily display the evaluation metrics of each model
    '''
    mae = mean_absolute_error(y_test, predictions)
    mse = mean_squared_error(y_test, predictions)
    rmse = np.sqrt(mse)

    #Source: https://www.freecodecamp.org/news/
 ↪python-print-variable-how-to-print-a-string-and-variable/
    print(f'MAE: {mae}')
    print(f'MSE: {mse}')
    print(f'RMSE: {rmse}')
    print()
    dictionary = {'MAE':mae,'MSE':mse,'RMSE':rmse}
    return dictionary
```

[97]:

```
# get y_test and predictions for calculations

y_test_original = predictions_with_actuals_df['Actual_Close']
predicted_prices_original = predictions_with_actuals_df['Predicted_Close']
```

```
model_evaluation(y_test_original, predicted_prices_original)
```

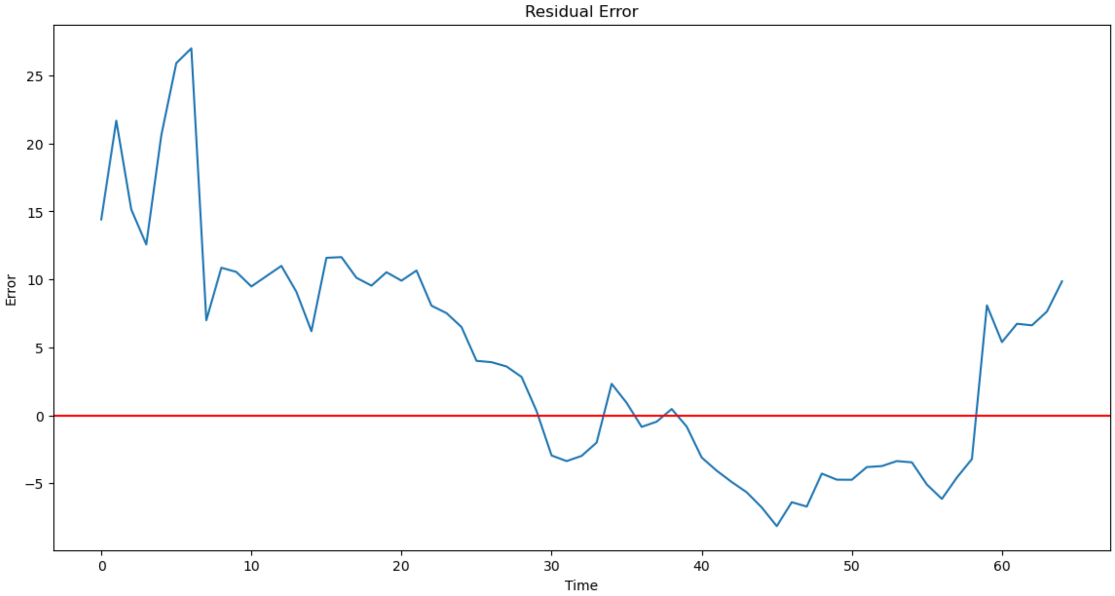
```
MAE: 7.3370143915940504
MSE: 84.27664042180878
RMSE: 9.180230956888218
```

```
[97]: {'MAE': 7.3370143915940504,
       'MSE': 84.27664042180878,
       'RMSE': 9.180230956888218}
```

```
[98]: #model.save('stock_model')
```

Display Model Residuals

```
[99]: # calculate residuals
residuals = y_test_original - predicted_prices_original

# plot residuals
plt.figure(figsize=(14, 7))
plt.plot(residuals)
plt.title('Residual Error')
plt.xlabel('Time')
plt.ylabel('Error')
plt.axhline(y=0, color='red')
plt.show()
```



5.0.1 Model with Rolling Averages

[100]:
```
df_stock_rolling.head()
```

[100]:
```
         Date    Open    High     Low    Volume  7 Day Rolling Avg   Close
0  2021-01-04  4.7500  4.7750  4.2875  40090000           4.594286  4.3125
1  2021-01-05  4.3375  4.5200  4.3075  19846000           4.594286  4.3425
2  2021-01-06  4.3350  4.7450  4.3325  24224800           4.594286  4.5900
3  2021-01-07  4.6175  4.8625  4.5050  24517200           4.594286  4.5200
4  2021-01-08  4.5450  4.5750  4.2700  25928000           4.594286  4.4225
```

[101]:
```
# new features to include in model
features2 = ['Open', 'High', 'Low', 'Close', 'Volume', '7 Day Rolling Avg']
NUM_FEATURES = len(features2)
features2 = df_stock_rolling[features2].values
```

[102]:
```
# feature scaling
scaler2 = MinMaxScaler(feature_range=(0,1))
scaled_data2 = scaler2.fit_transform(features2)
```

[103]:
```
SEQUENCE_LENGTH = 30  # You can adjust this value -> parameter tuning; default␣
 ↪30
X, y = create_sequences(scaled_data2, SEQUENCE_LENGTH)
```

[104]:
```
print(X.shape)
print(y.shape)
```

```
(222, 30, 6)
(222, 6)
```

[105]:
```
# Split the data into training and test sets (train on data until May 31st)
TRAIN_END_DATE = '2021-05-31'
train_indices = df_stock_rolling[df_stock_rolling['Date'] <= TRAIN_END_DATE].
 ↪index
X_train, y_train = X[:train_indices[-1]-SEQUENCE_LENGTH], y[:
 ↪train_indices[-1]-SEQUENCE_LENGTH]
```

[106]:
```
print(X_train.shape, y_train.shape)
```

```
(71, 30, 6) (71, 6)
```

[107]:
```
# create model with Stock data and 7 Day Rolling Average
# model2 = lstm_model(SEQUENCE_LENGTH, NUM_FEATURES)
```

[108]:
```
model2 = load_model('stock_rolling_model')
```

[109]:
```
# model2.compile(optimizer='adam', loss='mean_squared_error')
```

[110]:

```
# train the model
# model2.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)
```

[111]:

```
print(X_train[-1].shape)
```

```
(30, 6)
```

[112]:

```
# Get the last sequence from the training data
#Source: https://stackoverflow.com/questions/69753424/
 ↪keras-lstm-how-to-predict-beyond-validation-vs-predictions
last_sequence = X_train[-1].reshape((1, SEQUENCE_LENGTH, 6))
```

[113]:

```
last_sequence.shape
```

[113]:

```
(1, 30, 6)
```

[114]:

```
model2.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_4 (LSTM)               (None, 30, 50)            11400

 lstm_5 (LSTM)               (None, 50)                20200

 dense_4 (Dense)             (None, 25)                1275

 dense_5 (Dense)             (None, 1)                 26

=================================================================
Total params: 32901 (128.52 KB)
Trainable params: 32901 (128.52 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[115]:

```
# Create a list to hold predictions
predictions = []

# Predict future prices
for i in range(len(business_days)):
    # Get the prediction (scaled value)
    current_prediction = model2.predict(last_sequence, verbose=0)[0]

    # Append the prediction
    predictions.append(current_prediction)

    # Update the sequence
    last_sequence = np.roll(last_sequence, -1, axis=1)
    last_sequence[0, -1, 0] = current_prediction

# Inverse transform the predictions to get actual values
predicted_prices = scaler.inverse_transform(np.array(predictions).reshape(-1,␣
 ↪1))

# Create a DataFrame with the predicted stock prices and dates
predictions_df = pd.DataFrame({
    'Date': business_days,
    'Predicted_Close': predicted_prices.flatten()
})

# Show the prediction results
print(predictions_df)
```

```
         Date  Predicted_Close
0  2021-06-01        38.211109
1  2021-06-02        38.455814
2  2021-06-03        38.039856
3  2021-06-04        37.494091
4  2021-06-07        36.716011
..        …              …
60 2021-08-25        37.766880
61 2021-08-26        38.052567
62 2021-08-27        37.773609
63 2021-08-30        37.394672
64 2021-08-31        36.771214

[65 rows x 2 columns]
```

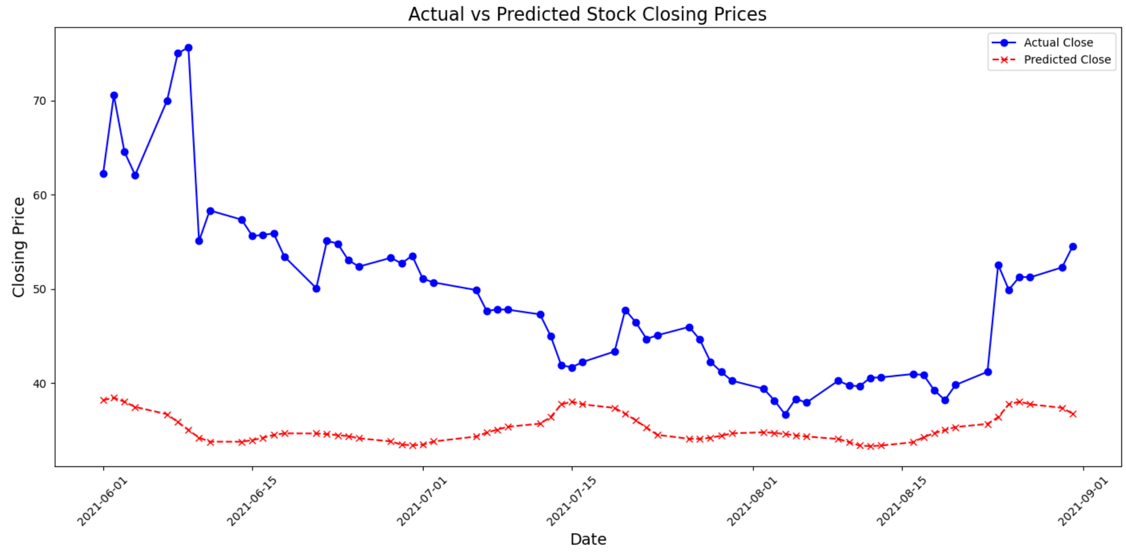
[116]:

```
# Continue from the previous predictions_df creation code

# Ensure the 'Date' columns in both DataFrames are in the same format
df_stock_rolling['Date'] = pd.to_datetime(df_stock_rolling['Date'])
predictions_df['Date'] = pd.to_datetime(predictions_df['Date'])

# Merge the predictions with the actual closing prices from 'df'
predictions_with_actuals_df = predictions_df.merge(df_stock_rolling[['Date',␣
 ↪'Close']], on='Date', how='left')

# Rename columns for clarity
predictions_with_actuals_df.rename(columns={'Close': 'Actual_Close'},␣
 ↪inplace=True)

# Show the DataFrame with predictions and actual closing prices
print(predictions_with_actuals_df)
```

```
          Date  Predicted_Close  Actual_Close
0   2021-06-01        38.211109     62.255001
1   2021-06-02        38.455814     70.559998
2   2021-06-03        38.039856     64.544998
3   2021-06-04        37.494091     62.090000
4   2021-06-07        36.716011     70.002502
..         …              …            …
60  2021-08-25        37.766880     49.912498
61  2021-08-26        38.052567     51.305000
62  2021-08-27        37.773609     51.237499
63  2021-08-30        37.394672     52.299999
64  2021-08-31        36.771214     54.560001

[65 rows x 3 columns]
```

[117]:

```python
import matplotlib.pyplot as plt

# Ensure the 'Date' column is in datetime format for proper plotting
predictions_with_actuals_df['Date'] = pd.
 ↪to_datetime(predictions_with_actuals_df['Date'])

# Setting the plot size for better readability
plt.figure(figsize=(14, 7))

# Plotting the actual closing prices
plt.plot(predictions_with_actuals_df['Date'],␣
 ↪predictions_with_actuals_df['Actual_Close'], label='Actual Close',␣
 ↪color='blue', marker='o')

# Plotting the predicted closing prices
plt.plot(predictions_with_actuals_df['Date'],␣
 ↪predictions_with_actuals_df['Predicted_Close'], label='Predicted Close',␣
 ↪color='red', linestyle='--', marker='x')

# Adding title and labels with font size adjustments
plt.title('Actual vs Predicted Stock Closing Prices', fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Closing Price', fontsize=14)

# Rotating date labels for better visibility
plt.xticks(rotation=45)

# Adding a legend to distinguish between actual and predicted values
plt.legend()

# Display the plot
plt.tight_layout()
plt.show()
```



[118]:

```
predictions_with_actuals_df.columns
```

[118]:

```
Index(['Date', 'Predicted_Close', 'Actual_Close'], dtype='object')
```

Display Evaluation Metrics

[119]:

```
y_test_original2 = predictions_with_actuals_df['Actual_Close']
predicted_prices_original2 = predictions_with_actuals_df['Predicted_Close']

model_evaluation(y_test_original2, predicted_prices_original2)
```

```
MAE: 13.92465897054913
MSE: 271.4392588442223
RMSE: 16.475413768528618
```

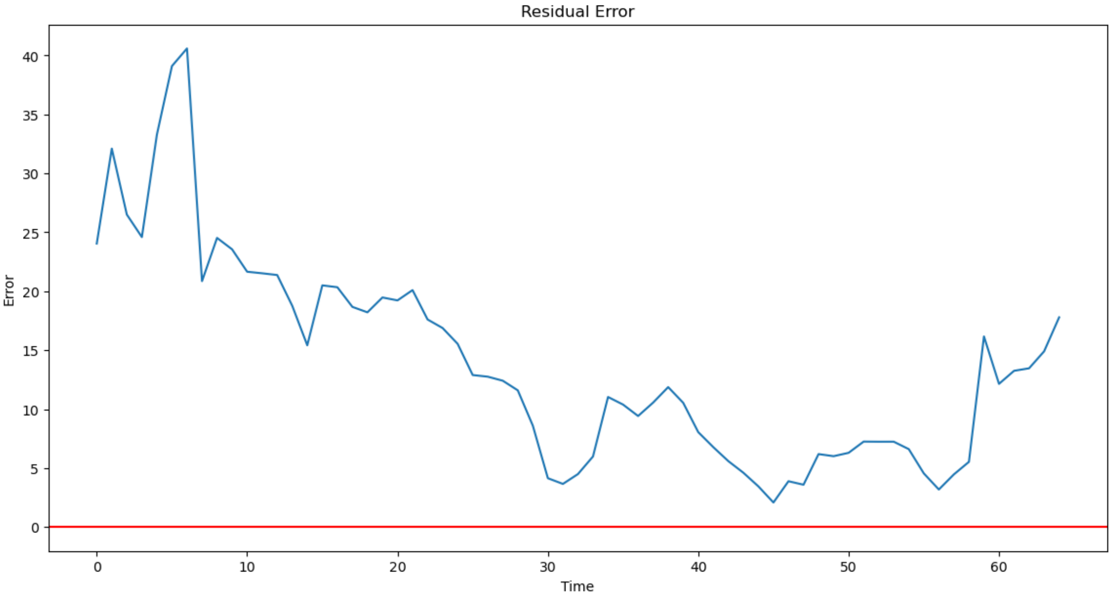
[119]:

```
{'MAE': 13.92465897054913,
 'MSE': 271.4392588442223,
 'RMSE': 16.475413768528618}
```

[120]:

```
# plot residuals
residuals = y_test_original2 - predicted_prices_original2

# plot residuals
plt.figure(figsize=(14, 7))
plt.plot(residuals)
plt.title('Residual Error')
plt.xlabel('Time')
```

```python
plt.ylabel('Error')
plt.axhline(y=0, color='red')
plt.show()
```



[121]:
```python
# save model
# model2.save('stock_rolling_model')
```

6 Model Fusion

6.0.1 Training and Evaluating Stock and TDM Studio Sentiment

[122]:
```python
features3 = ['Close', 'Open', 'High', 'Low', 'Volume',
       '7 Day Rolling Avg', 'Anger', 'Disgust', 'Fear', 'Sadness', 'Happiness',
       'Love', 'Surprise', 'Neutral', 'Other']
NUM_FEATURES = len(features3)
features3 = df_merged1[features3].values
```

[123]:
```python
scaler3 = MinMaxScaler(feature_range=(0,1))
scaled_data3 = scaler3.fit_transform(features3)
```

[124]:
```python
SEQUENCE_LENGTH = 30  # You can adjust this value -> parameter tuning
X, y = create_sequences(scaled_data3, SEQUENCE_LENGTH)
```

[125]:
```python
print(X.shape)
print(y.shape)
```

```
(222, 30, 15)
(222, 15)
```

[126]:
```
# Split the data into training and test sets (train on data until May 31st)
TRAIN_END_DATE = '2021-05-31'
train_indices = df_merged1[df_merged1['Date'] <= TRAIN_END_DATE].index
X_train, y_train = X[:train_indices[-1]-SEQUENCE_LENGTH], y[:
↪train_indices[-1]-SEQUENCE_LENGTH]
```

[127]:
```
print(X_train.shape, y_train.shape)
```

```
(71, 30, 15) (71, 15)
```

[128]:
```
# model
# model3 = lstm_model(SEQUENCE_LENGTH, NUM_FEATURES)
```

[129]:
```
model3 = load_model('stock_tdm_model')
```

[130]:
```
# model3.compile(optimizer='adam', loss='mean_squared_error')
```

[131]:
```
# model3.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)
```

[132]:
```
print(X_train[-1].shape)
```

```
(30, 15)
```

[133]:
```
# Get the last sequence from the training data
last_sequence = X_train[-1].reshape((1, SEQUENCE_LENGTH, 15))
```

[134]:
```
last_sequence.shape
```

[134]:
```
(1, 30, 15)
```

[135]:
```
model3.summary()
```

```
Model: "sequential_4"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_8 (LSTM)               (None, 30, 50)            13200

 lstm_9 (LSTM)               (None, 50)                20200

 dense_8 (Dense)             (None, 25)                1275

 dense_9 (Dense)             (None, 1)                 26

=================================================================
Total params: 34701 (135.55 KB)
Trainable params: 34701 (135.55 KB)
```

```
Non-trainable params: 0 (0.00 Byte)
________________________________________________________________
```

[136]:

```
# Create a list to hold predictions
predictions = []

# Predict future prices
for i in range(len(business_days)):
    # Get the prediction (scaled value)
    current_prediction = model3.predict(last_sequence, verbose=0)[0]

    # Append the prediction
    predictions.append(current_prediction)

    # Update the sequence
    last_sequence = np.roll(last_sequence, -1, axis=1)
    last_sequence[0, -1, 0] = current_prediction

# Inverse transform the predictions to get actual values
predicted_prices = scaler.inverse_transform(np.array(predictions).reshape(-1,␣
 ↪1))

# Create a DataFrame with the predicted stock prices and dates
predictions_df = pd.DataFrame({
    'Date': business_days,
    'Predicted_Close': predicted_prices.flatten()
})

# Show the prediction results
print(predictions_df)
```

```
         Date  Predicted_Close
0  2021-06-01        42.256397
1  2021-06-02        42.309738
2  2021-06-03        42.300713
3  2021-06-04        42.317627
4  2021-06-07        42.238552
..        …              …
60 2021-08-25        41.947495
61 2021-08-26        42.006321
62 2021-08-27        42.016747
63 2021-08-30        42.070389
64 2021-08-31        42.037766

[65 rows x 2 columns]
```

[137]:

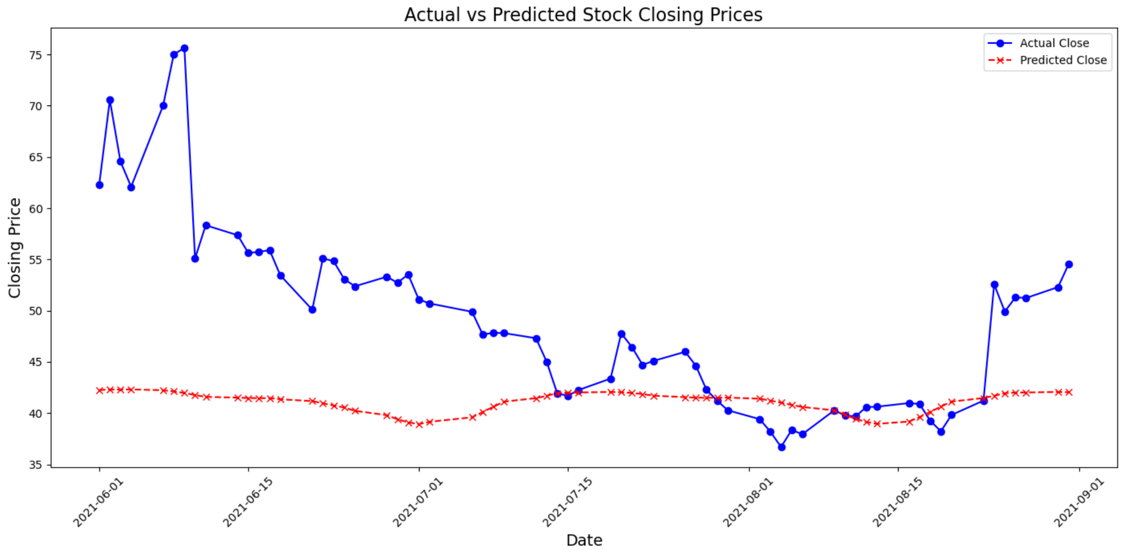
```
# Continue from the previous predictions_df creation code
# Ensure the 'Date' columns in both DataFrames are in the same format
df_merged1['Date'] = pd.to_datetime(df_merged1['Date'])
predictions_df['Date'] = pd.to_datetime(predictions_df['Date'])

# Merge the predictions with the actual closing prices from 'df'
predictions_with_actuals_df = predictions_df.merge(df_merged1[['Date',␣
↪'Close']], on='Date', how='left')

# Rename columns for clarity
predictions_with_actuals_df.rename(columns={'Close': 'Actual_Close'},␣
↪inplace=True)

# Show the DataFrame with predictions and actual closing prices
print(predictions_with_actuals_df)
```

```
         Date  Predicted_Close  Actual_Close
0  2021-06-01        42.256397     62.255001
1  2021-06-02        42.309738     70.559998
2  2021-06-03        42.300713     64.544998
3  2021-06-04        42.317627     62.090000
4  2021-06-07        42.238552     70.002502
..        …              …            …
60 2021-08-25        41.947495     49.912498
61 2021-08-26        42.006321     51.305000
62 2021-08-27        42.016747     51.237499
63 2021-08-30        42.070389     52.299999
64 2021-08-31        42.037766     54.560001

[65 rows x 3 columns]
```

[138]:

```
import matplotlib.pyplot as plt

# Ensure the 'Date' column is in datetime format for proper plotting
predictions_with_actuals_df['Date'] = pd.
↪to_datetime(predictions_with_actuals_df['Date'])

# Setting the plot size for better readability
plt.figure(figsize=(14, 7))

# Plotting the actual closing prices
plt.plot(predictions_with_actuals_df['Date'],␣
↪predictions_with_actuals_df['Actual_Close'], label='Actual Close',␣
↪color='blue', marker='o')

# Plotting the predicted closing prices
```

```
plt.plot(predictions_with_actuals_df['Date'],␣
 ↪predictions_with_actuals_df['Predicted_Close'], label='Predicted Close',␣
 ↪color='red', linestyle='--', marker='x')

# Adding title and labels with font size adjustments
plt.title('Actual vs Predicted Stock Closing Prices', fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Closing Price', fontsize=14)

# Rotating date labels for better visibility
plt.xticks(rotation=45)

# Adding a legend to distinguish between actual and predicted values
plt.legend()

# Display the plot
plt.tight_layout()
plt.show()
```



[139]:
```
predictions_with_actuals_df.columns
```

[139]:
```
Index(['Date', 'Predicted_Close', 'Actual_Close'], dtype='object')
```

Evaluation Metrics

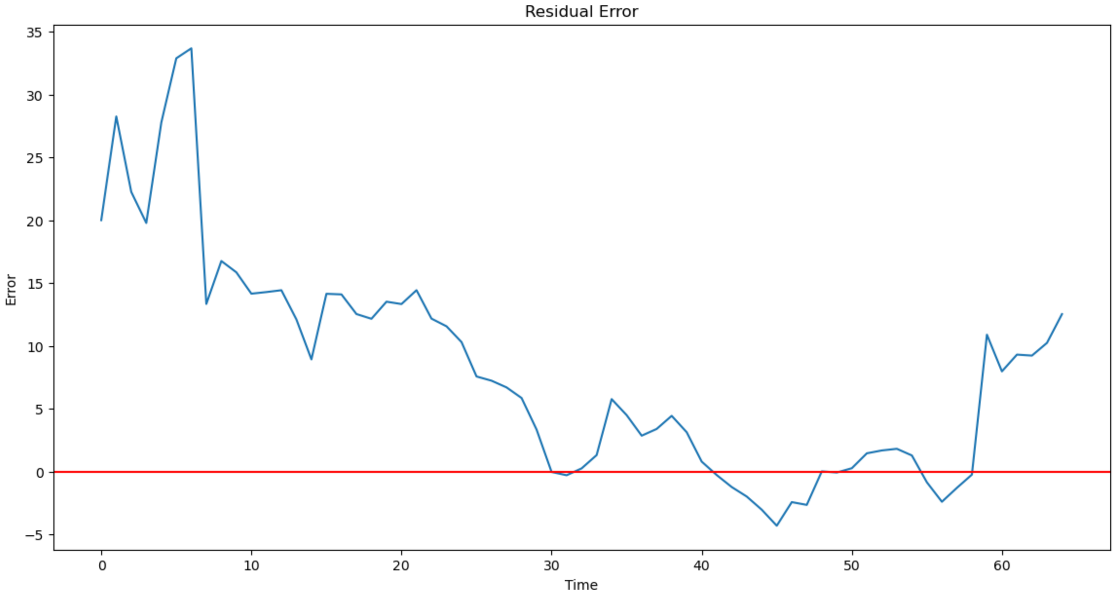
[140]:
```
# evaluation metrics
y_test_original3 = predictions_with_actuals_df['Actual_Close']
predicted_prices_original3 = predictions_with_actuals_df['Predicted_Close']
```

```
model_evaluation(y_test_original3, predicted_prices_original3)
```

```
MAE: 8.728209603402945
MSE: 142.4334536682908
RMSE: 11.934548741711637
```

```
[140]: {'MAE': 8.728209603402945,
        'MSE': 142.4334536682908,
        'RMSE': 11.934548741711637}
```

```
[141]: # plot residuals
residuals = y_test_original3 - predicted_prices_original3

# plot residuals
plt.figure(figsize=(14, 7))
plt.plot(residuals)
plt.title('Residual Error')
plt.xlabel('Time')
plt.ylabel('Error')
plt.axhline(y=0, color='red')
plt.show()
```



```
[142]: # save model
# model3.save('stock_tdm_model')
```

6.0.2 Training and Evaluating Stock and Google News Sentiment

[143]:
```
df_join_stock_news.head()
```

[143]:
```
         Date    Open    High     Low    Volume  7 Day Rolling Avg  compound  \
0  2021-01-04  4.7500  4.7750  4.2875  40090000           4.594286       0.0
1  2021-01-05  4.3375  4.5200  4.3075  19846000           4.594286       0.0
2  2021-01-06  4.3350  4.7450  4.3325  24224800           4.594286       0.0
3  2021-01-07  4.6175  4.8625  4.5050  24517200           4.594286       0.0
4  2021-01-08  4.5450  4.5750  4.2700  25928000           4.594286       0.0

   pos  neg   Close
0  0.0  0.0  4.3125
1  0.0  0.0  4.3425
2  0.0  0.0  4.5900
3  0.0  0.0  4.5200
4  0.0  0.0  4.4225
```

[144]:
```
df_join_stock_news.columns
```

[144]:
```
Index(['Date', 'Open', 'High', 'Low', 'Volume', '7 Day Rolling Avg',
       'compound', 'pos', 'neg', 'Close'],
      dtype='object')
```

[145]:
```
features4 = ['Close', 'Open', 'High', 'Low', 'Volume',
       '7 Day Rolling Avg', 'compound', 'pos', 'neg']
NUM_FEATURES = len(features4)
features4 = df_join_stock_news[features4].values
```

[146]:
```
scaler4 = MinMaxScaler(feature_range=(0,1))
scaled_data4 = scaler4.fit_transform(features4)
```

[147]:
```
SEQUENCE_LENGTH = 30  # You can adjust this value -> parameter tuning
X, y = create_sequences(scaled_data4, SEQUENCE_LENGTH)
```

[148]:
```
print(X.shape)
print(y.shape)
```

```
(222, 30, 9)
(222, 9)
```

[149]:
```
# Split the data into training and test sets (train on data until May 31st)
TRAIN_END_DATE = '2021-05-31'
train_indices = df_join_stock_news[df_join_stock_news['Date'] <=␣
 ↪TRAIN_END_DATE].index
X_train, y_train = X[:train_indices[-1]-SEQUENCE_LENGTH], y[:
 ↪train_indices[-1]-SEQUENCE_LENGTH]
```

[150]:

```python
print(X_train.shape, y_train.shape)
```

```
(71, 30, 9) (71, 9)
```

[151]:

```python
# model4 = lstm_model(SEQUENCE_LENGTH, NUM_FEATURES)
```

[152]:

```python
model4 = load_model('stock_google_model')
```

[153]:

```python
# model4.compile(optimizer='adam', loss='mean_squared_error')
```

[154]:

```python
# model4.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)
```

[155]:

```python
print(X_train[-1].shape)
```

```
(30, 9)
```

[156]:

```python
# Get the last sequence from the training data
last_sequence = X_train[-1].reshape((1, SEQUENCE_LENGTH, 9))
```

[157]:

```python
last_sequence.shape
```

[157]:

```
(1, 30, 9)
```

[158]:

```python
model4.summary()
```

```
Model: "sequential_8"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_16 (LSTM)              (None, 30, 50)            12000

 lstm_17 (LSTM)              (None, 50)                20200

 dense_16 (Dense)            (None, 25)                1275

 dense_17 (Dense)            (None, 1)                 26

=================================================================
Total params: 33501 (130.86 KB)
Trainable params: 33501 (130.86 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[159]:

```python
# Create a list to hold predictions
predictions = []

# Predict future prices
for i in range(len(business_days)):
    # Get the prediction (scaled value)
    current_prediction = model4.predict(last_sequence, verbose=0)[0]

    # Append the prediction
    predictions.append(current_prediction)

    # Update the sequence
    last_sequence = np.roll(last_sequence, -1, axis=1)
    last_sequence[0, -1, 0] = current_prediction

# Inverse transform the predictions to get actual values
predicted_prices = scaler.inverse_transform(np.array(predictions).reshape(-1,␣
 ↪1))

# Create a DataFrame with the predicted stock prices and dates
predictions_df = pd.DataFrame({
    'Date': business_days,
    'Predicted_Close': predicted_prices.flatten()
})

# Show the prediction results
print(predictions_df)
```

```
          Date  Predicted_Close
0   2021-06-01        33.773552
1   2021-06-02        33.652767
2   2021-06-03        33.127979
3   2021-06-04        32.586037
4   2021-06-07        32.179817
..         …              …
60  2021-08-25        32.012051
61  2021-08-26        31.977097
62  2021-08-27        31.656624
63  2021-08-30        31.363289
64  2021-08-31        31.196779

[65 rows x 2 columns]
```

[160]:

```
# Continue from the previous predictions_df creation code

# Ensure the 'Date' columns in both DataFrames are in the same format
df_join_stock_news['Date'] = pd.to_datetime(df_join_stock_news['Date'])
predictions_df['Date'] = pd.to_datetime(predictions_df['Date'])

# Merge the predictions with the actual closing prices from 'df'
predictions_with_actuals_df = predictions_df.merge(df_join_stock_news[['Date',␣
 ↪'Close']], on='Date', how='left')
```

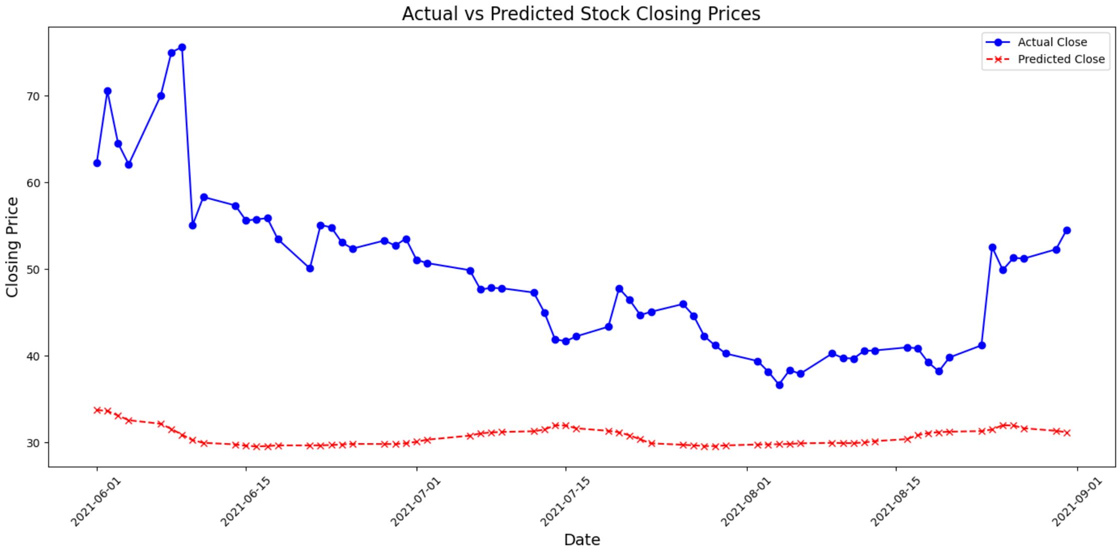
```
# Rename columns for clarity
predictions_with_actuals_df.rename(columns={'Close': 'Actual_Close'},␣
 ↪inplace=True)

# Show the DataFrame with predictions and actual closing prices
print(predictions_with_actuals_df)
```

```
         Date  Predicted_Close  Actual_Close
0  2021-06-01        33.773552     62.255001
1  2021-06-02        33.652767     70.559998
2  2021-06-03        33.127979     64.544998
3  2021-06-04        32.586037     62.090000
4  2021-06-07        32.179817     70.002502
..        …              …             …
60 2021-08-25        32.012051     49.912498
61 2021-08-26        31.977097     51.305000
62 2021-08-27        31.656624     51.237499
63 2021-08-30        31.363289     52.299999
64 2021-08-31        31.196779     54.560001

[65 rows x 3 columns]
```

[161]:

```
import matplotlib.pyplot as plt

# Ensure the 'Date' column is in datetime format for proper plotting
predictions_with_actuals_df['Date'] = pd.
 ↪to_datetime(predictions_with_actuals_df['Date'])

# Setting the plot size for better readability
plt.figure(figsize=(14, 7))

# Plotting the actual closing prices
plt.plot(predictions_with_actuals_df['Date'],␣
 ↪predictions_with_actuals_df['Actual_Close'], label='Actual Close',␣
 ↪color='blue', marker='o')

# Plotting the predicted closing prices
plt.plot(predictions_with_actuals_df['Date'],␣
 ↪predictions_with_actuals_df['Predicted_Close'], label='Predicted Close',␣
 ↪color='red', linestyle='--', marker='x')

# Adding title and labels with font size adjustments
plt.title('Actual vs Predicted Stock Closing Prices', fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Closing Price', fontsize=14)

# Rotating date labels for better visibility
```

```
plt.xticks(rotation=45)

# Adding a legend to distinguish between actual and predicted values
plt.legend()

# Display the plot
plt.tight_layout()
plt.show()
```



[162]:
```
predictions_with_actuals_df.columns
```

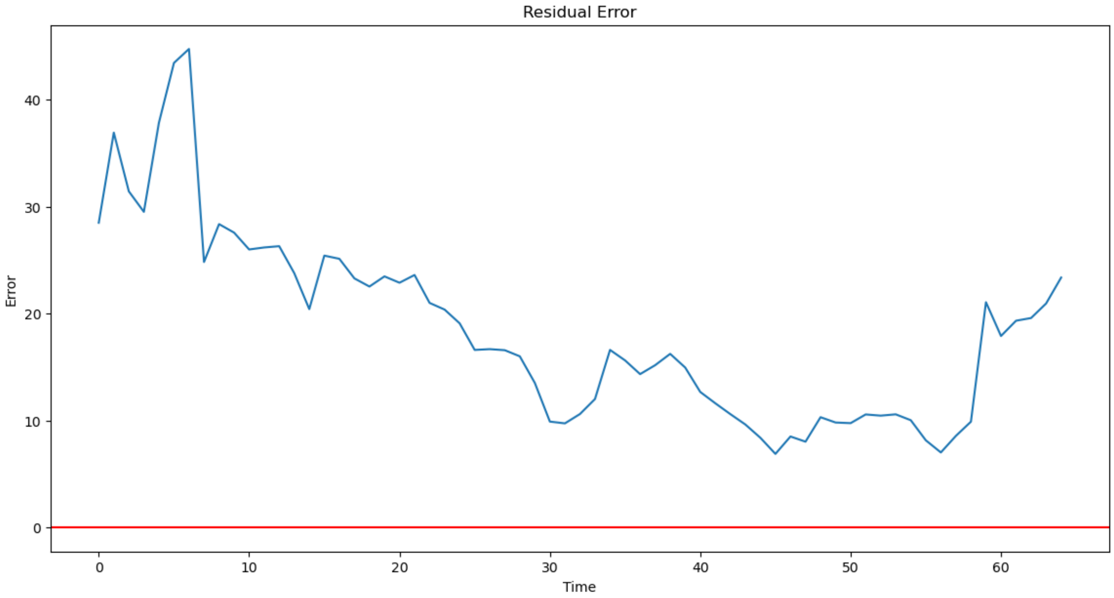
[162]:
```
Index(['Date', 'Predicted_Close', 'Actual_Close'], dtype='object')
```

[163]:
```
# evaluation metrics
y_test_original4 = predictions_with_actuals_df['Actual_Close']
predicted_prices_original4 = predictions_with_actuals_df['Predicted_Close']

model_evaluation(y_test_original4, predicted_prices_original4)
```

```
MAE: 18.469120969121846
MSE: 418.04327594922876
RMSE: 20.44610662080262
```

[163]:
```
{'MAE': 18.469120969121846,
 'MSE': 418.04327594922876,
 'RMSE': 20.44610662080262}
```

[164]:
```
residuals = y_test_original4 - predicted_prices_original4

# plot residuals
plt.figure(figsize=(14, 7))
plt.plot(residuals)
plt.title('Residual Error')
plt.xlabel('Time')
plt.ylabel('Error')
plt.axhline(y=0, color='red')
plt.show()
```



[165]:
```
# save model
# model4.save('stock_google_model')
```

6.0.3 Training and Evaluating Model with Stock Data and Sentiment from rGME

[166]:
```
df_rGME_sentiment_stock.head()
```

[166]:
```
         Date    Open    High     Low    Volume  7 Day Rolling Avg  compound  \
0  2021-01-04  4.7500  4.7750  4.2875  40090000           4.594286   0.98890
1  2021-01-05  4.3375  4.5200  4.3075  19846000           4.594286   0.11795
2  2021-01-06  4.3350  4.7450  4.3325  24224800           4.594286   0.38885
3  2021-01-07  4.6175  4.8625  4.5050  24517200           4.594286   0.00000
4  2021-01-08  4.5450  4.5750  4.2700  25928000           4.594286   0.00000

      pos     neg   Close
0  0.8660  0.0000  4.3125
```

```
1  0.1280  0.0575  4.3425
2  0.1885  0.0535  4.5900
3  0.0000  0.0000  4.5200
4  0.0000  0.0000  4.4225
```

```
[167]: df_rGME_sentiment_stock.columns
```

```
[167]: Index(['Date', 'Open', 'High', 'Low', 'Volume', '7 Day Rolling Avg',
              'compound', 'pos', 'neg', 'Close'],
             dtype='object')
```

```
[168]: features5 = ['Open', 'High', 'Low', 'Volume', '7 Day Rolling Avg', 'pos',␣
        ↪'neg', 'Close']
       NUM_FEATURES = len(features5)
       features5 = df_rGME_sentiment_stock[features5].values
```

```
[169]: scaler5 = MinMaxScaler(feature_range=(0,1))
       scaled_data5 = scaler5.fit_transform(features5)
```

```
[170]: SEQUENCE_LENGTH = 30  # You can adjust this value -> parameter tuning
       X, y = create_sequences(scaled_data5, SEQUENCE_LENGTH)
```

```
[171]: print(X.shape)
       print(y.shape)
```

```
(222, 30, 8)
(222, 8)
```

```
[172]: # Split the data into training and test sets (train on data until May 31st)
       TRAIN_END_DATE = '2021-05-31'
       train_indices = df_rGME_sentiment_stock[df_rGME_sentiment_stock['Date'] <=␣
        ↪TRAIN_END_DATE].index
       X_train, y_train = X[:train_indices[-1]-SEQUENCE_LENGTH], y[:
        ↪train_indices[-1]-SEQUENCE_LENGTH]
```

```
[173]: print(X_train.shape, y_train.shape)
```

```
(71, 30, 8) (71, 8)
```

```
[174]: # model5 = lstm_model(SEQUENCE_LENGTH, NUM_FEATURES)
```

```
[175]: model5 = load_model('stock_rGME_model')
```

```
[176]: # model5.compile(optimizer='adam', loss='mean_squared_error')
```

```
[177]: # model5.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)
```

```
[178]: print(X_train[-1].shape)
```

```
(30, 8)
```

[179]:
```
# Get the last sequence from the training data
last_sequence = X_train[-1].reshape((1, SEQUENCE_LENGTH, 8))
```

[180]:
```
last_sequence.shape
```

[180]:
```
(1, 30, 8)
```

[181]:
```
model5.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_22 (LSTM)              (None, 30, 50)            11800

 lstm_23 (LSTM)              (None, 50)                20200

 dense_22 (Dense)            (None, 25)                1275

 dense_23 (Dense)            (None, 1)                 26

=================================================================
Total params: 33301 (130.08 KB)
Trainable params: 33301 (130.08 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[182]:
```
# Create a list to hold predictions
predictions = []

# Predict future prices
for i in range(len(business_days)):
    # Get the prediction (scaled value)
    current_prediction = model5.predict(last_sequence, verbose=0)[0]

    # Append the prediction
    predictions.append(current_prediction)

    # Update the sequence
    last_sequence = np.roll(last_sequence, -1, axis=1)
    last_sequence[0, -1, 0] = current_prediction

# Inverse transform the predictions to get actual values
predicted_prices = scaler.inverse_transform(np.array(predictions).reshape(-1,␣
 ↪1))
```

```
# Create a DataFrame with the predicted stock prices and dates
predictions_df = pd.DataFrame({
    'Date': business_days,
    'Predicted_Close': predicted_prices.flatten()
})

# Show the prediction results
print(predictions_df)
```

```
         Date  Predicted_Close
0  2021-06-01        41.443176
1  2021-06-02        41.571697
2  2021-06-03        41.189522
3  2021-06-04        40.714966
4  2021-06-07        40.155785
..        …              …
60 2021-08-25        41.065037
61 2021-08-26        41.197727
62 2021-08-27        40.887226
63 2021-08-30        40.507992
64 2021-08-31        40.041340

[65 rows x 2 columns]
```

[183]:

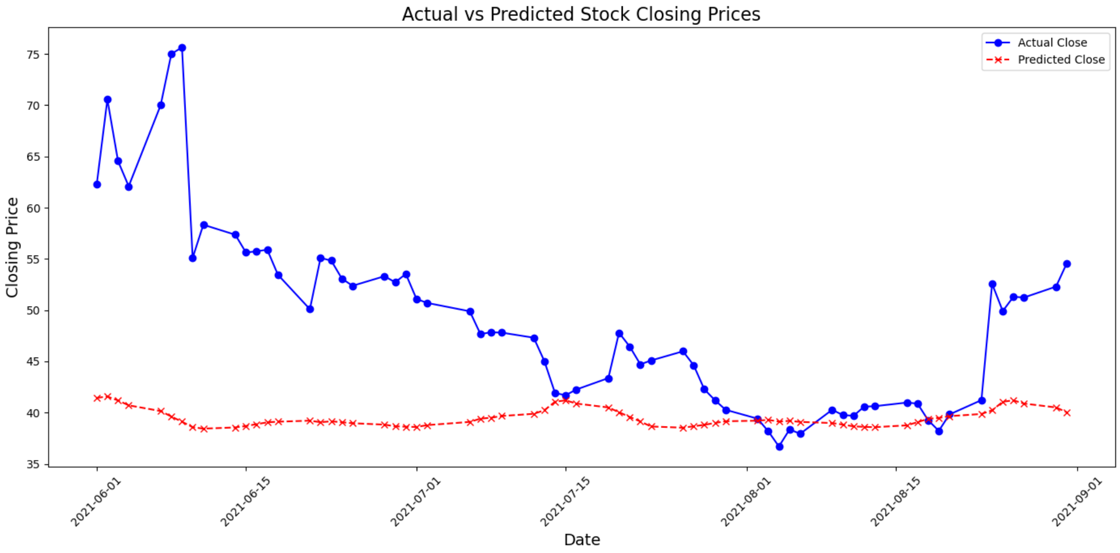
```
# Continue from the previous predictions_df creation code

# Ensure the 'Date' columns in both DataFrames are in the same format
df_rGME_sentiment_stock['Date'] = pd.
 ↪to_datetime(df_rGME_sentiment_stock['Date'])
predictions_df['Date'] = pd.to_datetime(predictions_df['Date'])

# Merge the predictions with the actual closing prices from 'df'
predictions_with_actuals_df = predictions_df.
 ↪merge(df_rGME_sentiment_stock[['Date', 'Close']], on='Date', how='left')

# Rename columns for clarity
predictions_with_actuals_df.rename(columns={'Close': 'Actual_Close'},␣
 ↪inplace=True)

# Show the DataFrame with predictions and actual closing prices
print(predictions_with_actuals_df)
```

```
         Date  Predicted_Close  Actual_Close
0  2021-06-01        41.443176     62.255001
1  2021-06-02        41.571697     70.559998
2  2021-06-03        41.189522     64.544998
3  2021-06-04        40.714966     62.090000
4  2021-06-07        40.155785     70.002502
```

```
..         …              …            …
60  2021-08-25         41.065037     49.912498
61  2021-08-26         41.197727     51.305000
62  2021-08-27         40.887226     51.237499
63  2021-08-30         40.507992     52.299999
64  2021-08-31         40.041340     54.560001

[65 rows x 3 columns]
```

[184]:

```
import matplotlib.pyplot as plt

# Ensure the 'Date' column is in datetime format for proper plotting
predictions_with_actuals_df['Date'] = pd.
 ↪to_datetime(predictions_with_actuals_df['Date'])

# Setting the plot size for better readability
plt.figure(figsize=(14, 7))

# Plotting the actual closing prices
plt.plot(predictions_with_actuals_df['Date'],␣
 ↪predictions_with_actuals_df['Actual_Close'], label='Actual Close',␣
 ↪color='blue', marker='o')

# Plotting the predicted closing prices
plt.plot(predictions_with_actuals_df['Date'],␣
 ↪predictions_with_actuals_df['Predicted_Close'], label='Predicted Close',␣
 ↪color='red', linestyle='--', marker='x')

# Adding title and labels with font size adjustments
plt.title('Actual vs Predicted Stock Closing Prices', fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Closing Price', fontsize=14)

# Rotating date labels for better visibility
plt.xticks(rotation=45)

# Adding a legend to distinguish between actual and predicted values
plt.legend()

# Display the plot
plt.tight_layout()
plt.show()
```



[185]:
```
predictions_with_actuals_df.columns
```

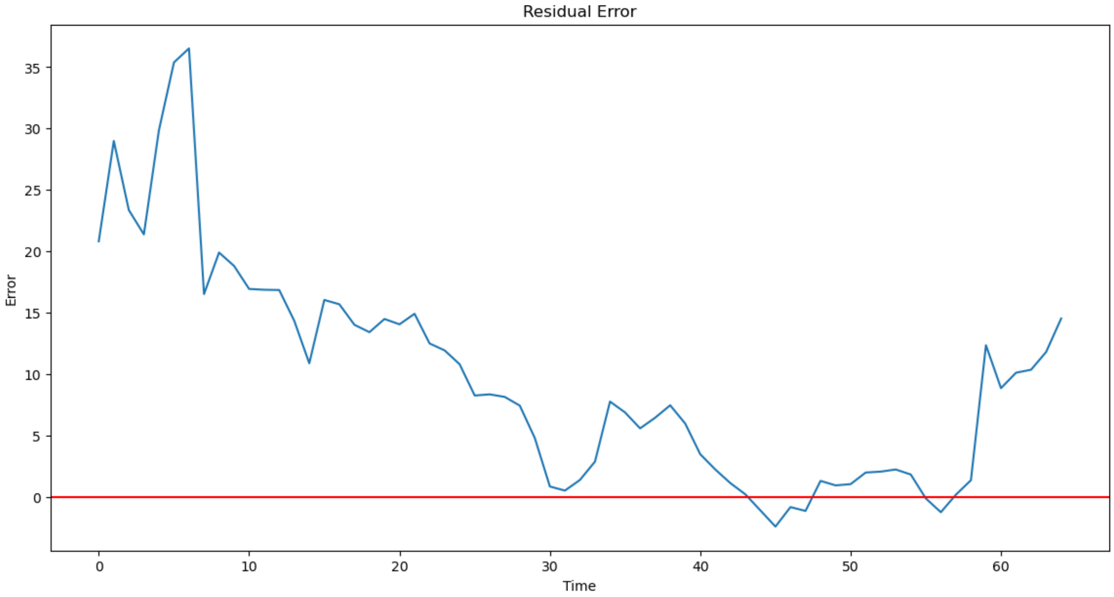
[185]:
```
Index(['Date', 'Predicted_Close', 'Actual_Close'], dtype='object')
```

[186]:
```
y_test_original5 = predictions_with_actuals_df['Actual_Close']
predicted_prices_original5 = predictions_with_actuals_df['Predicted_Close']

model_evaluation(y_test_original5, predicted_prices_original5)
```

```
MAE: 9.883221664888822
MSE: 173.5459132178784
RMSE: 13.173682598950014
```

[186]:
```
{'MAE': 9.883221664888822,
 'MSE': 173.5459132178784,
 'RMSE': 13.173682598950014}
```

[187]:
```
residuals = y_test_original5 - predicted_prices_original5

#plot residuals
plt.figure(figsize=(14, 7))
plt.plot(residuals)
plt.title('Residual Error')
plt.xlabel('Time')
plt.ylabel('Error')
plt.axhline(y=0, color='red')
plt.show()
```



[188]:
```
# save model
# model5.save('stock_rGME_model')
```

Transformer Model **** The Transformer elements of the code below were entirely generated by ChatGPT with edits and tuning by me to ensure that the input dimensions at various stages were correct for the data utilized.

See Appendix section below for detailed query.

Tuning: # of Features, Number of Attention Heads, Number of Epochs trained

[189]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Dropout,␣
 ↪GlobalAveragePooling1D
from tensorflow.keras.layers import MultiHeadAttention, LayerNormalization
from tensorflow.keras.optimizers import Adam
```

[190]:
```
# ChatGPT generated Transformer Encoder with MultiHead Attention,
# Feedforward layer (ReLu activation and fully connected), two normalization␣
 ↪layers, and two dropout layers
# Taxonomy: Input --> Attention --> Dropout --> Normalization --> Feedforward␣
 ↪--> Dropout --> Normalization --> Output

class TransformerEncoder(tf.keras.layers.Layer):
    def __init__(self, embed_dim, dense_dim, num_heads, **kwargs):
        super(TransformerEncoder, self).__init__(**kwargs)
        self.att = MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim)
        self.ffn = tf.keras.Sequential(
            [Dense(dense_dim, activation="relu"), Dense(6),]
        )
        self.layernorm1 = LayerNormalization(epsilon=1e-6)
        self.layernorm2 = LayerNormalization(epsilon=1e-6)
        self.dropout1 = Dropout(0.1)
        self.dropout2 = Dropout(0.1)

    def call(self, inputs, training):
        attn_output = self.att(inputs, inputs)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(inputs + attn_output)
        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output, training=training)
        return self.layernorm2(out1 + ffn_output)
```

```
[191]: def build_model(input_shape, num_heads, embed_dim, dense_dim):
           inputs = Input(shape=input_shape)
           x = TransformerEncoder(embed_dim, dense_dim, num_heads)(inputs)
           # Global Average Pooling for 1D from Transformer outputs
           x = GlobalAveragePooling1D()(x)
           x = Dropout(0.1)(x)
           outputs = Dense(1)(x)
           model = Model(inputs, outputs)
           return model
```

```
[192]: df_TDM_sentiment = df_merged1[['Anger',
              'Disgust', 'Fear', 'Sadness', 'Happiness', 'Love', 'Surprise',
              'Neutral', 'Other']].copy()
       df_combined_rGME_tdm_stock = df_rGME_sentiment_stock.join(df_TDM_sentiment)
       df_combined_rGME_tdm_stock.columns
```

```
[192]: Index(['Date', 'Open', 'High', 'Low', 'Volume', '7 Day Rolling Avg',
              'compound', 'pos', 'neg', 'Close', 'Anger', 'Disgust', 'Fear',
              'Sadness', 'Happiness', 'Love', 'Surprise', 'Neutral', 'Other'],
             dtype='object')
```

```
[193]: df_combined_rGME_tdm_stock = move_close_to_end(df_combined_rGME_tdm_stock)
```

```
[194]: df_combined_rGME_tdm_stock.columns
```

```
[194]: Index(['Date', 'Open', 'High', 'Low', 'Volume', '7 Day Rolling Avg',
              'compound', 'pos', 'neg', 'Anger', 'Disgust', 'Fear', 'Sadness',
              'Happiness', 'Love', 'Surprise', 'Neutral', 'Other', 'Close'],
             dtype='object')
```

```
[195]: # including features: Close Price; 7 Day Rolling Average; rGME compoound,␣
        ↪positive, and negative scores; and Opening Price
       features6 = ['Open', '7 Day Rolling Avg', 'compound', 'pos', 'neg', 'Close']
       num_features = len(features6)
       features6 = df_combined_rGME_tdm_stock[features6].values
```

```
[196]: scaler6 = MinMaxScaler(feature_range=(0,1))
       scaled_data6 = scaler5.fit_transform(features6)
       sequence_length = 30
```

```
[197]: X, y = create_sequences(scaled_data6, sequence_length)
```

```
[198]: print(X.shape, y.shape)
```

```
(222, 30, 6) (222, 6)
```

```
[199]: # Split the data into training and test sets (train on data until May 31st)
       TRAIN_END_DATE = '2021-05-31'
       train_indices = df_combined_rGME_tdm_stock[df_combined_rGME_tdm_stock['Date']␣
        ↪<= TRAIN_END_DATE].index
       X_train, y_train = X[:train_indices[-1]-SEQUENCE_LENGTH], y[:
        ↪train_indices[-1]-SEQUENCE_LENGTH]
```

```
[200]: print(X_train.shape, y_train.shape)
```

```
(71, 30, 6) (71, 6)
```

```
[201]: # Transformer Model with Stock, rGME, and TDM Studio Sentiment
       # model6 = load_model('transformer_model')
```

```
[202]: # train and test transformer model
       input_shape = (sequence_length, num_features)  # Adjust based on your data
       embed_dim = 32  # Embedding size for each token
       num_heads = 2  # Number of attention heads
       dense_dim = 32  # Hidden layer size in feed forward network inside transformer

       # model6 = build_model(input_shape, num_heads, embed_dim, dense_dim)
       # model6.compile(optimizer=Adam(learning_rate=1e-4), loss='mean_squared_error')

       # Assuming X_train and y_train are prepared
       # history = model6.fit(X_train, y_train, batch_size=32, epochs=200,␣
        ↪validation_split=0.2, verbose=0)
```

Load Transformer Model with Stock and rGME Sentiment data

```
[203]: model6 = load_model('transformer_model')
```

```
[204]: # Get the last sequence from the training data
       last_sequence = X_train[-1].reshape((1, sequence_length, 6))
```

[205]:
```
last_sequence.shape
```

[205]:
```
(1, 30, 6)
```

[206]:
```
model6.summary()
```

```
Model: "model_27"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_28 (InputLayer)       [(None, 30, 6)]           0

 transformer_encoder_27 (Tr  (None, 30, 6)             2180
 ansformerEncoder)

 global_average_pooling1d_2  (None, 6)                 0
 7 (GlobalAveragePooling1D)

 dropout_83 (Dropout)        (None, 6)                 0

 dense_83 (Dense)            (None, 1)                 7

=================================================================
Total params: 2187 (8.54 KB)
Trainable params: 2187 (8.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[207]:
```
# Create a list to hold predictions
predictions = []

# Predict future prices
for i in range(len(business_days)):
    # Get the prediction (scaled value)
    current_prediction = model6.predict(last_sequence, verbose=0)[0]

    # Append the prediction
    predictions.append(current_prediction)

    # Update the sequence
    last_sequence = np.roll(last_sequence, -1, axis=1)
    last_sequence[0, -1, 0] = current_prediction

# Inverse transform the predictions to get actual values
predicted_prices = scaler.inverse_transform(np.array(predictions).reshape(-1,␣
 ↪1))

# Create a DataFrame with the predicted stock prices and dates
```

```
predictions_df = pd.DataFrame({
    'Date': business_days,
    'Predicted_Close': predicted_prices.flatten()
})

# Show the prediction results
print(predictions_df)
```

```
         Date  Predicted_Close
0  2021-06-01        72.525536
1  2021-06-02        70.662308
2  2021-06-03        68.458336
3  2021-06-04        66.215660
4  2021-06-07        64.189880
..        …              …
60 2021-08-25        47.163063
61 2021-08-26        46.994282
62 2021-08-27        46.906700
63 2021-08-30        46.945496
64 2021-08-31        47.111336

[65 rows x 2 columns]
```

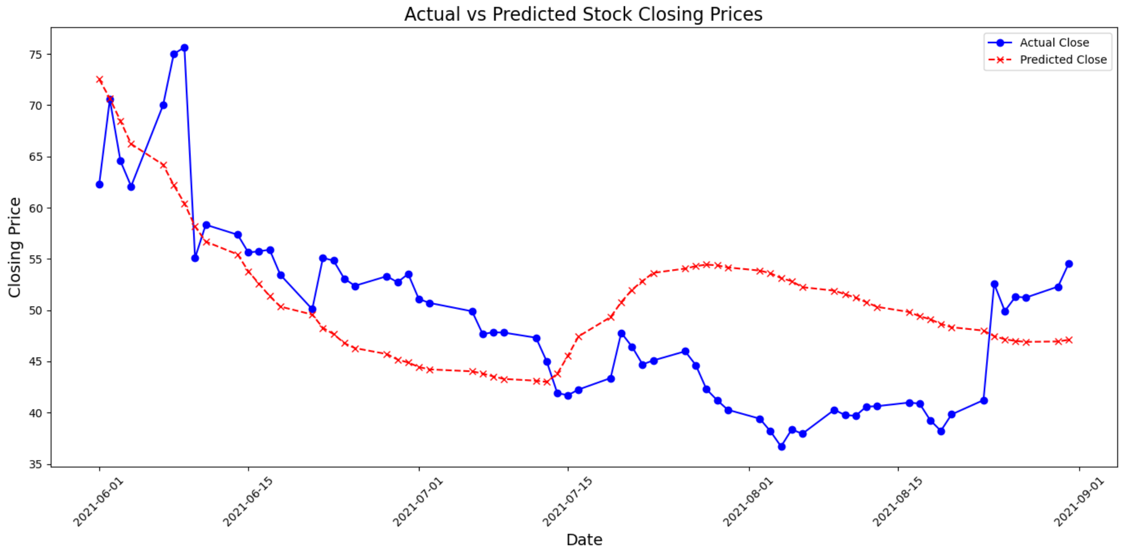
[208]:
```
# Continue from the previous predictions_df creation code

# Ensure the 'Date' columns in both DataFrames are in the same format
df_rGME_sentiment_stock['Date'] = pd.
 ↪to_datetime(df_rGME_sentiment_stock['Date'])
predictions_df['Date'] = pd.to_datetime(predictions_df['Date'])

# Merge the predictions with the actual closing prices from 'df'
predictions_with_actuals_df = predictions_df.
 ↪merge(df_rGME_sentiment_stock[['Date', 'Close']], on='Date', how='left')

# Rename columns for clarity
predictions_with_actuals_df.rename(columns={'Close': 'Actual_Close'},␣
 ↪inplace=True)

# Show the DataFrame with predictions and actual closing prices
print(predictions_with_actuals_df)
```

```
         Date  Predicted_Close  Actual_Close
0  2021-06-01        72.525536     62.255001
1  2021-06-02        70.662308     70.559998
2  2021-06-03        68.458336     64.544998
3  2021-06-04        66.215660     62.090000
4  2021-06-07        64.189880     70.002502
..        …              …            …
```

```
60 2021-08-25        47.163063     49.912498
61 2021-08-26        46.994282     51.305000
62 2021-08-27        46.906700     51.237499
63 2021-08-30        46.945496     52.299999
64 2021-08-31        47.111336     54.560001

[65 rows x 3 columns]
```

[209]:

```
import matplotlib.pyplot as plt

# Ensure the 'Date' column is in datetime format for proper plotting
predictions_with_actuals_df['Date'] = pd.
 ↪to_datetime(predictions_with_actuals_df['Date'])

# Setting the plot size for better readability
plt.figure(figsize=(14, 7))

# Plotting the actual closing prices
plt.plot(predictions_with_actuals_df['Date'],␣
 ↪predictions_with_actuals_df['Actual_Close'], label='Actual Close',␣
 ↪color='blue', marker='o')

# Plotting the predicted closing prices
plt.plot(predictions_with_actuals_df['Date'],␣
 ↪predictions_with_actuals_df['Predicted_Close'], label='Predicted Close',␣
 ↪color='red', linestyle='--', marker='x')

# Adding title and labels with font size adjustments
plt.title('Actual vs Predicted Stock Closing Prices', fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Closing Price', fontsize=14)

# Rotating date labels for better visibility
plt.xticks(rotation=45)

# Adding a legend to distinguish between actual and predicted values
plt.legend()

# Display the plot
plt.tight_layout()
plt.show()
```



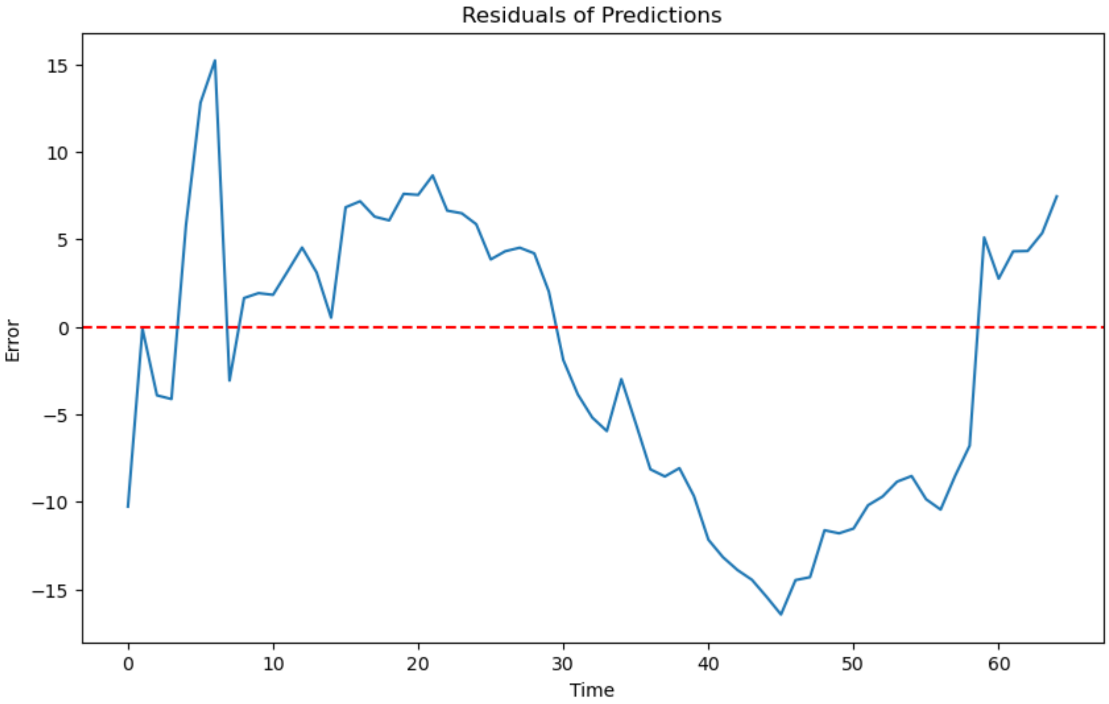
[210]:

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error

y_test_original6 = predictions_with_actuals_df['Actual_Close']
predicted_prices_original6 = predictions_with_actuals_df['Predicted_Close']

mae = mean_absolute_error(y_test_original6, predicted_prices_original6)
mse = mean_squared_error(y_test_original6, predicted_prices_original6)
rmse = np.sqrt(mse)

#Source: https://www.freecodecamp.org/news/
 ↪python-print-variable-how-to-print-a-string-and-variable/
print(f'MAE: {mae}')
print(f'MSE: {mse}')
print(f'RMSE: {rmse}')
```

```
MAE: 7.249460867367788
MSE: 68.9766142124163
RMSE: 8.30521608463117
```

[211]:

```python
residuals = y_test_original6 - predicted_prices_original6

plt.figure(figsize=(10,6))
plt.plot(residuals)
plt.title('Residuals of Predictions')
plt.xlabel('Time')
plt.ylabel('Error')
plt.axhline(y=0, color='red', linestyle='--')
plt.show()
```



[212]:
```
# save model
# model6.save('transformer_model')
```

Saving Initial Models

[213]:
```
# save models
# model.save('stock_model')
# model2.save('stock_rolling_model')
# model3.save('stock_tdm_model')
# model4.save('stock_google_model')
# model5.save('stock_rGME_model')
# model6.save('transformer_model')
```

Load Models for Comparison

[214]:
```
# # Stock only model
# model = load_model('stock_model')

# # Stock with 7 Day Rolling Average
# model2 = load_model('stock_rolling_model')

# # Stock with TDM Studio Sentiment
# model3 = load_model('stock_tdm_model')

# # Stock with Scraped Google News Headlines
# model4 = load_model('stock_google_model')

# # Stock with rGME Sentiment
# model5 = load_model('stock_rGME_model')

# # Transformer Model with Stock, rGME, and TDM Studio Sentiment
# model6 = load_model('transformer_model')
```

[215]:

```
# Stock only model
print("Stock only model: ")
model1_eval = model_evaluation(y_test=y_test_original,␣
 ↪predictions=predicted_prices_original)

print()
print("***Note: all of the following models utilize the 7 Day Rolling Average")
print()
# Stock with 7 Day Rolling Average
print("Stock with 7 Day Rolling Average model: ")
model2_eval = model_evaluation(y_test=y_test_original2,␣
 ↪predictions=predicted_prices_original2)

# Stock with TDM Studio Sentiment
print("Stock with TDM Studio Sentiment data: ")
model3_eval = model_evaluation(y_test=y_test_original3,␣
 ↪predictions=predicted_prices_original3)

# Stock with Scraped Google News Headlines
print("Stock with scraped Google News Headlines: " )
model4_eval = model_evaluation(y_test=y_test_original4,␣
 ↪predictions=predicted_prices_original4)

# Stock with rGME Sentiment
print("Stock with rGME Compound Sentiment scores: ")
model5_eval = model_evaluation(y_test=y_test_original5,␣
 ↪predictions=predicted_prices_original5)

# Transformer Model with Stock, rGME, and TDM Studio Sentiment
print("Transformer Model with Stock, rGME, and TDM Studio Sentiment data: ")
model6_eval = model_evaluation(y_test=y_test_original6,␣
 ↪predictions=predicted_prices_original6)
```

```
Stock only model:
MAE: 7.3370143915940504
MSE: 84.27664042180878
RMSE: 9.180230956888218
```

```
***Note: all of the following models utilize the 7 Day Rolling Average

Stock with 7 Day Rolling Average model:
MAE: 13.92465897054913
MSE: 271.4392588442223
RMSE: 16.475413768528618

Stock with TDM Studio Sentiment data:
MAE: 8.728209603402945
MSE: 142.4334536682908
RMSE: 11.934548741711637

Stock with scraped Google News Headlines:
MAE: 18.469120969121846
MSE: 418.04327594922876
RMSE: 20.44610662080262

Stock with rGME Compound Sentiment scores:
MAE: 9.883221664888822
MSE: 173.5459132178784
RMSE: 13.173682598950014

Transformer Model with Stock, rGME, and TDM Studio Sentiment data:
MAE: 7.249460867367788
MSE: 68.9766142124163
RMSE: 8.30521608463117
```

[216]:

```python
# create dataframe of results
df_results = pd.DataFrame([model1_eval, model2_eval, model3_eval, model4_eval,␣
 ↪model5_eval, model6_eval],
                           index=['Model 1', 'Model 2', 'Model 3', 'Model 4',␣
 ↪'Model 5', 'Model 6'])
df_results.to_csv('results.csv')
```

6.1 Conclusions

The Transformer Model trained using Stock price data, 7 Day Closing Price Rolling Average, and Sentiment from rGME yielded the lowest Mean Absolute Error, Mean Squared Error, and Root Mean Squared Error on the test set including the date range 01JUN-31AUG2021. Therefore it is currently the best model followed closely by the LSTM model utilizing only Stock Price data.

7 GameStop Short Squeeze and Model Adaptation

7.1 Event Analysis

[217]:

```
df_rGME.head()
```

[217]:

```
       id                                              title  \
0  kqfajb              You NEED to see this about GME
1  kqjh2t                     Short Squeeze Incoming
2  kqvp7l  THIS CONVINCED ME TO ALL IN  GME (EXTREME PUMP…
3  krcwch  You already know what we must do brothers and …
4  krnthg                          ICR conference (11th Jan)

                                                 url  score            author  \
0  https://www.reddit.com/r/GME/comments/kqfajb/y…    1.0      TitsDownOnly
1  /r/wallstreetbets/comments/kqcwdo/gamestops_gr…    1.0       zoomermoney
2  https://www.reddit.com/r/GME/comments/kqvp7l/t…    1.0      TitsDownOnly
3  /r/wallstreetbets/comments/kr98ym/gme_gang_we_…    1.0  dontforgettolive
4  https://www.reddit.com/r/GME/comments/krnthg/i…    1.0           nicky94

   num_comments        date flair  compound    neg  …  _poss  _intj _appos  \
0           9.0  2021-01-04   NaN    0.9872  0.000  …      0      0      6
1           1.0  2021-01-04   NaN    0.9906  0.000  …      0      0      3
2           6.0  2021-01-05   NaN    0.5319  0.000  …      0      0      0
3           4.0  2021-01-05   NaN   -0.2960  0.115  …      1      0      0
4          10.0  2021-01-06   NaN    0.0000  0.000  …      0      0      0

  _npadvmod  _predet  _case  _expl  _oprd  _dative  _nmod
0         0        0      0      0      0        0      0
1         0        0      0      0      0        0      0
2         0        0      0      0      0        0      0
3         0        0      0      0      0        0      0
4         1        0      0      0      0        0      0

[5 rows x 73 columns]
```

[218]:

```
# get rGME subset for January 2021
df_rGME_Jan = df_rGME[df_rGME['date'] <= '2021-01-31']
```

[219]:

```
df_rGME_Jan.describe()
```

[219]:

```
             score  num_comments     compound          neg          neu  \
count  6947.000000   6947.000000  6947.000000  6947.000000  6947.000000
mean      3.885994      5.280985     0.134468     0.065471     0.788413
std      31.667602     32.030128     0.423979     0.147826     0.258419
min       0.000000      0.000000    -0.973100     0.000000     0.000000
25%       1.000000      0.000000     0.000000     0.000000     0.633000
50%       1.000000      1.000000     0.000000     0.000000     0.884000
```

```
75%         1.000000      4.000000     0.420100     0.000000     1.000000
max      1607.000000   1610.000000     0.999900     1.000000     1.000000

               pos   char_count   word_count  word_count_cln  stopword_count  \
count  6947.000000  6947.000000  6947.000000     6947.000000     6947.000000
mean      0.146116    49.822657    11.998848        5.739744        4.611775
std       0.233944    45.351044    10.933535        5.393626        5.195043
min       0.000000     1.000000     1.000000        0.000000        0.000000
25%       0.000000    20.000000     5.000000        2.000000        1.000000
50%       0.000000    36.000000     9.000000        4.000000        3.000000
75%       0.244000    64.000000    15.000000        7.000000        6.000000
max       1.000000   303.000000   104.000000       81.000000       42.000000

       …        _poss        _intj       _appos    _npadvmod      _predet  \
count  …  6947.000000  6947.000000  6947.000000  6947.000000  6947.000000
mean   …     0.147690     0.042896     0.156039     0.159061     0.011660
std    …     0.418488     0.226141     0.751317     0.462725     0.108689
min    …     0.000000     0.000000     0.000000     0.000000     0.000000
25%    …     0.000000     0.000000     0.000000     0.000000     0.000000
50%    …     0.000000     0.000000     0.000000     0.000000     0.000000
75%    …     0.000000     0.000000     0.000000     0.000000     0.000000
max    …     4.000000     4.000000    33.000000     9.000000     2.000000

             _case        _expl        _oprd      _dative        _nmod
count  6947.000000  6947.000000  6947.000000  6947.000000  6947.000000
mean      0.013675     0.011372     0.009069     0.016698     0.160069
std       0.119807     0.110036     0.162947     0.135783     0.552199
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000     0.000000     0.000000
75%       0.000000     0.000000     0.000000     0.000000     0.000000
max       2.000000     2.000000    11.000000     2.000000    14.000000

[8 rows x 63 columns]
```

[220]:

```python
grouping = df_rGME_Jan.groupby("date").count()
grouping = grouping.sort_values("id", ascending=False)
grouping = pd.DataFrame(grouping)
grouping = grouping.reset_index()
grouping.head(10)
```

[220]:

```
         date    id  title   url  score  author  num_comments  flair  \
0  2021-01-28  2208   2208  2208   2208    2208          2208      0
1  2021-01-29  1549   1549  1549   1549    1549          1549      0
2  2021-01-27  1221   1221  1221   1221    1221          1221      0
3  2021-01-31   591    591   591    591     591           591      0
4  2021-01-30   577    577   577    577     577           577      0
```

```
5  2021-01-25   308    308   308    308     308           308      0
6  2021-01-26   164    164   164    164     164           164      0
7  2021-01-22    98     98    98     98      98            98      0
8  2021-01-24    60     60    60     60      60            60      0
9  2021-01-23    57     57    57     57      57            57      0

   compound   neg  ...  _poss  _intj  _appos  _npadvmod  _predet  _case  \
0      2208  2208  ...   2208   2208    2208       2208     2208   2208
1      1549  1549  ...   1549   1549    1549       1549     1549   1549
2      1221  1221  ...   1221   1221    1221       1221     1221   1221
3       591   591  ...    591    591     591        591      591    591
4       577   577  ...    577    577     577        577      577    577
5       308   308  ...    308    308     308        308      308    308
6       164   164  ...    164    164     164        164      164    164
7        98    98  ...     98     98      98         98       98     98
8        60    60  ...     60     60      60         60       60     60
9        57    57  ...     57     57      57         57       57     57

   _expl  _oprd  _dative  _nmod
0   2208   2208     2208   2208
1   1549   1549     1549   1549
2   1221   1221     1221   1221
3    591    591      591    591
4    577    577      577    577
5    308    308      308    308
6    164    164      164    164
7     98     98       98     98
8     60     60       60     60
9     57     57       57     57

[10 rows x 73 columns]
```

[221]:

```python
# mean number of comments in January
print("Mean # of Comments/Day: ", grouping['id'].mean())
# max number of comments in January
print("Max # of Comments (Day in January): ", grouping['id'].max())
# min number of comments in January
print("Min # of Comments (Day in January): ", grouping['id'].min())
```

```
Mean # of Comments/Day:  277.88
Max # of Comments (Day in January):  2208
Min # of Comments (Day in January):  1
```

[222]:

```python
alt.Chart(grouping).mark_line().encode(
    x=alt.X('date:T').title("Date"),
    y=alt.Y('id:Q').title("Number of Comments"),
    tooltip=['date:T','id:Q']
).properties(
    width=800,
    height=400,
    title="Number of Comments over Time"
).interactive()
```

```
[222]: alt.Chart(…)
```

7.1.1 Remove Emojis and Special Characters for Topic Modeling

```
[223]: # get text from comment titles
       s_titles = df_rGME_Jan['title']
       s_titles.head()
```

```
[223]: 0                 You NEED to see this about GME
       1                        Short Squeeze Incoming
       2    THIS CONVINCED ME TO ALL IN  GME (EXTREME PUMP…
       3    You already know what we must do brothers and …
       4                             ICR conference (11th Jan)
       Name: title, dtype: object
```

```
[224]: # regex to remove emojis and other special characters from post titles

       pattern = r'[^a-zA-Z0-9\s]'

       # Source: https://pandas.pydata.org/docs/reference/api/pandas.Series.replace.
        ↪html
       s_titles = s_titles.replace(pattern, '', regex=True)
       s_titles = s_titles.str.strip()
       df_rGME['title'] = s_titles
```

```
[225]: s_titles.to_csv("titles.csv",index=False)
```

8 Topic Modeling; Latent Dirichlet Allocation

Towards Data Science Article: Topic Modelling in Python with NLTK and Gensim Author: Susan Li

All code below copied from this article to perform topic modeling on prepared text data from rGME post titles in January 2021. Project changes include adjustments to handle prepared input data and adjustment to LDA model passes and topic number

8.1 Source for code: https://towardsdatascience.com/topic-modelling-in-python-with-nltk-and-gensim-4ef03213cd21

[226]:

```
# function to tokenize a line of input text, returns a list of tokens with␣
 ↪identifiers
import spacy
spacy.load('en_core_web_sm')
from spacy.lang.en import English
parser = English()
def tokenize(text):
    lda_tokens = []
    tokens = parser(text)
    for token in tokens:
        if token.orth_.isspace():
            continue
        elif token.like_url:
            lda_tokens.append('URL')
        elif token.orth_.startswith('@'):
            lda_tokens.append('SCREEN_NAME')
        else:
            lda_tokens.append(token.lower_)
    return lda_tokens
```

[227]:

```
# function to get the lemmas of token words
import nltk
nltk.download('wordnet')
from nltk.corpus import wordnet as wn
def get_lemma(word):
    lemma = wn.morphy(word)
    if lemma is None:
        return word
    else:
        return lemma

from nltk.stem.wordnet import WordNetLemmatizer
def get_lemma2(word):
    return WordNetLemmatizer().lemmatize(word)
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\patul\AppData\Roaming\nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

[228]:

```
# get stopwords to filter them out
nltk.download('stopwords')
en_stop = set(nltk.corpus.stopwords.words('english'))
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\patul\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[229]:

```
# function to tokenize text to prepare it for LDA function
def prepare_text_for_lda(text):
    tokens = tokenize(text)
    tokens = [token for token in tokens if len(token) > 4]
    tokens = [token for token in tokens if token not in en_stop]
    tokens = [get_lemma(token) for token in tokens]
    return tokens
```

[230]:

```
# read in lines from the titles.csv file and tokenize
# import random
text_data = []
with open('titles.csv') as f:
    for line in f:
        tokens = prepare_text_for_lda(line)
        # if random.random() > .99:
        #     #print(tokens)
        text_data.append(tokens)
```

[231]:

```
# create dictionary and corpus
from gensim import corpora
dictionary = corpora.Dictionary(text_data)
corpus = [dictionary.doc2bow(text) for text in text_data]
import pickle
pickle.dump(corpus, open('corpus.pkl', 'wb'))
dictionary.save('dictionary.gensim')
```

[232]:

```
# use LDA to find 10 lists of topics
import gensim
NUM_TOPICS = 10
ldamodel = gensim.models.ldamodel.LdaModel(corpus, num_topics = NUM_TOPICS,␣
  ↪id2word=dictionary, passes=50)
ldamodel.save('model5.gensim')
topics = ldamodel.print_topics(num_words=4)
for topic in topics:
    print(topic)
```

```
(0, '0.077*"stock" + 0.062*"robinhood" + 0.058*"today" + 0.053*"tomorrow"')
(1, '0.045*"hedge" + 0.038*"funds" + 0.035*"selling" + 0.026*"short"')
(2, '0.040*"strong" + 0.040*"people" + 0.037*"comment" + 0.022*"discord"')
(3, '0.104*"gamestop" + 0.042*"still" + 0.026*"trade" + 0.023*"support"')
(4, '0.060*"short" + 0.059*"squeeze" + 0.041*"shorts" + 0.040*"market"')
(5, '0.087*"going" + 0.048*"stocks" + 0.036*"trading" + 0.024*"try"')
(6, '0.049*"limit" + 0.032*"retard" + 0.030*"order" + 0.029*"robinhood"')
(7, '0.068*"share" + 0.041*"monday" + 0.040*"buy" + 0.035*"fucking"')
(8, '0.089*"holding" + 0.041*"think" + 0.035*"diamond" + 0.034*"hands"')
(9, '0.066*"buying" + 0.044*"everyone" + 0.033*"money" + 0.018*"advice"')
```

8.1.1 Event Analysis Conclusions

- January 25-31, 2021 were the most active days on rGME with over 100 posts each day.
- January 28, 2021 was the day with the most activity at 2208 posts.
- January 28, 2021 coincided with the day that Robinhood suspended trading of GME stock. Source: https://www.cnet.com/personal-finance/investing/robinhood-backlash-what-you-should-know-about-the-gamestop-stock-controversy/
- Monthly statistics:
  - Mean # of Comments/Day: 277.88
  - Max # of Comments (Day in January): 2208
  - Min # of Comments (Day in January): 1
- Topics:
  - (0, ‘0.038*“hedge” + 0.038*”robin” + 0.020*“nothing” + 0.020*”webull”’)
  - (1, ‘0.060*“short” + 0.045*”squeeze” + 0.031*“holding” + 0.030*”think”’)
  - (2, ‘0.025*“store” + 0.025*”today” + 0.013*“squeeze” + 0.013*”short”’)
  - (3, ‘0.036*“hours” + 0.036*”robinhood” + 0.024*“retard” + 0.024*”stock”’)
  - (4, ‘0.036*“share” + 0.036*”almost” + 0.019*“explain” + 0.019*”someone”’)
  - (5, ‘0.043*“today” + 0.043*”history” + 0.043*“ready” + 0.023*”happening”’)
  - (6, ‘0.040*“buy” + 0.027*”restrict” + 0.027*“pushhh” + 0.027*”seem”’)
  - (7, ‘0.061*“gamestop” + 0.046*”start” + 0.031*“second” + 0.016*”happen”’)
  - (8, ‘0.046*“still” + 0.031*”little” + 0.031*“panic” + 0.031*”stonks”’)
  - (9, ‘0.049*“fucking” + 0.033*”webull” + 0.017*“robinhood” + 0.017*”still”’)

8.1.2 Appendix A - AI Prompts and Responses

How might I use VADER to then perform sentiment analysis on my data? Query driven by visit to source: https://realpython.com/python-nltk-sentiment-analysis/

To perform sentiment analysis using VADER (Valence Aware Dictionary and sEntiment Reasoner) on your data in a Pandas DataFrame, you’ll need to follow these steps:

Step 1: Install VADER VADER is included in the nltk (Natural Language Toolkit) package. If you haven’t installed NLTK, you can do so via pip. You also need to download the VADER lexicon.

bash Copy code pip install nltk In your Python script, download the VADER lexicon:

python Copy code import nltk nltk.download(‘vader_lexicon’) Step 2: Import and Initialize VADER Sentiment Analyzer python Copy code from nltk.sentiment import SentimentIntensityAnalyzer

sia = SentimentIntensityAnalyzer() Step 3: Apply Sentiment Analysis to DataFrame Assuming you have a DataFrame df with a text column (e.g., ‘Text’), you can apply VADER to perform sentiment analysis:

python Copy code import pandas as pd

Example DataFrame data = {‘Text’: [‘I love sunny days in New York.’, ‘The movie was terrible and boring.’]} df = pd.DataFrame(data)

Function to get sentiment def get_sentiment(text): return sia.polarity_scores(text)

Apply the function to the DataFrame df[‘Sentiment’] = df[‘Text’].apply(get_sentiment)

Display the DataFrame print(df) This script will create a new column in your DataFrame called 'Sentiment', where each row contains a dictionary with scores for each of the four categories in VADER: 'neg' for negative, 'neu' for neutral, 'pos' for positive, and 'compound' for the aggregated score.

****If I have two date values for a start and end date, how would I get all of the values in between them? For example, if I have 2015-11-16 to 2015-11-22?**

[233]:
```
# from datetime import datetime, timedelta

# # Start and end dates
# start_date = datetime.strptime("2015-11-16", "%Y-%m-%d")
# end_date = datetime.strptime("2015-11-22", "%Y-%m-%d")

# # Generate all dates between start and end date
# date_generated = [start_date + timedelta(days=x) for x in range(0,␣
 ↪(end_date-start_date).days + 1)]

# # Convert to strings if needed
# date_strings = [date.strftime("%Y-%m-%d") for date in date_generated]

# date_strings
```

******What would be a simple Transformer model to implement for analysis of stock data and sentiment data? All of the data is already numeric in a dataframe.** For a simple Transformer model applied to stock and sentiment data, you can start with a basic architecture that includes an embedding layer (if necessary), Transformer encoder layers, and a final dense layer for prediction. Since your data is numeric and structured, you might skip the embedding layer or use it to project numeric features into a higher-dimensional space, depending on the complexity of your task.

Here's a basic outline of what such a Transformer model could look like using TensorFlow and Keras. This example assumes you're working on a regression task, such as predicting future stock prices:

Step 1: Import Necessary Libraries python Copy code import numpy as np import tensorflow as tf from tensorflow.keras.models import Model from tensorflow.keras.layers import Input, Dense, Dropout from tensorflow.keras.layers import MultiHeadAttention, LayerNormalization from tensorflow.keras.optimizers import Adam Step 2: Define Transformer Block A Transformer block consists of multi-head self-attention and position-wise feed-forward networks, along with layer normalization.

python Copy code class TransformerEncoder(tf.keras.layers.Layer): def **init**(self, embed_dim, dense_dim, num_heads, **kwargs): super(TransformerEncoder, self).__init__(**kwargs) self.att = MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim) self.ffn = tf.keras.Sequential([Dense(dense_dim, activation="relu"), Dense(embed_dim),]) self.layernorm1 = LayerNormalization(epsilon=1e-6) self.layernorm2 = LayerNormalization(epsilon=1e-6) self.dropout1 = Dropout(0.1) self.dropout2 = Dropout(0.1)

```
def call(self, inputs, training):
    attn_output = self.att(inputs, inputs)
    attn_output = self.dropout1(attn_output, training=training)
    out1 = self.layernorm1(inputs + attn_output)
    ffn_output = self.ffn(out1)
    ffn_output = self.dropout2(ffn_output, training=training)
    return self.layernorm2(out1 + ffn_output)
```

Step 3: Build the Model Assemble the Transformer model. This example builds a model that takes sequences of stock and sentiment data as input.

python Copy code def build_model(input_shape, num_heads, embed_dim, dense_dim): inputs = Input(shape=input_shape) x = TransformerEncoder(embed_dim, dense_dim, num_heads)(inputs) x = GlobalAveragePooling1D()(x) x = Dropout(0.1)(x) outputs = Dense(1)(x) model = Model(inputs, outputs) return model Step 4: Compile and Train the Model Compile and train your model. Adjust the learning rate, batch size, and number of epochs as necessary.

python Copy code input_shape = (sequence_length, num_features) # Adjust based on your data embed_dim = 32 # Embedding size for each token num_heads = 2 # Number of attention heads dense_dim = 32 # Hidden layer size in feed forward network inside transformer

model = build_model(input_shape, num_heads, embed_dim, dense_dim) model.compile(optimizer=Adam(learning_rate=1e-4), loss='mean_squared_error')

9 Assuming X_train and y_train are prepared

history = model.fit(X_train, y_train, batch_size=32, epochs=100, validation_split=0.2) Notes Input Shape: Adjust input_shape to match your data. It should be (sequence_length, num_features), where sequence_length is the number of time steps you're including in each sample, and num_features includes both stock and sentiment features. Feature Engineering: Even though the Transformer model can capture complex dependencies, thoughtful feature engineering can significantly impact model performance. Consider including lagged stock features, moving averages, and sentiment scores as part of your feature set. Hyperparameters: The model's performance can be sensitive to the choice of hyperparameters, including the number of attention heads, the dimensionality of the Transformer, and the learning rate. Experiment with different configurations to find the best setup for your specific dataset. This simple Transformer model provides a starting point for analyzing stock and sentiment data. Depending on your task's complexity and dataset size, you may need to adjust the model architecture, depth, and training parameters.

```
[ ]:
```