Overall

October 7, 2025

1 Multimodal Ensemble Architecture for Time Series Forecasting

- EAD
- Feature Engineering
- CNN-BiLSTM Modeling
- Transformer Modeling

1.1 Import Necessary Libraries

[1]:
```python
# Python Standard Libraries
import os
import csv
import math
import random
import unicodedata

# Data Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# NLP - NLTK
import nltk
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Scikit-learn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# PyTorch
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader

# TensorFlow / Keras
```

```
import tensorflow as tf
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import (
    Input, Dense, Dropout, LSTM, Bidirectional,
    Conv1D, Conv2D, MaxPooling1D, MaxPooling2D,
    Flatten, GlobalAveragePooling1D, LayerNormalization,
    MultiHeadAttention, Add
)
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau,␣
 ↪ModelCheckpoint
import tensorflow.keras.backend as K
from tensorflow.keras.losses import Huber

# XGBoost
import xgboost as xgb
from xgboost import XGBRegressor

# Shap
import shap
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /Users/yourth/nltk_data…
[nltk_data]   Package vader_lexicon is already up-to-date!
```

1.2 Data Collections

[2]:
```
raw_stocks = pd.read_csv('./data/raw/stock_yfinance.csv')
raw_tweets = pd.read_csv('./data/raw/stock_tweets.csv')
```

1.3 Exploratory Data Analysis & Data Preprocessing

1.3.1 1. stock_yfinance Dataset

[3]:
```
raw_stocks.head()
```

[3]:
```
         Date       Open       High        Low      Close  Adj Close  \
0  2015-01-02  27.847500  27.860001  26.837500  27.332500  24.320429
1  2015-01-05  27.072500  27.162500  26.352501  26.562500  23.635286
2  2015-01-06  26.635000  26.857500  26.157499  26.565001  23.637505
3  2015-01-07  26.799999  27.049999  26.674999  26.937500  23.968958
4  2015-01-08  27.307501  28.037500  27.174999  27.972500  24.889906

      Volume Stock Name
0  212818400       AAPL
1  257142000       AAPL
2  263188400       AAPL
3  160423600       AAPL
```

```
4  237458000       AAPL
```

[4]:
```
raw_stocks.tail()
```

[4]:
```
            Date       Open       High        Low      Close  Adj Close  \
6285  2019-12-24  27.890667  28.364668  27.512667  28.350000  28.350000
6286  2019-12-26  28.527332  28.898666  28.423332  28.729334  28.729334
6287  2019-12-27  29.000000  29.020666  28.407333  28.691999  28.691999
6288  2019-12-30  28.586000  28.600000  27.284000  27.646667  27.646667
6289  2019-12-31  27.000000  28.086000  26.805332  27.888666  27.888666

         Volume Stock Name
6285  120820500       TSLA
6286  159508500       TSLA
6287  149185500       TSLA
6288  188796000       TSLA
6289  154285500       TSLA
```

[ ]:
```
raw_stocks.describe()
```

[ ]:
```
              Open         High          Low        Close    Adj Close  \
count  6290.000000  6290.000000  6290.000000  6290.000000  6290.000000
mean     47.939449    48.373949    47.468337    47.943927    46.149473
std      28.802247    28.991926    28.561164    28.793088    27.501038
min       9.488000    10.331333     9.403333     9.578000     9.578000
25%      26.413126    26.652000    26.131500    26.448125    24.851683
50%      42.177500    42.528000    41.861500    42.233999    40.205023
75%      58.738500    59.248500    58.206749    58.749249    57.247396
max     159.449997   159.550003   158.220001   158.960007   151.738663

             Volume
count  6.290000e+03
mean   7.857451e+07
std    6.388470e+07
min    7.425600e+06
25%    3.243302e+07
50%    6.051200e+07
75%    1.035403e+08
max    6.488252e+08
```

[6]:
```
raw_stocks.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6290 entries, 0 to 6289
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Date        6290 non-null   object
 1   Open        6290 non-null   float64
 2   High        6290 non-null   float64
 3   Low         6290 non-null   float64
 4   Close       6290 non-null   float64
 5   Adj Close   6290 non-null   float64
 6   Volume      6290 non-null   int64
 7   Stock Name  6290 non-null   object
dtypes: float64(5), int64(1), object(2)
memory usage: 393.3+ KB
```

[7]:
```
raw_stocks['Stock Name'].unique()
```

[7]:
```
array(['AAPL', 'AMZN', 'GOOGL', 'MSFT', 'TSLA'], dtype=object)
```

[8]:
```
stocks = raw_stocks.copy()

stocks['Date'] = pd.to_datetime(stocks['Date'])
stocks
```

[8]:
```
           Date       Open       High        Low      Close  Adj Close  \
0    2015-01-02  27.847500  27.860001  26.837500  27.332500  24.320429
1    2015-01-05  27.072500  27.162500  26.352501  26.562500  23.635286
2    2015-01-06  26.635000  26.857500  26.157499  26.565001  23.637505
3    2015-01-07  26.799999  27.049999  26.674999  26.937500  23.968958
4    2015-01-08  27.307501  28.037500  27.174999  27.972500  24.889906
…           …        …        …        …        …        …        …
6285 2019-12-24  27.890667  28.364668  27.512667  28.350000  28.350000
6286 2019-12-26  28.527332  28.898666  28.423332  28.729334  28.729334
6287 2019-12-27  29.000000  29.020666  28.407333  28.691999  28.691999
6288 2019-12-30  28.586000  28.600000  27.284000  27.646667  27.646667
6289 2019-12-31  27.000000  28.086000  26.805332  27.888666  27.888666

         Volume Stock Name
0     212818400       AAPL
1     257142000       AAPL
2     263188400       AAPL
3     160423600       AAPL
4     237458000       AAPL
…           …        …
6285  120820500       TSLA
6286  159508500       TSLA
6287  149185500       TSLA
6288  188796000       TSLA
6289  154285500       TSLA

[6290 rows x 8 columns]
```

1.3.2 2. stock_tweets Dataset

[9]:
```
raw_tweets.head()
```

[9]:
```
              Tweet ID            Writer         UTC  \
0  550441509175443456  VisualStockRSRC  1420070457
1  550441672312512512      KeralaGuy77  1420070496
2  550441732014223360      DozenStocks  1420070510
3  550442977802207232     ShowDreamCar  1420070807
4  550443807834402816     i_Know_First  1420071005

                                               Tweet  Like Num Stock Name  \
0  lx21 made $10,008  on $AAPL -Check it out! htt…         1       AAPL
1  Insanity of today weirdo massive selling. $aap…         0       AAPL
2  S&P100 #Stocks Performance $HD $LOW $SBUX $TGT…         0       AMZN
3  $GM $TSLA: Volkswagen Pushes 2014 Record Recal…         1       TSLA
4  Swing Trading: Up To 8.91% Return In 14 Days h…         1       AAPL

                       Date
0  2015-01-01 00:00:57+00:00
1  2015-01-01 00:01:36+00:00
2  2015-01-01 00:01:50+00:00
3  2015-01-01 00:06:47+00:00
4  2015-01-01 00:10:05+00:00
```

[10]:
```
raw_tweets.tail()
```

[10]:
```
                    Tweet ID           Writer         UTC  \
3943871  1212159838882533376  ShortingIsFun  1577836401
3943872  1212160015332728833    Commuternyc  1577836443
3943873  1212160410692046849    MoriaCrypto  1577836537
3943874  1212160410692046849    MoriaCrypto  1577836537
3943875  1212160477159206912       treabase  1577836553

                                                     Tweet  Like Num  \
3943871  In 2020 I may start Tweeting out positive news…         1
3943872  Patiently Waiting for the no twitter sitter tw…         5
3943873  I don't discriminate. I own both $aapl and $ms…         1
3943874  I don't discriminate. I own both $aapl and $ms…         1
3943875  $AAPL #patent 10,522,475 Vertical interconnect…         0

        Stock Name                       Date
3943871       TSLA  2019-12-31 23:53:21+00:00
3943872       TSLA  2019-12-31 23:54:03+00:00
3943873       MSFT  2019-12-31 23:55:37+00:00
3943874       AAPL  2019-12-31 23:55:37+00:00
3943875       AAPL  2019-12-31 23:55:53+00:00
```

[11]:
```
raw_tweets.info(show_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3943876 entries, 0 to 3943875
Data columns (total 7 columns):
 #   Column      Non-Null Count    Dtype
---  ------      --------------    -----
 0   Tweet ID    3943876 non-null  int64
 1   Writer      3895635 non-null  object
 2   UTC         3943876 non-null  int64
 3   Tweet       3943876 non-null  object
 4   Like Num    3943876 non-null  int64
 5   Stock Name  3943876 non-null  object
 6   Date        3943876 non-null  object
dtypes: int64(3), object(4)
memory usage: 210.6+ MB
```

[12]:
```
tweets = raw_tweets.copy()

tweets = tweets.drop(columns=['Tweet ID', 'Writer', 'UTC', 'Like Num'])
tweets = tweets.dropna(subset=['Date', 'Tweet'])
tweets.isna().sum()
```

[12]:
```
Tweet         0
Stock Name    0
Date          0
dtype: int64
```

[13]:
```
# Make sure Date is in datetime format
tweets['Date'] = pd.to_datetime(tweets['Date'], errors='coerce')
tweets['Date'] = tweets['Date'].dt.date

tweets.head()
```

[13]:
```
                                               Tweet Stock Name        Date
0  1x21 made $10,008  on $AAPL -Check it out! htt…       AAPL  2015-01-01
1  Insanity of today weirdo massive selling. $aap…       AAPL  2015-01-01
2  S&P100 #Stocks Performance $HD $LOW $SBUX $TGT…       AMZN  2015-01-01
3  $GM $TSLA: Volkswagen Pushes 2014 Record Recal…       TSLA  2015-01-01
4  Swing Trading: Up To 8.91% Return In 14 Days h…       AAPL  2015-01-01
```

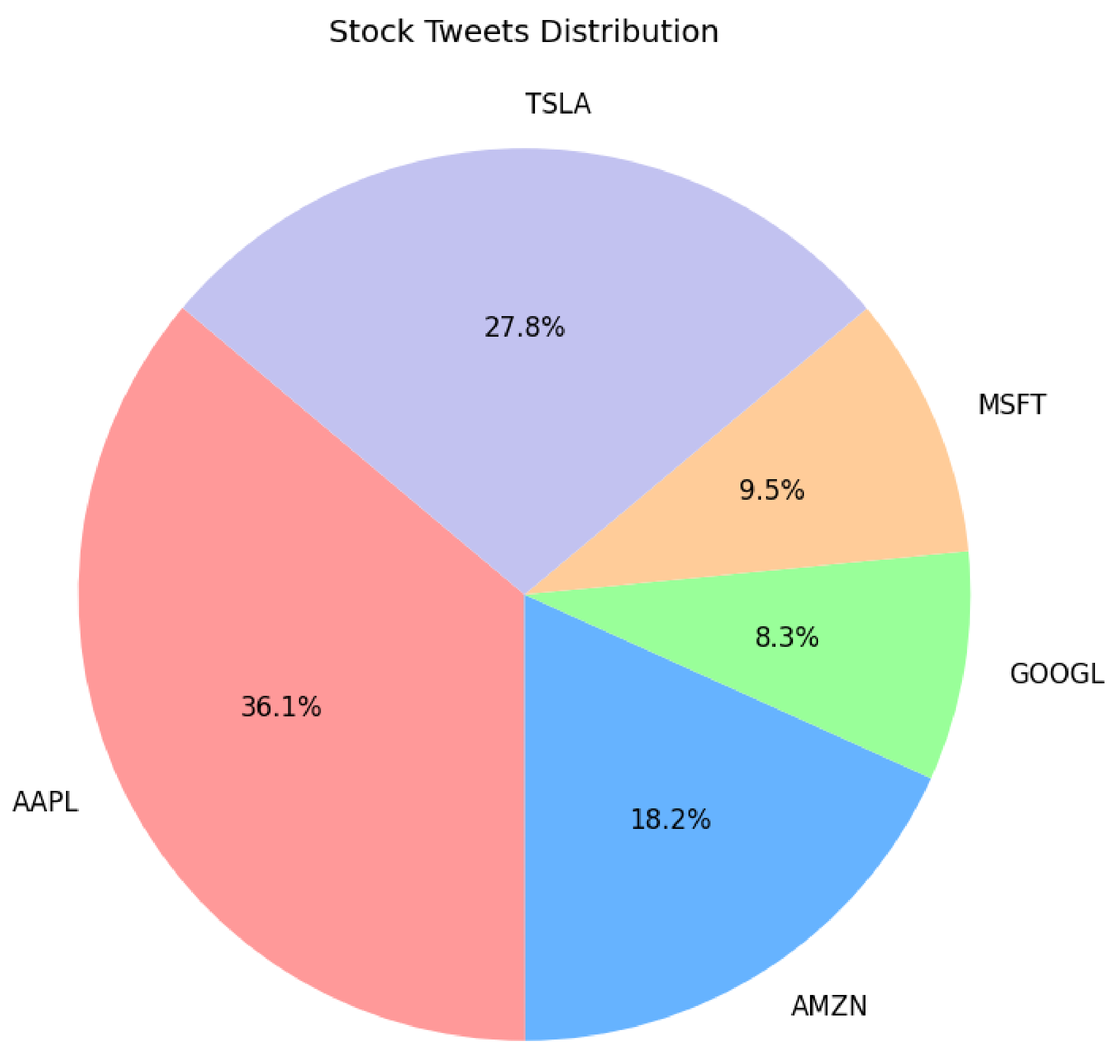
[14]:
```
tweets.info(show_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3943876 entries, 0 to 3943875
Data columns (total 3 columns):
 #   Column      Non-Null Count    Dtype
---  ------      --------------    -----
 0   Tweet       3943876 non-null  object
 1   Stock Name  3943876 non-null  object
 2   Date        3943876 non-null  object
dtypes: object(3)
memory usage: 90.3+ MB
```

[15]:
```
tweet_stock_count = raw_tweets['Stock Name'].value_counts()

perc = tweet_stock_count / tweet_stock_count.sum()
dstr = pd.DataFrame({'Count': tweet_stock_count, 'Percentage': perc}).
  ↪reset_index()
dstr = dstr.rename(columns={'index': 'Stock Name'})
dstr.head()
```

[15]:
```
  Stock Name    Count  Percentage
0       AAPL  1425013    0.361323
1       TSLA  1096868    0.278119
2       AMZN   718715    0.182236
3       MSFT   375711    0.095264
4      GOOGL   327569    0.083058
```

[16]:
```
# Set a threshold percentage
threshold = 0.02  # 2%
colors = ['#ff9999', '#66b3ff', '#99ff99', '#ffcc99', '#c2c2f0', '#ffb3e6',␣
  ↪'#c4e17f', '#76d7c4', '#f7c6c7']

# Create a new DataFrame grouping smaller slices
dstr['Grouped'] = dstr.apply(lambda row: 'Other' if row['Percentage'] <␣
  ↪threshold else row['Stock Name'], axis=1)

dstr_grouped = dstr.groupby(
    dstr['Grouped'].where(dstr['Grouped'] != 'Other', 'Other')
).agg({'Count': 'sum'}).reset_index()

# Plot
plt.figure(figsize=(8, 8))
plt.pie(
    dstr_grouped['Count'],
    labels=dstr_grouped['Grouped'],
    autopct='%1.1f%%',
    startangle=140,
    colors=colors,
    textprops={'fontsize': 12}
)
plt.title('Stock Tweets Distribution', fontsize=14, pad=30)
plt.axis('equal')
plt.show()
```



```
[17]: stocks.head()
```

```
[17]:          Date       Open       High        Low      Close  Adj Close  \
      0 2015-01-02  27.847500  27.860001  26.837500  27.332500  24.320429
      1 2015-01-05  27.072500  27.162500  26.352501  26.562500  23.635286
      2 2015-01-06  26.635000  26.857500  26.157499  26.565001  23.637505
      3 2015-01-07  26.799999  27.049999  26.674999  26.937500  23.968958
      4 2015-01-08  27.307501  28.037500  27.174999  27.972500  24.889906

            Volume Stock Name
      0  212818400       AAPL
      1  257142000       AAPL
      2  263188400       AAPL
```

```
3  160423600       AAPL
4  237458000       AAPL
```

```
[18]: tweets.head()
```

```
[18]:                                                Tweet Stock Name        Date
      0  lx21 made $10,008  on $AAPL -Check it out! htt…       AAPL  2015-01-01
      1  Insanity of today weirdo massive selling. $aap…       AAPL  2015-01-01
      2  S&P100 #Stocks Performance $HD $LOW $SBUX $TGT…       AMZN  2015-01-01
      3  $GM $TSLA: Volkswagen Pushes 2014 Record Recal…       TSLA  2015-01-01
      4  Swing Trading: Up To 8.91% Return In 14 Days h…       AAPL  2015-01-01
```

```
[19]: def find_last_trading_day(x, trading_days):
          """
              x  trading_days      x



                      NaT
          """
          eligible_days = trading_days[trading_days <= x]
          if not eligible_days.empty:
              return eligible_days.max()
          else:
              return pd.NaT

      def map_to_last_trading_day(tweet_dates, trading_days):
          """




              tweet_dates: Series  DatetimeIndex
              trading_days:



              Series
          """
          trading_days = pd.to_datetime(sorted(set(trading_days)))

          #   apply + lambda
          mapped_dates = tweet_dates.apply(lambda x: find_last_trading_day(x,␣
        ↪trading_days))

          #         NaT
          valid_mask = mapped_dates.notna()

          return mapped_dates[valid_mask]
```

```
[20]: #
      tweets['Date'] = pd.to_datetime(tweets['Date'])
      stocks['Date'] = pd.to_datetime(stocks['Date'])

      #
      trading_days = stocks['Date'].unique()

      #
      mapped_dates = map_to_last_trading_day(tweets['Date'], trading_days)

      #      tweets
      tweets = tweets.loc[mapped_dates.index].copy()
      tweets['Trading Date'] = mapped_dates
```

```
[21]: tweets['Date'] = tweets['Trading Date']
      tweets = tweets.drop(columns=['Trading Date'])  #

      tweets.head()
```

```
[21]:                                                  Tweet Stock Name       Date
      628  S&P100 #Stocks Performance $HD $LOW $SBUX $TGT…       AMZN 2015-01-02
      629  Will Audi's Electric Q7 Cause $TSLA Model X Ba…       TSLA 2015-01-02
      630  Either way you're a winnah. RT @dbbrakebill: i…       AAPL 2015-01-02
      631  @Weeklyoptions http://Weeklyoptionplays.com we…       AAPL 2015-01-02
      632  Cash flow machine. RT @themandotcom: $MSFT, wh…       MSFT 2015-01-02
```

1.4 Feature Construction

1.4.1 1. Sentiment Analysis

```
[22]: def vader_sentiment_scores(df, text_col='Tweet'):

          sentiment_analyzer = SentimentIntensityAnalyzer()
          df = df.copy()

          tweets['Vader_Negative'] = np.nan
          tweets['Vader_Neutral'] = np.nan
          tweets['Vader_Positive'] = np.nan
          tweets['Vader_Polarity'] = np.nan

          for indx, row in df.iterrows():
              try:
                  # Normalize the text to ASCII
                  text = unicodedata.normalize('NFKD', row[text_col])
                  sentiment = sentiment_analyzer.polarity_scores(text)

                  df.at[indx, 'Vader_Negative'] = sentiment['neg']
                  df.at[indx, 'Vader_Neutral'] = sentiment['neu']
```

```
            df.at[indx, 'Vader_Positive'] = sentiment['pos']
            df.at[indx, 'Vader_Polarity'] = sentiment['compound']

        except TypeError:
            print(f"TypeError on row {indx}: {row[text_col]}")
            break

    return df
```

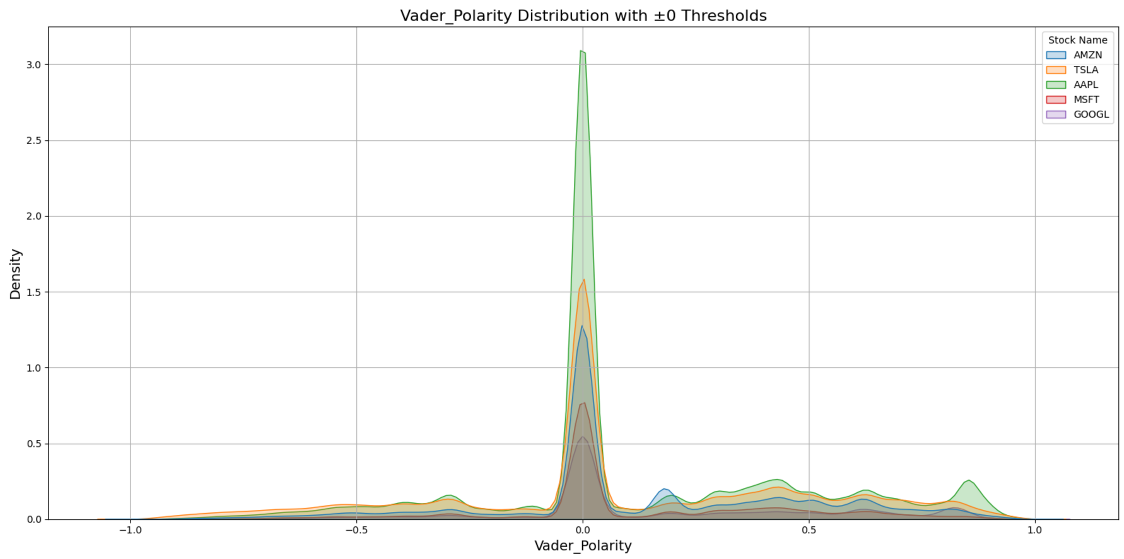
[23]:
```
tweets_sent = vader_sentiment_scores(tweets)
```

[24]:
```
tweets_sent.head()
```

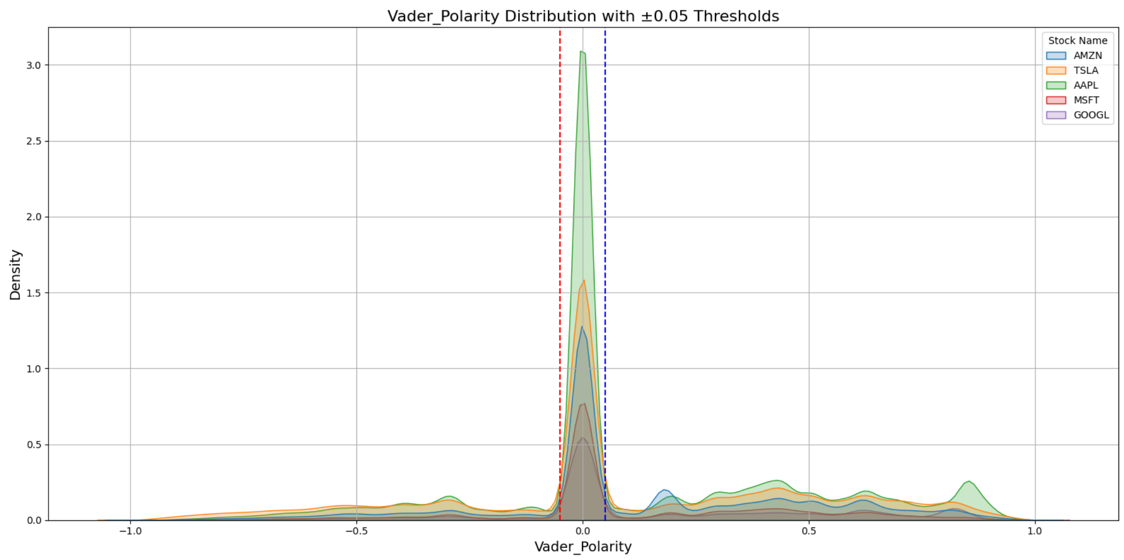
[24]:
```
                                                 Tweet Stock Name       Date  \
628  S&P100 #Stocks Performance $HD $LOW $SBUX $TGT…       AMZN 2015-01-02
629  Will Audi's Electric Q7 Cause $TSLA Model X Ba…       TSLA 2015-01-02
630  Either way you're a winnah. RT @dbbrakebill: i…       AAPL 2015-01-02
631  @Weeklyoptions http://Weeklyoptionplays.com we…       AAPL 2015-01-02
632  Cash flow machine. RT @themandotcom: $MSFT, wh…       MSFT 2015-01-02

     Vader_Negative  Vader_Neutral  Vader_Positive  Vader_Polarity
628           0.000          1.000           0.000          0.0000
629           0.000          1.000           0.000          0.0000
630           0.155          0.845           0.000         -0.3736
631           0.000          0.756           0.244          0.6369
632           0.000          1.000           0.000          0.0000
```

[25]:
```
def plot_kde(df, polarity_col, threshold=0, stock_col='Stock Name'):

    plt.figure(figsize=(16, 8))

    sns.kdeplot(data=df, x=polarity_col, hue=stock_col, fill=True)

    if threshold != 0:
        # Vertical threshold lines
        plt.axvline(-threshold, color='red', linestyle='--', label=f'Lower␣
↪Threshold -{threshold}')
        plt.axvline(threshold, color='blue', linestyle='--', label=f'Upper␣
↪Threshold +{threshold}')

    # Final touches
    plt.title(f"{polarity_col} Distribution with ±{threshold} Thresholds",␣
↪fontsize=16)
    plt.xlabel(polarity_col, fontsize=14)
    plt.ylabel("Density", fontsize=14)
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

[26]:
```
plot_kde(tweets_sent, polarity_col='Vader_Polarity')
```



[27]:
```
threshold = 0.05
plot_kde(tweets_sent, polarity_col='Vader_Polarity', threshold=threshold)
```



[28]:
```
def polarity_filter_by_threshold(df, threshold, polarity_col='Vader_Polarity',␣
 ↪date_col='Date', verbose=True):
    df = df.copy()
    df[date_col] = pd.to_datetime(df[date_col])

    # Apply polarity threshold filter
    filtered_df = df[
        (df[polarity_col] >= threshold) |
        (df[polarity_col] <= -threshold)
    ]

    # Compute stats
    total_dates = df[date_col].nunique()
    remaining_dates = filtered_df[date_col].nunique()
    dates_lost = total_dates - remaining_dates

    if verbose:
        print(f"-> Total unique dates before filtering: {total_dates}")
        print(f"+> Remaining unique dates after filtering: {remaining_dates}")
        print(f">> Dates lost: {dates_lost}")

    return filtered_df
```

[29]:
```
tweets_filtered = polarity_filter_by_threshold(tweets_sent, threshold)
```

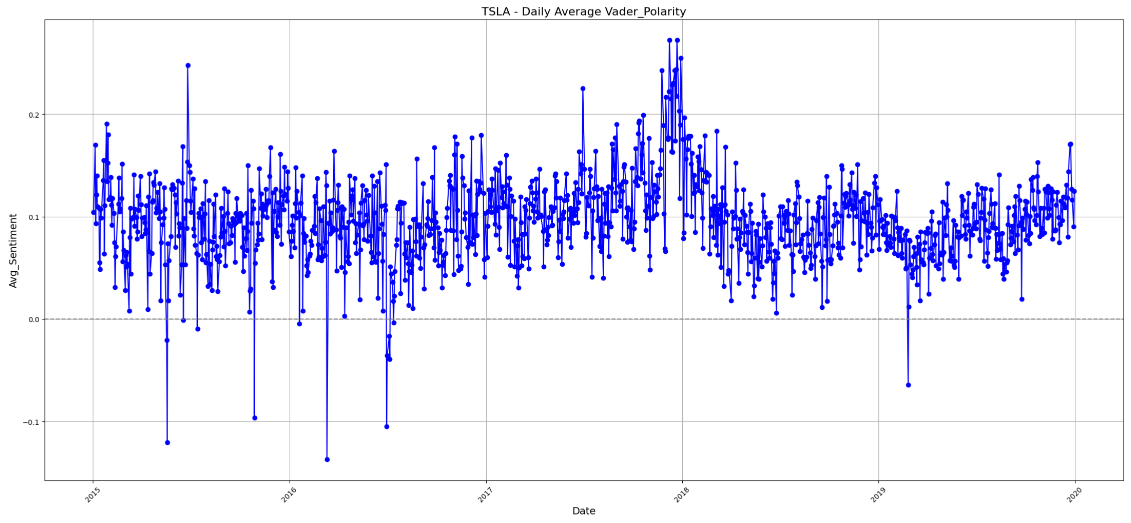
```
-> Total unique dates before filtering: 1258
+> Remaining unique dates after filtering: 1258
>> Dates lost: 0
```

[30]:
```
def plot_daily_sentiment(df, company, sentiment_col, date_col='Date',␣
 ↪company_col='Stock Name'):

    # Convert date column to datetime if needed
    df = df.copy()
    df[date_col] = pd.to_datetime(df[date_col])

    # Group and compute daily average sentiment
    daily_sentiment = (
        df.groupby([date_col, company_col])[sentiment_col]
        .mean()
        .reset_index(name='Avg_Sentiment')
    )

    # Filter for selected company
    company_df = daily_sentiment[daily_sentiment[company_col] == company]

    # Plot
    plt.figure(figsize=(22, 10))
    plt.plot(company_df[date_col], company_df['Avg_Sentiment'], color='blue',␣
↪marker='o')
    plt.title(f"{company} - Daily Average {sentiment_col}", fontsize=16)
    plt.xlabel("Date", fontsize=14)
    plt.ylabel("Avg_Sentiment", fontsize=14)
    plt.axhline(0, color='gray', linestyle='--')
    plt.grid(True)
    plt.tight_layout()
    plt.xticks(rotation=45)
    plt.show()
```

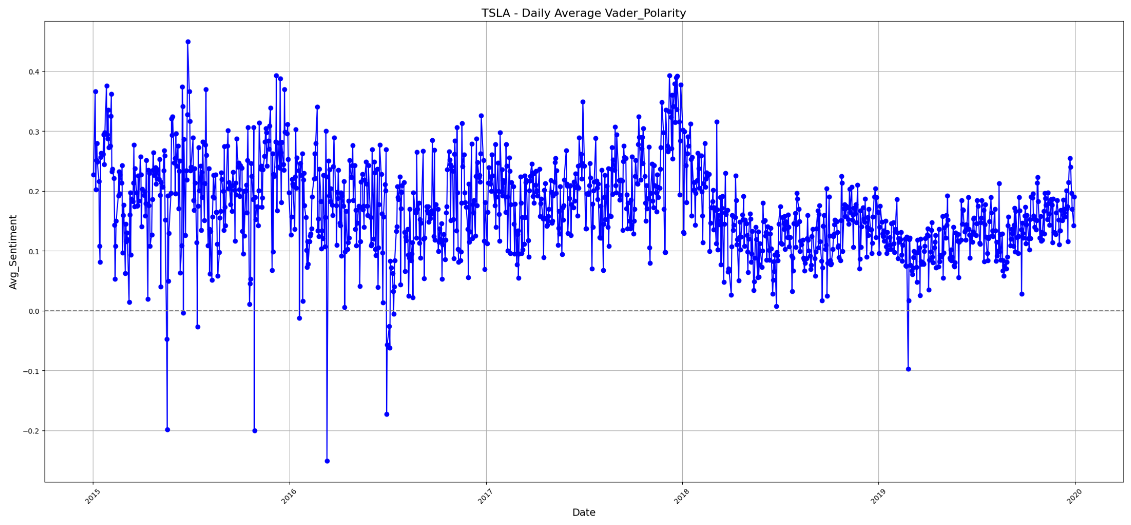
[31]:
```
tweets_sent.head()
```

[31]:
```
                                                 Tweet Stock Name        Date  \
628  S&P100 #Stocks Performance $HD $LOW $SBUX $TGT…       AMZN 2015-01-02
629  Will Audi's Electric Q7 Cause $TSLA Model X Ba…       TSLA 2015-01-02
630  Either way you're a winnah. RT @dbbrakebill: i…       AAPL 2015-01-02
631  @Weeklyoptions http://Weeklyoptionplays.com we…       AAPL 2015-01-02
632  Cash flow machine. RT @themandotcom: $MSFT, wh…       MSFT 2015-01-02

     Vader_Negative  Vader_Neutral  Vader_Positive  Vader_Polarity
628           0.000          1.000           0.000          0.0000
629           0.000          1.000           0.000          0.0000
630           0.155          0.845           0.000         -0.3736
631           0.000          0.756           0.244          0.6369
632           0.000          1.000           0.000          0.0000
```

[32]:
```
plot_daily_sentiment(tweets_sent, company='TSLA',␣
↪sentiment_col='Vader_Polarity')
```



[33]:
```
plot_daily_sentiment(tweets_filtered, company='TSLA',␣
    ↪sentiment_col='Vader_Polarity')
```



[34]:
```
tweets_filtered.head()
```

[34]:
```
                                                  Tweet Stock Name       Date  \
630  Either way you're a winnah. RT @dbbrakebill: i…       AAPL 2015-01-02
631  @Weeklyoptions http://Weeklyoptionplays.com we…       AAPL 2015-01-02
633  .@BenBajarin @counternotions @GlennF Thankful …       AAPL 2015-01-02
634  perfectly trading the S&P 500 in 2014 $FB $MU …       AMZN 2015-01-02
637  @KyleRohde @thehilker It could be, but that's …       AMZN 2015-01-02

     Vader_Negative  Vader_Neutral  Vader_Positive  Vader_Polarity
630           0.155          0.845           0.000         -0.3736
631           0.000          0.756           0.244          0.6369
633           0.000          0.802           0.198          0.5719
634           0.000          0.781           0.219          0.6369
637           0.103          0.641           0.255          0.7351
```

[35]:
```
tweets_filtered.to_csv("./data/filtered/tweets_filtered.csv", index=False)
```

1.4.2 2. Calculate Technical Indicators

[36]:
```
# RSI Calculation
def calculate_rsi(series, window=14):
    delta = series.diff()
    gain = delta.where(delta > 0, 0).rolling(window=window).mean()
    loss = -delta.where(delta < 0, 0).rolling(window=window).mean()
    rs = gain / loss
    rsi = 100 - (100 / (1 + rs))
    return rsi
```

[37]:
```
def get_technical_indicators(df):
    df = df.copy()
    # Trend
    df['SMA_5'] = df['Close'].rolling(window=5).mean()
    df['SMA_20'] = df['Close'].rolling(window=20).mean()

    # Bollinger Bands
    df['BB_Mid'] = df['SMA_5']
    df['BB_Std'] = df['Close'].rolling(window=20).std()
    df['BB_Upper'] = df['BB_Mid'] + 2 * df['BB_Std']
    df['BB_Lower'] = df['BB_Mid'] - 2 * df['BB_Std']

    # RSI
    df['RSI_14'] = calculate_rsi(df['Close'], window=14)

    # Log Return
    df['Log_Return'] = np.log(df['Close'] / df['Close'].shift(1))

    # OBV
    df['OBV'] = (np.sign(df['Close'].diff()) * df['Volume']).fillna(0).cumsum()

    # Lag Features
    df['Prev_Close'] = df['Close'].shift(1)
    df['Prev_Volume'] = df['Volume'].shift(1)

    # Time Features
    df['DayOfWeek'] = df.index.dayofweek
    df['Month'] = df.index.month

    # Clean
    df = df.bfill().ffill()

    return df
```

[38]:
```
stocks.head()
```

[38]:
```
         Date       Open       High        Low      Close  Adj Close  \
0 2015-01-02  27.847500  27.860001  26.837500  27.332500  24.320429
1 2015-01-05  27.072500  27.162500  26.352501  26.562500  23.635286
2 2015-01-06  26.635000  26.857500  26.157499  26.565001  23.637505
3 2015-01-07  26.799999  27.049999  26.674999  26.937500  23.968958
4 2015-01-08  27.307501  28.037500  27.174999  27.972500  24.889906

      Volume Stock Name
0  212818400       AAPL
```

```
1  257142000       AAPL
2  263188400       AAPL
3  160423600       AAPL
4  237458000       AAPL
```

[39]:
```
stocks = stocks.set_index('Date').sort_index()
stocks = get_technical_indicators(stocks)

stocks.head()
```

[39]:
```
                 Open       High        Low      Close  Adj Close     Volume  \
Date
2015-01-02  27.847500  27.860001  26.837500  27.332500  24.320429  212818400
2015-01-02  14.858000  14.883333  14.217333  14.620667  14.620667   71466000
2015-01-02  26.629999  26.790001  26.393999  26.477501  26.351515   26480000
2015-01-02  46.660000  47.419998  46.540001  46.759998  40.072136   27913900
2015-01-02  15.629000  15.737500  15.348000  15.426000  15.426000   55664000

           Stock Name      SMA_5     SMA_20     BB_Mid     BB_Std   BB_Upper  \
Date
2015-01-02       AAPL  26.123333  25.620192  26.123333  11.873179  49.228223
2015-01-02       TSLA  26.123333  25.620192  26.123333  11.873179  49.228223
2015-01-02      GOOGL  26.123333  25.620192  26.123333  11.873179  49.228223
2015-01-02       MSFT  26.123333  25.620192  26.123333  11.873179  49.228223
2015-01-02       AMZN  26.123333  25.620192  26.123333  11.873179  49.228223

            BB_Lower     RSI_14  Log_Return         OBV  Prev_Close  \
Date
2015-01-02  1.735509  46.167054   -0.625640         0.0   27.332500
2015-01-02  1.735509  46.167054   -0.625640 -71466000.0   27.332500
2015-01-02  1.735509  46.167054    0.593859 -44986000.0   14.620667
2015-01-02  1.735509  46.167054    0.568733 -17072100.0   26.477501
2015-01-02  1.735509  46.167054   -1.108974 -72736100.0   46.759998

            Prev_Volume  DayOfWeek  Month
Date
2015-01-02  212818400.0          4      1
2015-01-02  212818400.0          4      1
2015-01-02   71466000.0          4      1
2015-01-02   26480000.0          4      1
2015-01-02   27913900.0          4      1
```

1.4.3 3. Filter Company Data

[40]:
```
tweets_filtered = pd.read_csv("./data/filtered/tweets_filtered.csv")
```

[41]:
```
company_list = ['AAPL', 'AMZN', 'MSFT', 'GOOGL', 'TSLA']
```

[42]:
```
# Filter for selected stocks
filter_tweets = tweets_filtered[tweets_filtered['Stock Name'].
 ↪isin(company_list)].copy()

# Group and compute daily average sentiment
daily_sentiment = (
    filter_tweets
    .groupby(['Stock Name', 'Date'])['Vader_Polarity']
    .mean()
    .reset_index()
)

# Set Date as index (for plotting or merging)
daily_sentiment['Date'] = pd.to_datetime(daily_sentiment['Date'])
daily_sentiment = daily_sentiment.set_index('Date').sort_index()

daily_sentiment.head()
```

[42]:
```
           Stock Name  Vader_Polarity
Date
2015-01-02       AAPL        0.276950
2015-01-02      GOOGL        0.470091
2015-01-02       TSLA        0.227755
2015-01-02       AMZN        0.222096
2015-01-02       MSFT        0.260802
```

[43]:
```
filter_stocks = stocks[stocks['Stock Name'].isin(company_list)]
filter_stocks.head()
```

[43]:
```
                 Open       High        Low      Close  Adj Close     Volume  \
Date
2015-01-02  27.847500  27.860001  26.837500  27.332500  24.320429  212818400
2015-01-02  14.858000  14.883333  14.217333  14.620667  14.620667   71466000
2015-01-02  26.629999  26.790001  26.393999  26.477501  26.351515   26480000
2015-01-02  46.660000  47.419998  46.540001  46.759998  40.072136   27913900
2015-01-02  15.629000  15.737500  15.348000  15.426000  15.426000   55664000

           Stock Name      SMA_5     SMA_20     BB_Mid     BB_Std   BB_Upper  \
Date
2015-01-02       AAPL  26.123333  25.620192  26.123333  11.873179  49.228223
2015-01-02       TSLA  26.123333  25.620192  26.123333  11.873179  49.228223
2015-01-02      GOOGL  26.123333  25.620192  26.123333  11.873179  49.228223
2015-01-02       MSFT  26.123333  25.620192  26.123333  11.873179  49.228223
2015-01-02       AMZN  26.123333  25.620192  26.123333  11.873179  49.228223

             BB_Lower     RSI_14  Log_Return         OBV  Prev_Close  \
Date
```

```
2015-01-02  1.735509  46.167054   -0.625640         0.0   27.332500
2015-01-02  1.735509  46.167054   -0.625640 -71466000.0   27.332500
2015-01-02  1.735509  46.167054    0.593859 -44986000.0   14.620667
2015-01-02  1.735509  46.167054    0.568733 -17072100.0   26.477501
2015-01-02  1.735509  46.167054   -1.108974 -72736100.0   46.759998

            Prev_Volume  DayOfWeek  Month
Date
2015-01-02  212818400.0          4      1
2015-01-02  212818400.0          4      1
2015-01-02   71466000.0          4      1
2015-01-02   26480000.0          4      1
2015-01-02   27913900.0          4      1
```

1.4.4 4. Merging (Stock + Sentiment)

[44]:
```
def merging(company_list, filter_stocks, daily_sentiment):
    """
    Create a joined dataset of stock data and sentiment scores for a list of␣
 ↪stock tickers.

    Parameters:
    -----------
    company_list : list
        List of stock ticker symbols (e.g., ['AAPL', 'MSFT', 'GOOGL'])
    filter_stocks : pandas.DataFrame
        DataFrame containing stock price data with 'Stock Name' column
    daily_sentiment : pandas.DataFrame
        DataFrame containing sentiment scores with 'Stock Name' column

    Returns:
    --------
    dict
        Dictionary with ticker symbols as keys and joined DataFrames as values
    """
    # Dictionary to store results
    result_dict = {}

    for company in company_list:
        stock = filter_stocks[filter_stocks['Stock Name'] == company].copy()
        ticker_sentiment = daily_sentiment[daily_sentiment['Stock Name'] ==␣
 ↪company].copy()

        # Normalize datetime index (remove time, timezone)
        stock.index = pd.to_datetime(stock.index).normalize()
        ticker_sentiment.index = pd.to_datetime(ticker_sentiment.index).
 ↪normalize()
```

```
        # Perform inner join to keep only dates present in both
        joined_data = stock.join(ticker_sentiment[['Vader_Polarity']],␣
↪how='inner')

        result_dict[company] = joined_data

    return result_dict

stock_data_dict = merging(company_list, filter_stocks, daily_sentiment)
```

[45]:
```
AAPL = stock_data_dict['AAPL']

AAPL.head()
```

[45]:
```
                 Open       High        Low      Close  Adj Close     Volume  \
Date
2015-01-02  27.847500  27.860001  26.837500  27.332500  24.320429  212818400
2015-01-05  27.072500  27.162500  26.352501  26.562500  23.635286  257142000
2015-01-06  26.635000  26.857500  26.157499  26.565001  23.637505  263188400
2015-01-07  26.799999  27.049999  26.674999  26.937500  23.968958  160423600
2015-01-08  27.307501  28.037500  27.174999  27.972500  24.889906  237458000

           Stock Name      SMA_5     SMA_20     BB_Mid  …   BB_Upper  \
Date                                                    …
2015-01-02       AAPL  26.123333  25.620192  26.123333  …  49.228223
2015-01-05       AAPL  19.415400  25.620192  19.415400  …  49.228223
2015-01-06       AAPL  31.822701  25.620192  31.822701  …  49.228223
2015-01-07       AAPL  27.564067  25.620192  27.564067  …  49.228223
2015-01-08       AAPL  19.783466  25.672308  19.783466  …  43.502581

            BB_Lower     RSI_14  Log_Return          OBV  Prev_Close  \
Date
2015-01-02  1.735509  46.167054   -0.625640          0.0   27.332500
2015-01-05  1.735509  46.167054    0.640015   89576400.0   14.006000
2015-01-06  1.735509  46.167054   -0.541409 -170386000.0   45.650002
2015-01-07  1.735509  53.500699    0.649948  -41384800.0   14.063333
2015-01-08 -3.935649  54.182309    0.621640  205035200.0   15.023000

            Prev_Volume  DayOfWeek  Month  Vader_Polarity
Date
2015-01-02  212818400.0          4      1        0.276950
2015-01-05   80527500.0          0      1        0.251420
2015-01-06   36447900.0          1      1        0.281760
2015-01-07   44526000.0          2      1        0.271373
2015-01-08   61768000.0          3      1        0.321796
```

```
[5 rows x 21 columns]
```

[46]:
```
for company in company_list:
    df = stock_data_dict[company]
    df.to_csv(f"./data/filtered/{company}_filtered.csv", index=True)
```

1.4.5 5. Visualizing Related Features

[47]:
```
company_list = ['TSLA', 'AAPL', 'AMZN', 'GOOGL']
stock_data_dict = {}

for symbol in company_list:
    path = f"./data/filtered/{symbol}_filtered.csv"
    stock_data_dict[symbol] = pd.read_csv(path)
```

[48]:
```
tweets_filtered = pd.read_csv("./data/filtered/tweets_filtered.csv")
```

[49]:
```
TSLA = stock_data_dict['TSLA']
TSLA.head()
```

[49]:
```
         Date       Open       High        Low      Close  Adj Close  \
0  2015-01-02  14.858000  14.883333  14.217333  14.620667  14.620667
1  2015-01-05  14.303333  14.433333  13.810667  14.006000  14.006000
2  2015-01-06  14.004000  14.280000  13.614000  14.085333  14.085333
3  2015-01-07  14.223333  14.318667  13.985333  14.063333  14.063333
4  2015-01-08  14.187333  14.253333  14.000667  14.041333  14.041333

     Volume Stock Name      SMA_5     SMA_20  …   BB_Upper  BB_Lower  \
0  71466000       TSLA  26.123333  25.620192  …  49.228223  1.735509
1  80527500       TSLA  23.454899  25.620192  …  49.228223  1.735509
2  93928500       TSLA  31.838567  25.620192  …  49.228223  1.735509
3  44526000       TSLA  25.129466  25.620192  …  49.228223  1.735509
4  51637500       TSLA  25.994467  25.587975  …  50.001643  1.987290

      RSI_14  Log_Return          OBV  Prev_Close  Prev_Volume  DayOfWeek  \
0  46.167054   -0.625640  -71466000.0   27.332500  212818400.0          4
1  46.167054   -0.075838 -167565600.0   15.109500   55484000.0          0
2  46.167054   -0.634461 -264314500.0   26.565001  263188400.0          1
3  41.060477   -1.190058 -201808400.0   46.230000   29114100.0          2
4  41.785148   -1.220617  109988900.0   47.590000   29645200.0          3

   Month  Vader_Polarity
0      1        0.227755
1      1        0.366453
2      1        0.202426
3      1        0.251155
4      1        0.279833
```

```
[5 rows x 22 columns]
```

[50]:
```
# Combine all stock DataFrames into one
stocks = pd.concat(stock_data_dict.values(), ignore_index=True)

#      datetime
stocks['Date'] = pd.to_datetime(stocks['Date'])
stocks = stocks.sort_values(['Date', 'Stock Name']).reset_index(drop=True)

print(stocks['Stock Name'].value_counts())
```

```
Stock Name
AMZN     1258
GOOGL    1258
TSLA     1258
AAPL     1255
Name: count, dtype: int64
```

[51]:
```
stocks.head()
```

[51]:
```
        Date       Open       High        Low      Close  Adj Close  \
0 2015-01-02  27.847500  27.860001  26.837500  27.332500  24.320429
1 2015-01-02  15.629000  15.737500  15.348000  15.426000  15.426000
2 2015-01-02  26.629999  26.790001  26.393999  26.477501  26.351515
3 2015-01-02  14.858000  14.883333  14.217333  14.620667  14.620667
4 2015-01-05  27.072500  27.162500  26.352501  26.562500  23.635286

      Volume Stock Name      SMA_5     SMA_20  …   BB_Upper  BB_Lower  \
0  212818400       AAPL  26.123333  25.620192  …  49.228223  1.735509
1   55664000       AMZN  26.123333  25.620192  …  49.228223  1.735509
2   26480000      GOOGL  26.123333  25.620192  …  49.228223  1.735509
3   71466000       TSLA  26.123333  25.620192  …  49.228223  1.735509
4  257142000       AAPL  19.415400  25.620192  …  49.228223  1.735509

      RSI_14  Log_Return         OBV  Prev_Close  Prev_Volume  DayOfWeek  \
0  46.167054   -0.625640         0.0   27.332500  212818400.0          4
1  46.167054   -1.108974 -72736100.0   46.759998   27913900.0          4
2  46.167054    0.593859 -44986000.0   14.620667   71466000.0          4
3  46.167054   -0.625640 -71466000.0   27.332500  212818400.0          4
4  46.167054    0.640015  89576400.0   14.006000   80527500.0          0

   Month  Vader_Polarity
0      1        0.276950
1      1        0.222096
2      1        0.470091
3      1        0.227755
4      1        0.251420
```

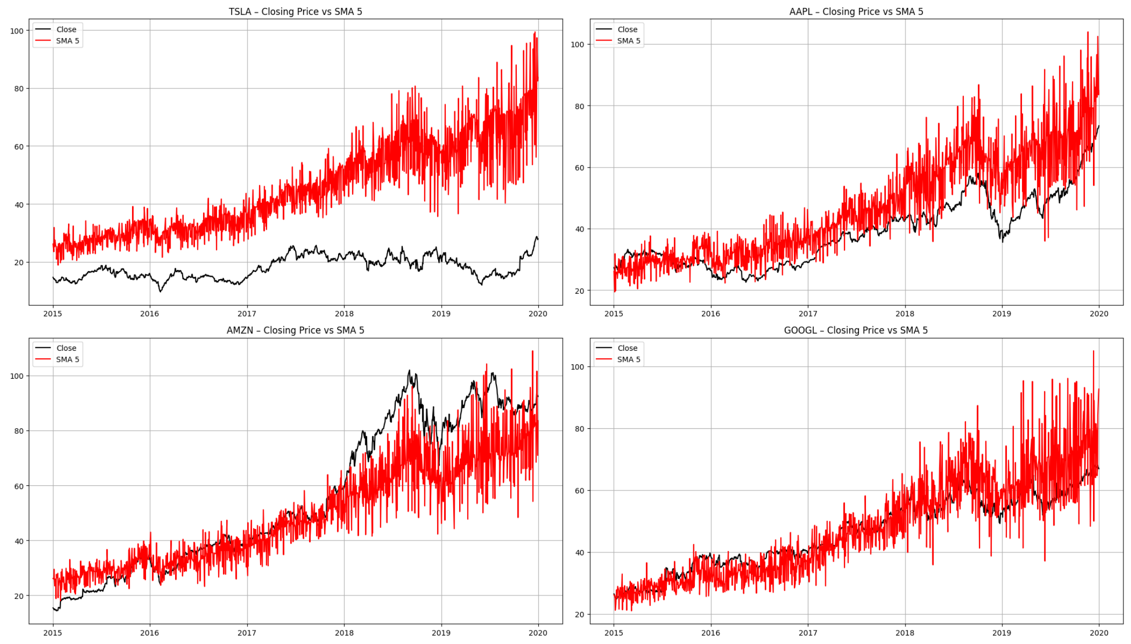
```
[5 rows x 22 columns]
```

[52]:
```
stocks['Date'].unique().value_counts()
```

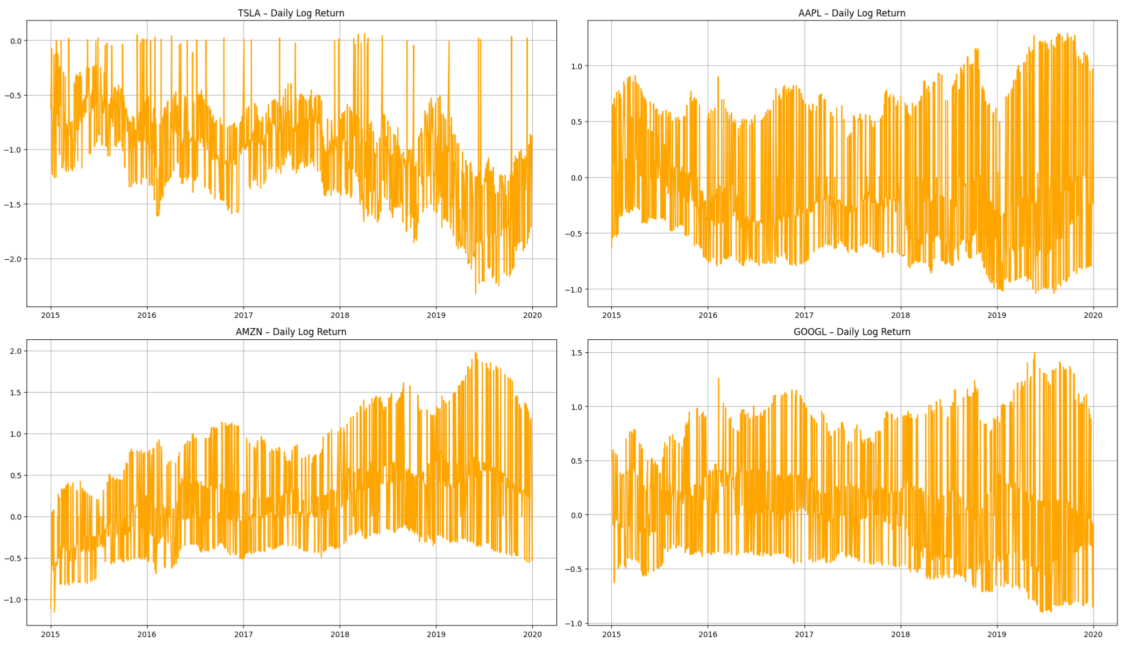
[52]:
```
2015-01-02    1
2015-01-05    1
2015-01-06    1
2015-01-07    1
2015-01-08    1
             ..
2019-12-24    1
2019-12-26    1
2019-12-27    1
2019-12-30    1
2019-12-31    1
Name: count, Length: 1258, dtype: int64
```

[53]:
```
def plot_price_sma_grid(df_all, company_list):
    fig, axs = plt.subplots(2, 2, figsize=(21, 12))  # 14x6 per plot
    axs = axs.flatten()

    for i, company in enumerate(company_list):
        df = df_all[df_all['Stock Name'] == company]
        axs[i].plot(df['Date'], df['Close'], label='Close', color='black')
        axs[i].plot(df['Date'], df['SMA_5'], label='SMA 5', color='red')
        axs[i].set_title(f'{company} - Closing Price vs SMA 5')
        axs[i].legend()
        axs[i].grid(True)

    plt.tight_layout()
    plt.show()

def plot_log_return_grid(df_all, company_list):
    fig, axs = plt.subplots(2, 2, figsize=(21, 12))
    axs = axs.flatten()

    for i, company in enumerate(company_list):
        df = df_all[df_all['Stock Name'] == company]
        axs[i].plot(df['Date'], df['Log_Return'], color='orange')
        axs[i].set_title(f'{company} - Daily Log Return')
        axs[i].grid(True)

    plt.tight_layout()
    plt.show()

def plot_rsi_grid(df_all, company_list):
    fig, axs = plt.subplots(2, 2, figsize=(21, 12))
    axs = axs.flatten()

    for i, company in enumerate(company_list):
        df = df_all[df_all['Stock Name'] == company]
        axs[i].plot(df['Date'], df['RSI_14'], color='purple')
        axs[i].axhline(70, color='red', linestyle='--')
        axs[i].axhline(30, color='green', linestyle='--')
        axs[i].set_title(f'{company} - RSI (14-day)')
        axs[i].grid(True)

    plt.tight_layout()
    plt.show()

def plot_volume_grid(df_all, company_list):
    fig, axs = plt.subplots(2, 2, figsize=(21, 12))
    axs = axs.flatten()

    for i, company in enumerate(company_list):
        df = df_all[df_all['Stock Name'] == company]
        axs[i].plot(df['Date'], df['Volume'], color='gray')
        axs[i].set_title(f'{company} - Daily Trading Volume')
        axs[i].grid(True)

    plt.tight_layout()
    plt.show()
```

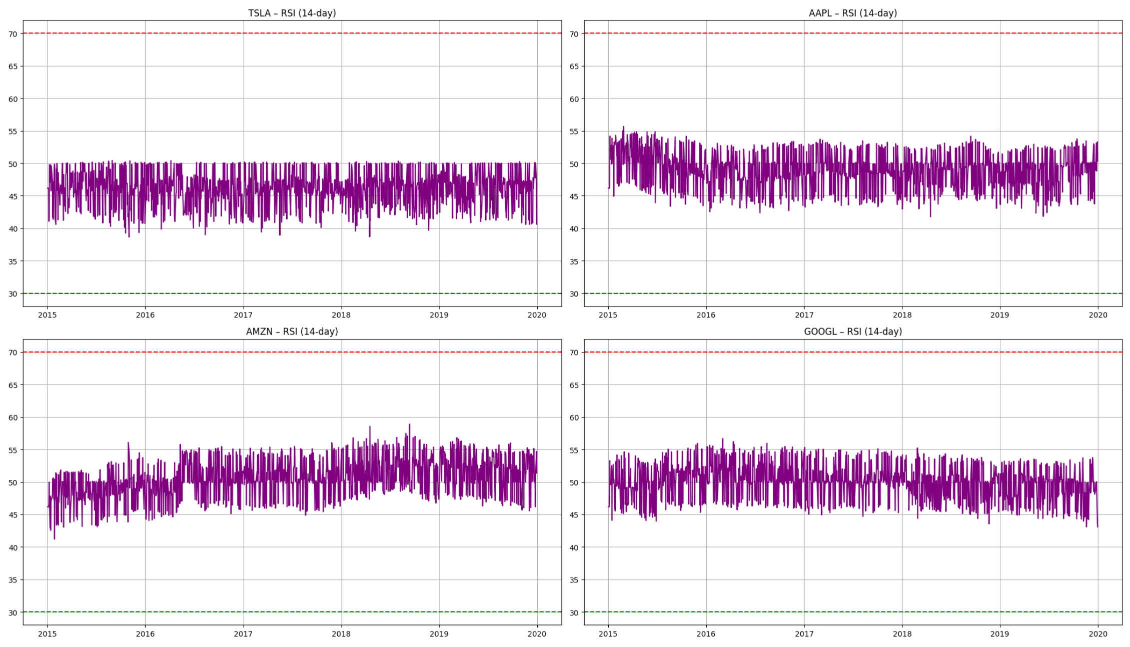
[54]:
```
plot_price_sma_grid(stocks, company_list)
```



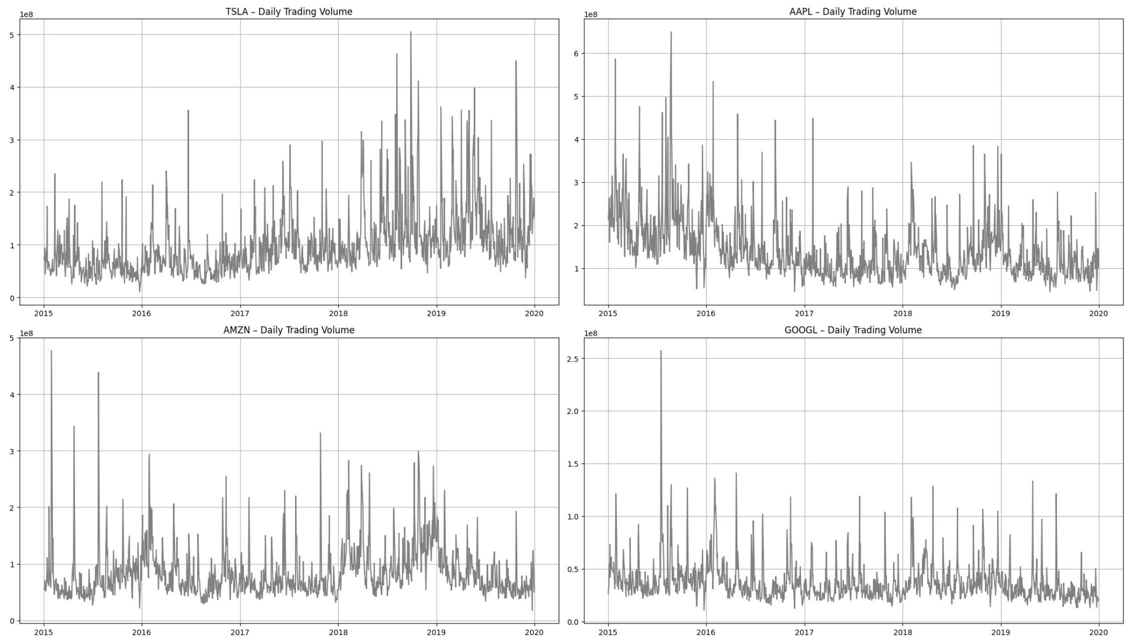
[55]:
```
plot_log_return_grid(stocks, company_list)
```



[56]:
```
plot_rsi_grid(stocks, company_list)
```

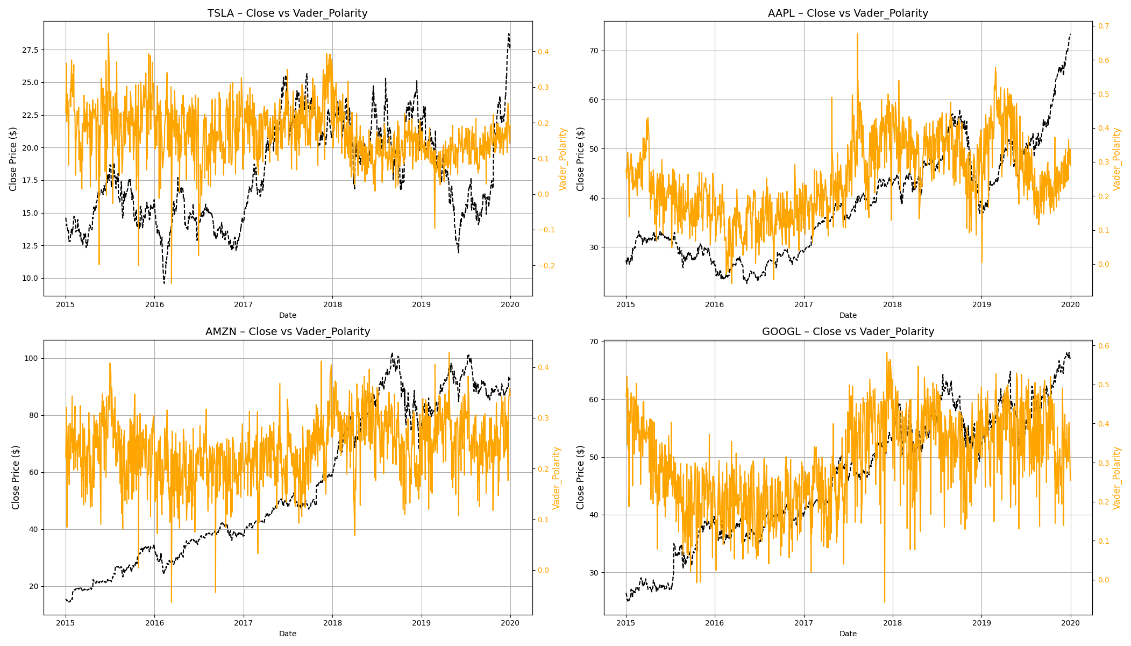


[57]:
```
plot_volume_grid(stocks, company_list)
```



[58]:
```
def plot_price_vs_sentiment(df_all, company_list,␣
↪sentiment_col='Vader_Polarity'):
    fig, axs = plt.subplots(2, 2, figsize=(21, 12))  # Grid for 4 companies
    axs = axs.flatten()

    for i, company in enumerate(company_list):
        df = df_all[df_all['Stock Name'] == company]
        ax1 = axs[i]

        # Left y-axis: Close Price
        ax1.plot(df['Date'], df['Close'], color='black', linestyle='--',␣
↪label='Close')
        ax1.set_ylabel('Close Price ($)', color='black', fontsize=12)
        ax1.tick_params(axis='y', labelcolor='black')
        ax1.grid(True)  #

        # Twin y-axis for sentiment
        ax2 = ax1.twinx()
        ax2.plot(df['Date'], df[sentiment_col], color='orange',␣
↪label=sentiment_col)
        ax2.set_ylabel(sentiment_col, color='orange', fontsize=12)
        ax2.tick_params(axis='y', labelcolor='orange')
        ax2.grid(False)  #
        # Title and grid
        ax1.set_title(f'{company} - Close vs {sentiment_col}', fontsize=14)
        ax1.set_xlabel('Date')

    plt.tight_layout()
    plt.show()
```

[59]:
```
plot_price_vs_sentiment(stocks, company_list, sentiment_col='Vader_Polarity')
```



1.5 Feature Selection

[60]:
```
TSLA = stock_data_dict['TSLA']
df = TSLA.copy()

df.head()
```

[60]:
```
         Date       Open       High        Low      Close  Adj Close  \
0  2015-01-02  14.858000  14.883333  14.217333  14.620667  14.620667
1  2015-01-05  14.303333  14.433333  13.810667  14.006000  14.006000
2  2015-01-06  14.004000  14.280000  13.614000  14.085333  14.085333
3  2015-01-07  14.223333  14.318667  13.985333  14.063333  14.063333
4  2015-01-08  14.187333  14.253333  14.000667  14.041333  14.041333

     Volume Stock Name      SMA_5     SMA_20  …   BB_Upper  BB_Lower  \
0  71466000       TSLA  26.123333  25.620192  …  49.228223  1.735509
1  80527500       TSLA  23.454899  25.620192  …  49.228223  1.735509
```

```
2  93928500       TSLA  31.838567  25.620192  …  49.228223  1.735509
3  44526000       TSLA  25.129466  25.620192  …  49.228223  1.735509
4  51637500       TSLA  25.994467  25.587975  …  50.001643  1.987290

      RSI_14  Log_Return          OBV  Prev_Close  Prev_Volume  DayOfWeek  \
0  46.167054   -0.625640  -71466000.0   27.332500  212818400.0          4
1  46.167054   -0.075838 -167565600.0   15.109500   55484000.0          0
2  46.167054   -0.634461 -264314500.0   26.565001  263188400.0          1
3  41.060477   -1.190058 -201808400.0   46.230000   29114100.0          2
4  41.785148   -1.220617  109988900.0   47.590000   29645200.0          3

   Month  Vader_Polarity
0      1        0.227755
1      1        0.366453
2      1        0.202426
3      1        0.251155
4      1        0.279833

[5 rows x 22 columns]
```

[61]:
```python
df.columns
```

[61]:
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume',
       'Stock Name', 'SMA_5', 'SMA_20', 'BB_Mid', 'BB_Std', 'BB_Upper',
       'BB_Lower', 'RSI_14', 'Log_Return', 'OBV', 'Prev_Close', 'Prev_Volume',
       'DayOfWeek', 'Month', 'Vader_Polarity'],
      dtype='object')
```

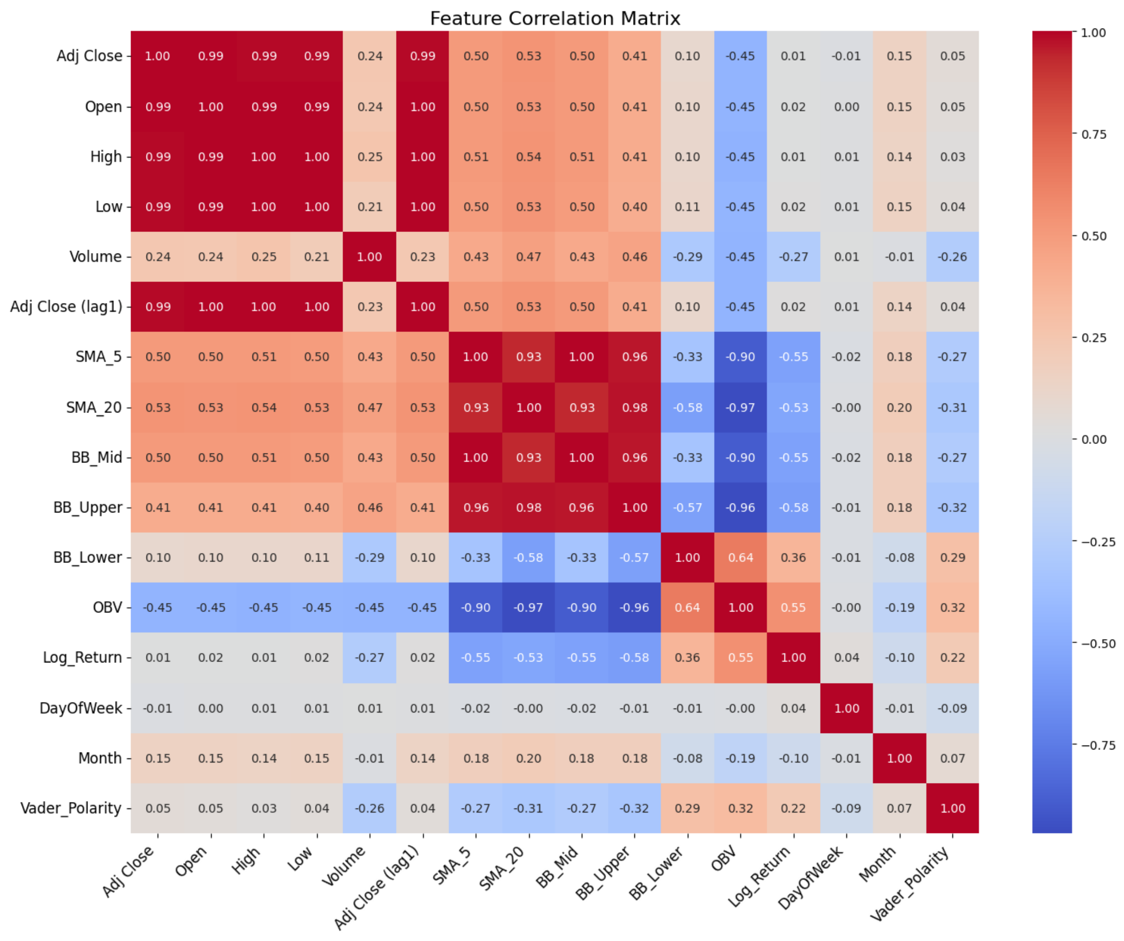
1.5.1 1. Avoid Data Leakage

[62]:
```python
# Lag technical indicators to avoid leakage
lag_cols = [
    'Adj Close', 'High', 'Low', 'Volume', 'SMA_5', 'SMA_20',
    'BB_Mid', 'BB_Std', 'BB_Upper', 'BB_Lower',
    'RSI_14', 'Log_Return', 'OBV', 'Vader_Polarity'
]

for col in lag_cols:
    if col == "Adj Close":
        df[f"{col} (lag1)"] = df[col].shift(1)
    else:
        df[col] = df[col].shift(1)

feature_cols = [
    'Adj Close',
    'Open', 'High', 'Low', 'Volume', 'Adj Close (lag1)',
    'SMA_5', 'SMA_20', 'BB_Mid', 'BB_Upper', 'BB_Lower',
    'OBV', 'Log_Return', 'DayOfWeek', 'Month',
    'Vader_Polarity'
]

df = df[feature_cols]
```

[63]:
```
feature_cols = [
    'Adj Close',                #    Adj Close
    'Open', 'High', 'Low', 'Volume',
    'Adj Close (lag1)',         #
    'SMA_5', 'SMA_20', 'BB_Mid', 'BB_Upper', 'BB_Lower',
    'OBV', 'Log_Return', 'DayOfWeek', 'Month',
    'Vader_Polarity'
]

target_col = 'Adj Close'  # or 'Log_Return'
```

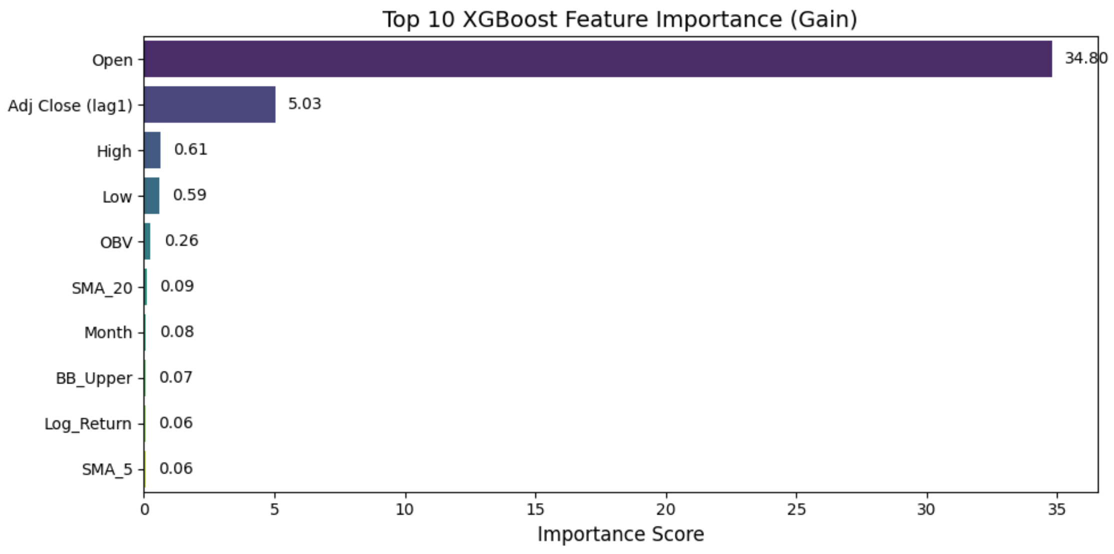
1.5.2 2. Feature Importance

[64]:
```
# ---     X_corr     Adj Close shift  ---
X_corr = df[feature_cols].copy()
X_corr = X_corr.iloc[:-1, :]  #   label   y = shift(-1)

# --- 2. Correlation Heatmap ---
plt.figure(figsize=(14, 11))
corr_matrix = X_corr.corr()
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm',
            annot_kws={"size": 10})  #

plt.title("Feature Correlation Matrix", fontsize=16)  #
plt.xticks(rotation=45, ha='right', fontsize=12)       # x
plt.yticks(rotation=0, fontsize=12)                    # y

plt.tight_layout()
plt.show()
```

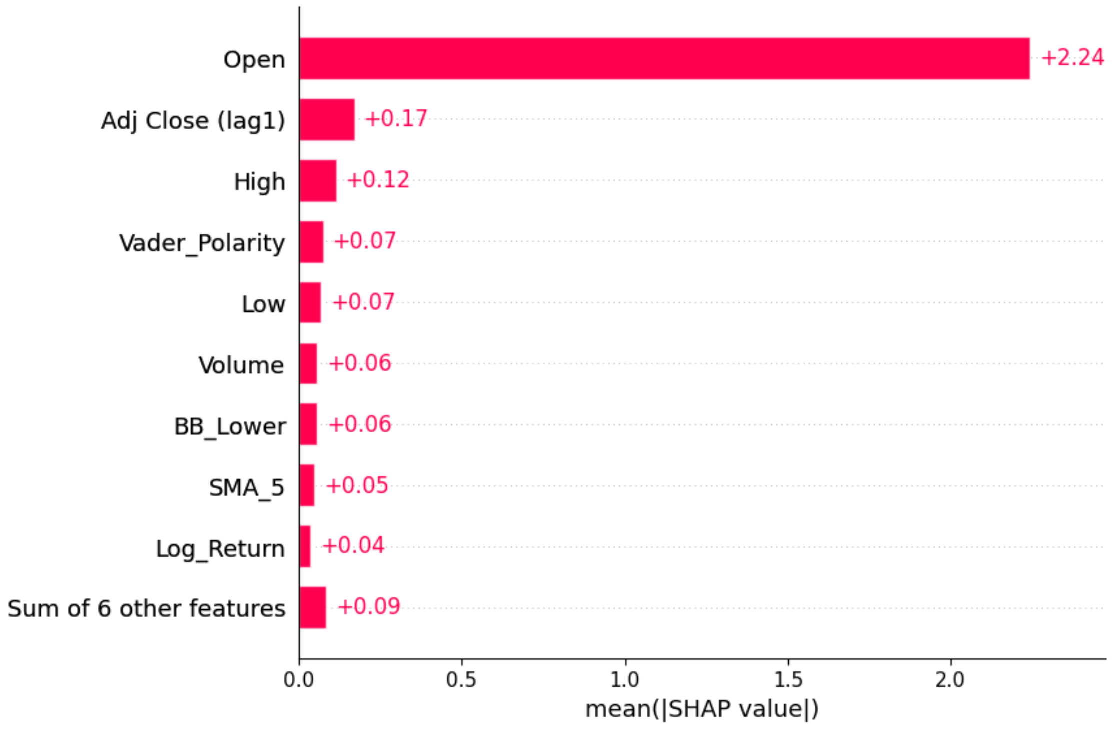


[65]:
```
#   1.   X    Adj Close   y
X = df[feature_cols].copy()
X = X.drop(columns=['Adj Close'])  #      Adj Close
y = df['Adj Close']                #   y

#   2.      lag SMA
X = X.iloc[1:, :]    #       lag1
y = y.iloc[1:]       #

#   3.
scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

#   4.
X_train, X_val, y_train, y_val = train_test_split(
    X_scaled, y, test_size=0.3, shuffle=False
)
```

```
[66]: # --- 3. XGBoost Feature Importance ---
xgb_model = xgb.XGBRegressor(n_estimators=100)
xgb_model.fit(X_train, y_train)

importances = xgb_model.get_booster().get_score(importance_type='gain')
importances = sorted(importances.items(), key=lambda x: x[1], reverse=True)

imp_df = pd.DataFrame(importances, columns=['Feature', 'Importance']).head(10)

plt.figure(figsize=(10, 5))
sns.barplot(
    data=imp_df,
    x='Importance',
    y='Feature',
    hue='Feature',
    palette='viridis',
    dodge=False,
    legend=False
)

#    x
for i, v in enumerate(imp_df['Importance']):
    plt.text(v + 0.5, i, f'{v:.2f}', va='center', fontsize=10)

plt.title("Top 10 XGBoost Feature Importance (Gain)", fontsize=14)
plt.xlabel("Importance Score", fontsize=12)
plt.ylabel("")
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.tight_layout()
plt.show()
```



```
[67]: # --- 4. SHAP with PermutationExplainer (CPU-friendly) ---
explainer = shap.Explainer(xgb_model.predict, X_val, algorithm="permutation")
shap_values = explainer(X_val)

#     +   10
plt.gcf().set_size_inches(10, 5)
shap.plots.bar(shap_values, max_display=10)
plt.tight_layout()
plt.show()
```



```
<Figure size 640x480 with 0 Axes>
```

```
[68]: # ========== 1.          ==========
X_corr_check = df[feature_cols].copy().iloc[:-1]   #
corr_matrix = X_corr_check.corr().abs()
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
to_drop_corr = [col for col in upper.columns if any(upper[col] > 0.90)]

print("  Dropped (correlated > 0.90):")
for col in to_drop_corr:
    print(f"   · {col}")
```

```
# ========== 2.   XGBoost Top 10   ==========
#     xgb_model
importances_dict = xgb_model.get_booster().get_score(importance_type='gain')
top10_sorted = sorted(importances_dict.items(), key=lambda x: x[1],␣
 ↪reverse=True)[:10]

print("\n Top 10 XGBoost Features (by Gain):")
for i, (feat, score) in enumerate(top10_sorted, 1):
    print(f"  {i:>2d}. {feat:<20} → {score:.2f}")
```

```
 Dropped (correlated > 0.90):
  · Open
  · High
  · Low
  · Adj Close (lag1)
  · SMA_20
  · BB_Mid
  · BB_Upper
  · OBV

 Top 10 XGBoost Features (by Gain):
   1. Open                 → 34.80
   2. Adj Close (lag1)     → 5.03
   3. High                 → 0.61
   4. Low                  → 0.59
   5. OBV                  → 0.26
   6. SMA_20               → 0.09
   7. Month                → 0.08
   8. BB_Upper             → 0.07
   9. Log_Return           → 0.06
  10. SMA_5                → 0.06
```

1.6 Time Series Train-Test Split

1.6.1 1. Loading Pre-processed Data

[133]:
```
company_list = ['TSLA', 'AAPL', 'AMZN', 'GOOGL', 'MSFT']
stock_data_dict = {}

for symbol in company_list:
    path = f"./data/filtered/{symbol}_filtered.csv"
    stock_data_dict[symbol] = pd.read_csv(path)
```

[134]:
```
TSLA = stock_data_dict['TSLA']
AAPL = stock_data_dict['AAPL']
AMZN = stock_data_dict['AMZN']
GOOGL = stock_data_dict['GOOGL']
```

```
MSFT = stock_data_dict['MSFT']
```

```
[135]: df = TSLA.copy()
       df.head()
```

```
[135]:          Date       Open       High        Low      Close  Adj Close  \
       0  2015-01-02  14.858000  14.883333  14.217333  14.620667  14.620667
       1  2015-01-05  14.303333  14.433333  13.810667  14.006000  14.006000
       2  2015-01-06  14.004000  14.280000  13.614000  14.085333  14.085333
       3  2015-01-07  14.223333  14.318667  13.985333  14.063333  14.063333
       4  2015-01-08  14.187333  14.253333  14.000667  14.041333  14.041333

            Volume Stock Name      SMA_5     SMA_20  …   BB_Upper  BB_Lower  \
       0  71466000       TSLA  26.123333  25.620192  …  49.228223  1.735509
       1  80527500       TSLA  23.454899  25.620192  …  49.228223  1.735509
       2  93928500       TSLA  31.838567  25.620192  …  49.228223  1.735509
       3  44526000       TSLA  25.129466  25.620192  …  49.228223  1.735509
       4  51637500       TSLA  25.994467  25.587975  …  50.001643  1.987290

             RSI_14  Log_Return          OBV  Prev_Close  Prev_Volume  DayOfWeek  \
       0  46.167054   -0.625640  -71466000.0   27.332500  212818400.0          4
       1  46.167054   -0.075838 -167565600.0   15.109500   55484000.0          0
       2  46.167054   -0.634461 -264314500.0   26.565001  263188400.0          1
       3  41.060477   -1.190058 -201808400.0   46.230000   29114100.0          2
       4  41.785148   -1.220617  109988900.0   47.590000   29645200.0          3

          Month  Vader_Polarity
       0      1        0.227755
       1      1        0.366453
       2      1        0.202426
       3      1        0.251155
       4      1        0.279833

       [5 rows x 22 columns]
```

1.6.2 2. Avoid Data Leakage

```
[136]: # Lag technical indicators to avoid leakage
       lag_cols = [
           'Adj Close', 'High', 'Low', 'Volume', 'SMA_5', 'SMA_20',
           'BB_Mid', 'BB_Std', 'BB_Upper', 'BB_Lower',
           'RSI_14', 'Log_Return', 'OBV', 'Vader_Polarity'
       ]

       for col in lag_cols:
           if col == "Adj Close":
               df[f"{col} (lag1)"] = df[col].shift(1)
```

```
    else:
        df[col] = df[col].shift(1)

feature_cols = [
    'Adj Close',
    'Open', 'High', 'Low', 'Volume', 'Adj Close (lag1)',
    'SMA_5', 'SMA_20', 'BB_Mid', 'BB_Upper', 'BB_Lower',
    'OBV', 'Log_Return', 'DayOfWeek', 'Month',
    'Vader_Polarity'
]

df = df[feature_cols]
```

[137]:
```
df.head()
```

[137]:
```
   Adj Close       Open       High        Low      Volume  Adj Close (lag1)  \
0  14.620667  14.858000        NaN        NaN         NaN               NaN
1  14.006000  14.303333  14.883333  14.217333  71466000.0         14.620667
2  14.085333  14.004000  14.433333  13.810667  80527500.0         14.006000
3  14.063333  14.223333  14.280000  13.614000  93928500.0         14.085333
4  14.041333  14.187333  14.318667  13.985333  44526000.0         14.063333

       SMA_5     SMA_20     BB_Mid   BB_Upper  BB_Lower          OBV  \
0        NaN        NaN        NaN        NaN       NaN          NaN
1  26.123333  25.620192  26.123333  49.228223  1.735509  -71466000.0
2  23.454899  25.620192  23.454899  49.228223  1.735509 -167565600.0
3  31.838567  25.620192  31.838567  49.228223  1.735509 -264314500.0
4  25.129466  25.620192  25.129466  49.228223  1.735509 -201808400.0

   Log_Return  DayOfWeek  Month  Vader_Polarity
0         NaN          4      1             NaN
1   -0.625640          0      1        0.227755
2   -0.075838          1      1        0.366453
3   -0.634461          2      1        0.202426
4   -1.190058          3      1        0.251155
```

[138]:
```
# feature_cols = [
#     'Adj Close',  # should be the first one for Y
#     'Adj Close (lag1)',
#     'SMA_5',               # short-term trend
#     'Volume',
#     'BB_Mid',            # risk signal
#     'Log_Return',
#     'DayOfWeek',
#     'Month',
#     'Vader_Polarity'  # should be the last one for SENTIMENT
# ]
```

[139]:
```
feature_cols = [
    'Adj Close',  # should be the first one for Y
    'Adj Close (lag1)',
    'SMA_5',               # short-term trend
    'Volume',
    'BB_Mid',            # risk signal
    'Log_Return',
    'DayOfWeek',
    'Month',
    'Vader_Polarity'  # should be the last one for SENTIMENT
]
```

1.6.3 Define Rolling Window & Prediction Day

[140]:
```
# Step 0: Define sliding window parameters
n_past = 5      # Use past 5 days
n_future = 1    # Predict next 1 day
```

1.6.4 Train-Test Split

[141]:
```
train_size = 0.7
train_split_idx = int(train_size * len(df))
```

[142]:
```
df_filtered = df[feature_cols]
df_filtered = df_filtered.iloc[1:] # delete nan       lag1

# Step 0: Define split boundaries BEFORE scaling
train_df = df_filtered.iloc[:train_split_idx]
test_df  = df_filtered.iloc[train_split_idx:]

# Step 1: Fit scaler only on training data (Avoid Data Leakage)
scaler = MinMaxScaler()
scaler.fit(train_df)

# Step 2: Scale training and test data separately
train_scaled = scaler.transform(train_df)
test_scaled  = scaler.transform(test_df)

# Step 3: For inference later, only scale ['Adj Close']
scaler_for_inference = MinMaxScaler()
actual_scaled_close = scaler_for_inference.fit_transform(
    df_filtered[['Adj Close']]
)

# Step 3: Reconstruct sliding windows for train and test
def create_sequences(data, n_past, n_future):
    X, y = [], []
```

```python
    for i in range(n_past, len(data) - n_future + 1):
        X.append(data[i - n_past:i, 1:])
        y.append(data[i + n_future - 1:i + n_future, [0]])  # Predict Adj Close
    return np.array(X), np.array(y)

trainX, trainY = create_sequences(train_scaled, n_past, n_future)
testX, testY   = create_sequences(test_scaled, n_past, n_future)

# trainY = trainY.reshape(-1, 1)
# testY = testY.reshape(-1, 1)

print('TrainX shape = {}'.format(trainX.shape))
print('TrainY shape = {}'.format(trainY.shape))
print('TestX shape = {}'.format(testX.shape))
print('TestY shape = {}'.format(testY.shape))
```

```
TrainX shape = (875, 5, 8)
TrainY shape = (875, 1, 1)
TestX shape = (372, 5, 8)
TestY shape = (372, 1, 1)
```

1.6.5 (Un)Sentiment Split

[143]:
```python
# Without Sentiment (Baseline Model)
trainX_wo_tweet = trainX[:, :, :-1]  # Exclude last feature
testX_wo_tweet  = testX[:, :, :-1]
trainY_wo_tweet = trainY
testY_wo_tweet  = testY

# With Sentiment (Tweet-based Model)
trainX_with_tweet = trainX
testX_with_tweet  = testX
trainY_with_tweet = trainY
testY_with_tweet  = testY
```

[144]:
```python
print(trainX_with_tweet.shape, trainY_with_tweet.shape)
```

```
(875, 5, 8) (875, 1, 1)
```

2 Modeling

2.0.1 Random Seed

[145]:
```python
def set_all_seeds(seed=42):
    os.environ['PYTHONHASHSEED'] = str(seed)
    random.seed(seed)
    np.random.seed(seed)
    tf.random.set_seed(seed)
    tf.config.experimental.enable_op_determinism()  # TensorFlow 2.12+

    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)
        torch.backends.cudnn.deterministic = True
        torch.backends.cudnn.benchmark = False

set_all_seeds(seed=42)
```

2.0.2 CNN + BiLSTM

Pre-load Plotting Functions

```
[146]: def plot_metrics(model, history, X_train, y_train, X_val, y_val,␣
        ↪sentiment_mode):
            """
              RMSE   MAE         RMSE / MAE
            """
            history_data = history.history  #

            # 1.
            y_train_pred = model.predict(X_train, verbose=0).squeeze()
            y_val_pred   = model.predict(X_val, verbose=0).squeeze()

            y_train_true = y_train.squeeze()
            y_val_true   = y_val.squeeze()
            # y_train_true = y_train.reshape(-1, 1)
            # y_val_true   = y_val.reshape(-1, 1)

            # 2.
            train_rmse = np.sqrt(mean_squared_error(y_train_true, y_train_pred))
            val_rmse   = np.sqrt(mean_squared_error(y_val_true, y_val_pred))

            train_mae = mean_absolute_error(y_train_true, y_train_pred)
            val_mae   = mean_absolute_error(y_val_true, y_val_pred)

            title_suffix = " w/ Tweets" if sentiment_mode == "with" else "w/o Tweets"

            # 3.
            print(f"  {title_suffix} RMSE: Train = {train_rmse:.4f}, Test = {val_rmse:.
        ↪4f}")
            print(f"  {title_suffix} MAE : Train = {train_mae:.4f}, Test = {val_mae:.
        ↪4f}")

            # 4.    loss
            plt.figure(figsize=(21, 9))
```

```
    # ---- RMSE subplot
    plt.subplot(1, 2, 1)
    plt.plot(np.sqrt(history_data['loss']), label='Train RMSE')
    plt.plot(np.sqrt(history_data['val_loss']), label='Val RMSE')
    plt.title(f'RMSE - {title_suffix}', fontsize=18)
    plt.xlabel('Epoch', fontsize=18);
    plt.ylabel('RMSE', fontsize=18)
    plt.xticks(fontsize=16)
    plt.yticks(fontsize=16)
    plt.legend()

    # ---- MAE subplot
    plt.subplot(1, 2, 2)
    if 'mae' in history_data:
        plt.plot(history_data['mae'], label='Train MAE')
        plt.plot(history_data['val_mae'], label='Val MAE')
    else:
        # fallback: flat line (not recommended long term)
        plt.plot([train_mae] * len(history_data['loss']), label='Train MAE␣
  ↪(flat)')
        plt.plot([val_mae]   * len(history_data['val_loss']), label='Val MAE␣
  ↪(flat)')
    plt.title(f'MAE - {title_suffix}', fontsize=18)
    plt.xlabel('Epoch', fontsize=18);
    plt.ylabel('MAE', fontsize=18)
    plt.xticks(fontsize=16)
    plt.yticks(fontsize=16)
    plt.legend()

    #  Add supertitle here
    plt.suptitle(f'Model Training Metrics - CNN+BiLSTM', fontsize=24)
    plt.tight_layout()

    plt.show()
```

[147]:
```
# Helper Function: get predicted and true values in original (unscaled) price␣
  ↪units
def inv_preds(model, X, y):
    # Predict using the model and reshape to [samples, output_dim]
    p = model.predict(X, verbose=0).reshape(-1, y.shape[-1])

    # Reshape ground truth to same shape for inverse transformation
    g = y.reshape(-1, y.shape[-1])

    # Apply inverse scaling to recover original price scale
    # inv_p = scaler_for_inference.inverse_transform(p)
    # inv_g = scaler_for_inference.inverse_transform(g)

    # Return only the first column ('Adj Close') from each
    # return inv_p[:, 0], inv_g[:, 0]

    # Use the full-feature scaler
    inv_p = scaler.inverse_transform(
        np.concatenate([p, np.zeros((p.shape[0], scaler.n_features_in_ - 1))],␣
↪axis=1)
    )[:, 0]
    inv_g = scaler.inverse_transform(
        np.concatenate([g, np.zeros((g.shape[0], scaler.n_features_in_ - 1))],␣
↪axis=1)
    )[:, 0]

    return inv_p, inv_g
```

[148]:
```
df_filtered.head()
```

[148]:
```
   Adj Close  Adj Close (lag1)      SMA_5      Volume     BB_Mid  Log_Return  \
1  14.006000          14.620667  26.123333  71466000.0  26.123333   -0.625640
2  14.085333          14.006000  23.454899  80527500.0  23.454899   -0.075838
3  14.063333          14.085333  31.838567  93928500.0  31.838567   -0.634461
4  14.041333          14.063333  25.129466  44526000.0  25.129466   -1.190058
5  13.777333          14.041333  25.994467  51637500.0  25.994467   -1.220617

   DayOfWeek  Month  Vader_Polarity
1          0      1        0.227755
2          1      1        0.366453
3          2      1        0.202426
4          3      1        0.251155
5          4      1        0.279833
```

[149]:
```
def plot_pred_vs_actual(
    model, X_train, y_train, X_test, y_test, model_name,␣
↪sentiment_mode="without",
    all_dates=df.index, all_adj=df['Adj Close'],␣
↪scaler_for_inference=scaler_for_inference,
    stock_name="TSLA"
):
    """
        +      vs



    - model
    - X_train / y_train
    - X_test / y_test
    - all_dates    index  df.index
    - all_adj       df['Adj Close']
    - scaler_for_inference     scaler   fit
    - title_prefix      'NFLX'
    - sentiment_mode 'with' / 'without'
    """

    # Window Siz: number of lookback days
    lookback = X_train.shape[1]

    # Number of training samples
    n_train  = X_train.shape[0]

    # Number of testing samples
    n_test   = X_test.shape[0]

    # Predictions will start from the index equal to lookback (since earlier␣
↪data was used for input windows)
    pred_start = lookback

    # Time indices for training predictions aligned with full date range
    train_pts = np.arange(pred_start, pred_start + n_train)

    # Time indices for test predictions aligned with full date range
    test_pts  = np.arange(pred_start + n_train,
                          pred_start + n_train + n_test)

    # Get model predictions and true values (unscaled)
    pred_close_train, _ = inv_preds(model, X_train, y_train)
    pred_close_test,  _ = inv_preds(model, X_test, y_test)

    # Set title & label based on mode
    title_suffix = f"({model_name} w/ Tweets)" if sentiment_mode == "with" else␣
↪f"({model_name} w/o Tweets)"

    plt.figure(figsize=(18, 7))
    sns.lineplot(x=all_dates, y=all_adj, label='Actual Closing', color='black')
    sns.lineplot(x=all_dates[train_pts], y=pred_close_train, label='Train␣
↪Predicted', color='red')
    sns.lineplot(x=all_dates[test_pts],  y=pred_close_test,  label='Test␣
↪Predicted')

    plt.title(f'{stock_name} - Closing Price vs Actual Stock {title_suffix}',␣
↪fontsize=20)
    plt.xlabel('Date', fontsize=18)
    plt.ylabel('Price (Dollar)', fontsize=18)
    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)
    plt.legend(fontsize=16)
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

Model Presetting & Building

```
[150]: # def cnn_biLSTM(input_shape, output_dim):

       #     model = Sequential()

       #     model.add(Conv1D(128, kernel_size=2, strides=1, padding='valid',␣
        ↪input_shape=input_shape))
       #     model.add(MaxPooling1D(pool_size=2, strides=2))

       #     model.add(Conv1D(64, kernel_size=2, strides=1, padding='valid'))
       #     model.add(MaxPooling1D(pool_size=1, strides=2))

       #     model.add(Bidirectional(LSTM(256, return_sequences=True)))
       #     model.add(Dropout(0.2))
       #     model.add(Bidirectional(LSTM(256, return_sequences=True)))
       #     model.add(Dropout(0.2))

       #     model.add(Dense(32, activation='relu'))
       #     model.add(Dense(output_dim, activation='relu'))
       #     # model.summary()
       #     return model
```

```
[151]: def cnn_biLSTM(input_shape, output_dim):

           model = Sequential()

           model.add(Conv1D(128, kernel_size=2, strides=1, padding='valid',␣
        ↪input_shape=input_shape))
           model.add(MaxPooling1D(pool_size=2, strides=2))

           model.add(Conv1D(64, kernel_size=2, strides=1, padding='valid'))
           model.add(MaxPooling1D(pool_size=1, strides=2))

           model.add(Bidirectional(LSTM(256, return_sequences=True)))
           model.add(Dropout(0.2))
           model.add(Bidirectional(LSTM(256, return_sequences=True)))
           model.add(Dropout(0.2))

           model.add(Dense(32, activation='relu'))
           model.add(Dense(output_dim, activation='relu'))
           # model.summary()
```

```
    return model
```

[152]:
```
def directional_loss(y_true, y_pred):
    sign_true = K.sign(y_true[1:] - y_true[:-1])
    sign_pred = K.sign(y_pred[1:] - y_pred[:-1])
    correct_direction = K.cast(K.equal(sign_true, sign_pred), dtype=tf.float32)
    return 1.0 - K.mean(correct_direction)

def integrated_loss(delta=0.1, lambda_dir=0.5):
    huber = Huber(delta=delta)
    def loss(y_true, y_pred):
        val_loss = huber(y_true, y_pred)
        dir_loss = directional_loss(y_true, y_pred)
        return (1 - lambda_dir) * val_loss + lambda_dir * dir_loss
    return loss
```

[153]:
```
# Build models
cnnBiLSTM_woSent = cnn_biLSTM(
    (trainX_wo_tweet.shape[1], trainX_wo_tweet.shape[2]), trainY.shape[2]
)
cnnBiLSTM_woSent.compile(
    optimizer=Adam(learning_rate=0.0005),
    # loss=integrated_loss(delta=0.1, lambda_dir=0.16),  #  adjust as needed
    loss='mse',  #  adjust as needed
    metrics=['mae']
)



cnnBiLSTM_withSent = cnn_biLSTM(
    (trainX_with_tweet.shape[1], trainX_with_tweet.shape[2]), trainY.shape[2]
)
cnnBiLSTM_withSent.compile(
    optimizer=Adam(learning_rate=0.0005),
    # loss=integrated_loss(delta=0.1, lambda_dir=0.16),  #  adjust as needed
    loss='mse',  #  adjust as needed
    metrics=['mae']
)
```

```
/Users/yourth/Desktop/aaa/venv/lib/python3.11/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[154]:
```
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=8,
    restore_best_weights=False
)

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.7,          # ← good default
    patience=3,
    min_lr=1e-6,
    verbose=1
)
```

Model Fitting & Evaluating

[155]:
```
# Fit models
history_cnnBiLSTM_woSent = cnnBiLSTM_woSent.fit(
    trainX_wo_tweet,
    trainY_wo_tweet,
    epochs=50,
    batch_size=64,
    validation_data=(testX_wo_tweet, testY_wo_tweet),  # ← use your test split␣
 ↪here
    verbose=1,
    callbacks=[early_stop, reduce_lr]
)

history_cnnBiLSTM_withSent = cnnBiLSTM_withSent.fit(
    trainX_with_tweet,
    trainY_with_tweet,
    epochs=50,
    batch_size=64,
    validation_data=(testX_with_tweet, testY_with_tweet),  # ← and here
    verbose=1,
    callbacks=[early_stop, reduce_lr]
)
```

```
Epoch 1/50

2025-05-10 21:49:59.185667: E tensorflow/core/framework/node_def_util.cc:680]
NodeDef mentions attribute use_unbounded_threadpool which is not in the op
definition: Op<name=MapDataset; signature=input_dataset:variant,
other_arguments: -> handle:variant; attr=f:func;
attr=Targuments:list(type),min=0; attr=output_types:list(type),min=1;
attr=output_shapes:list(shape),min=1;
attr=use_inter_op_parallelism:bool,default=true;
attr=preserve_cardinality:bool,default=false;
attr=force_synchronous:bool,default=false; attr=metadata:string,default=""> This
may be expected if your graph generating binary is newer  than this binary.
Unknown attributes will be ignored. NodeDef: {{node ParallelMapDatasetV2/_15}}
2025-05-10 21:49:59.186132: E tensorflow/core/framework/node_def_util.cc:680]
```

```
NodeDef mentions attribute use_unbounded_threadpool which is not in the op
definition: Op<name=MapDataset; signature=input_dataset:variant,
other_arguments: -> handle:variant; attr=f:func;
attr=Targuments:list(type),min=0; attr=output_types:list(type),min=1;
attr=output_shapes:list(shape),min=1;
attr=use_inter_op_parallelism:bool,default=true;
attr=preserve_cardinality:bool,default=false;
attr=force_synchronous:bool,default=false; attr=metadata:string,default=""> This
may be expected if your graph generating binary is newer  than this binary.
Unknown attributes will be ignored. NodeDef: {{node ParallelMapDatasetV2/_15}}
```

```
14/14            2s 26ms/step -
loss: 0.2413 - mae: 0.4284 - val_loss: 0.0749 - val_mae: 0.2474 - learning_rate:
5.0000e-04
Epoch 2/50
14/14            0s 9ms/step - loss:
0.0459 - mae: 0.1763 - val_loss: 0.0221 - val_mae: 0.1247 - learning_rate:
5.0000e-04
Epoch 3/50
14/14            0s 9ms/step - loss:
0.0158 - mae: 0.1042 - val_loss: 0.0181 - val_mae: 0.1132 - learning_rate:
5.0000e-04
Epoch 4/50
14/14            0s 9ms/step - loss:
0.0086 - mae: 0.0720 - val_loss: 0.0243 - val_mae: 0.1329 - learning_rate:
5.0000e-04
Epoch 5/50
14/14            0s 9ms/step - loss:
0.0055 - mae: 0.0585 - val_loss: 0.0178 - val_mae: 0.1139 - learning_rate:
5.0000e-04
Epoch 6/50
14/14            0s 9ms/step - loss:
0.0042 - mae: 0.0503 - val_loss: 0.0109 - val_mae: 0.0871 - learning_rate:
5.0000e-04
Epoch 7/50
14/14            0s 9ms/step - loss:
0.0041 - mae: 0.0494 - val_loss: 0.0076 - val_mae: 0.0682 - learning_rate:
5.0000e-04
Epoch 8/50
14/14            0s 8ms/step - loss:
0.0036 - mae: 0.0463 - val_loss: 0.0066 - val_mae: 0.0608 - learning_rate:
5.0000e-04
Epoch 9/50
14/14            0s 8ms/step - loss:
0.0037 - mae: 0.0471 - val_loss: 0.0063 - val_mae: 0.0586 - learning_rate:
5.0000e-04
Epoch 10/50
14/14            0s 9ms/step - loss:
```

```
0.0033 - mae: 0.0442 - val_loss: 0.0062 - val_mae: 0.0585 - learning_rate:
5.0000e-04
Epoch 11/50
14/14            0s 9ms/step - loss:
0.0034 - mae: 0.0445 - val_loss: 0.0061 - val_mae: 0.0575 - learning_rate:
5.0000e-04
Epoch 12/50
14/14            0s 9ms/step - loss:
0.0031 - mae: 0.0434 - val_loss: 0.0063 - val_mae: 0.0591 - learning_rate:
5.0000e-04
Epoch 13/50
14/14            0s 8ms/step - loss:
0.0031 - mae: 0.0434 - val_loss: 0.0062 - val_mae: 0.0584 - learning_rate:
5.0000e-04
Epoch 14/50
 9/14            0s 6ms/step - loss:
0.0030 - mae: 0.0414
Epoch 14: ReduceLROnPlateau reducing learning rate to 0.00035000001662410796.
14/14            0s 9ms/step - loss:
0.0029 - mae: 0.0414 - val_loss: 0.0060 - val_mae: 0.0577 - learning_rate:
5.0000e-04
Epoch 15/50
14/14            0s 9ms/step - loss:
0.0031 - mae: 0.0433 - val_loss: 0.0060 - val_mae: 0.0575 - learning_rate:
3.5000e-04
Epoch 16/50
14/14            0s 9ms/step - loss:
0.0031 - mae: 0.0440 - val_loss: 0.0060 - val_mae: 0.0574 - learning_rate:
3.5000e-04
Epoch 17/50
14/14            0s 9ms/step - loss:
0.0029 - mae: 0.0417 - val_loss: 0.0059 - val_mae: 0.0573 - learning_rate:
3.5000e-04
Epoch 18/50
14/14            0s 9ms/step - loss:
0.0030 - mae: 0.0420 - val_loss: 0.0059 - val_mae: 0.0575 - learning_rate:
3.5000e-04
Epoch 19/50
 9/14            0s 6ms/step - loss:
0.0032 - mae: 0.0434
Epoch 19: ReduceLROnPlateau reducing learning rate to 0.00024500001163687554.
14/14            0s 9ms/step - loss:
0.0031 - mae: 0.0430 - val_loss: 0.0059 - val_mae: 0.0573 - learning_rate:
3.5000e-04
Epoch 20/50
14/14            0s 8ms/step - loss:
0.0031 - mae: 0.0424 - val_loss: 0.0059 - val_mae: 0.0572 - learning_rate:
2.4500e-04
```

```
Epoch 21/50
14/14            0s 9ms/step - loss:
0.0028 - mae: 0.0409 - val_loss: 0.0059 - val_mae: 0.0571 - learning_rate:
2.4500e-04
Epoch 22/50
14/14            0s 9ms/step - loss:
0.0027 - mae: 0.0405 - val_loss: 0.0058 - val_mae: 0.0567 - learning_rate:
2.4500e-04
Epoch 23/50
14/14            0s 9ms/step - loss:
0.0029 - mae: 0.0412 - val_loss: 0.0058 - val_mae: 0.0566 - learning_rate:
2.4500e-04
Epoch 24/50
14/14            0s 9ms/step - loss:
0.0031 - mae: 0.0424 - val_loss: 0.0058 - val_mae: 0.0569 - learning_rate:
2.4500e-04
Epoch 25/50
 9/14            0s 6ms/step - loss:
0.0033 - mae: 0.0443
Epoch 25: ReduceLROnPlateau reducing learning rate to 0.00017150000203400848.
14/14            0s 9ms/step - loss:
0.0032 - mae: 0.0437 - val_loss: 0.0058 - val_mae: 0.0569 - learning_rate:
2.4500e-04
Epoch 26/50
14/14            0s 9ms/step - loss:
0.0028 - mae: 0.0403 - val_loss: 0.0060 - val_mae: 0.0576 - learning_rate:
1.7150e-04
Epoch 27/50
14/14            0s 9ms/step - loss:
0.0027 - mae: 0.0403 - val_loss: 0.0063 - val_mae: 0.0594 - learning_rate:
1.7150e-04
Epoch 28/50
 9/14            0s 6ms/step - loss:
0.0029 - mae: 0.0413
Epoch 28: ReduceLROnPlateau reducing learning rate to 0.00012004999734926967.
14/14            0s 9ms/step - loss:
0.0028 - mae: 0.0410 - val_loss: 0.0061 - val_mae: 0.0583 - learning_rate:
1.7150e-04
Epoch 29/50
14/14            0s 13ms/step -
loss: 0.0027 - mae: 0.0401 - val_loss: 0.0062 - val_mae: 0.0591 - learning_rate:
1.2005e-04
Epoch 30/50
14/14            0s 9ms/step - loss:
0.0026 - mae: 0.0393 - val_loss: 0.0061 - val_mae: 0.0586 - learning_rate:
1.2005e-04
Epoch 31/50
 9/14            0s 7ms/step - loss:
```

```
0.0026 - mae: 0.0386
Epoch 31: ReduceLROnPlateau reducing learning rate to 8.403499814448878e-05.
14/14             0s 9ms/step - loss:
0.0025 - mae: 0.0380 - val_loss: 0.0061 - val_mae: 0.0586 - learning_rate:
1.2005e-04
Epoch 1/50
```

```
2025-05-10 21:50:05.380287: E tensorflow/core/framework/node_def_util.cc:680]
NodeDef mentions attribute use_unbounded_threadpool which is not in the op
definition: Op<name=MapDataset; signature=input_dataset:variant,
other_arguments: -> handle:variant; attr=f:func;
attr=Targuments:list(type),min=0; attr=output_types:list(type),min=1;
attr=output_shapes:list(shape),min=1;
attr=use_inter_op_parallelism:bool,default=true;
attr=preserve_cardinality:bool,default=false;
attr=force_synchronous:bool,default=false; attr=metadata:string,default=""> This
may be expected if your graph generating binary is newer  than this binary.
Unknown attributes will be ignored. NodeDef: {{node ParallelMapDatasetV2/_15}}
2025-05-10 21:50:05.380653: E tensorflow/core/framework/node_def_util.cc:680]
NodeDef mentions attribute use_unbounded_threadpool which is not in the op
definition: Op<name=MapDataset; signature=input_dataset:variant,
other_arguments: -> handle:variant; attr=f:func;
attr=Targuments:list(type),min=0; attr=output_types:list(type),min=1;
attr=output_shapes:list(shape),min=1;
attr=use_inter_op_parallelism:bool,default=true;
attr=preserve_cardinality:bool,default=false;
attr=force_synchronous:bool,default=false; attr=metadata:string,default=""> This
may be expected if your graph generating binary is newer  than this binary.
Unknown attributes will be ignored. NodeDef: {{node ParallelMapDatasetV2/_15}}
```

```
14/14             2s 26ms/step -
loss: 0.1986 - mae: 0.3746 - val_loss: 0.0223 - val_mae: 0.1270 - learning_rate:
5.0000e-04
Epoch 2/50
14/14             0s 9ms/step - loss:
0.0262 - mae: 0.1320 - val_loss: 0.0524 - val_mae: 0.1963 - learning_rate:
5.0000e-04
Epoch 3/50
14/14             0s 9ms/step - loss:
0.0118 - mae: 0.0881 - val_loss: 0.0615 - val_mae: 0.2217 - learning_rate:
5.0000e-04
Epoch 4/50
 9/14             0s 6ms/step - loss:
0.0075 - mae: 0.0664
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.00035000001662410796.
14/14             0s 9ms/step - loss:
0.0070 - mae: 0.0640 - val_loss: 0.0267 - val_mae: 0.1430 - learning_rate:
5.0000e-04
Epoch 5/50
```

```
14/14           0s 9ms/step - loss:
0.0048 - mae: 0.0550 - val_loss: 0.0142 - val_mae: 0.1010 - learning_rate:
3.5000e-04
Epoch 6/50
14/14           0s 9ms/step - loss:
0.0038 - mae: 0.0477 - val_loss: 0.0075 - val_mae: 0.0674 - learning_rate:
3.5000e-04
Epoch 7/50
14/14           0s 8ms/step - loss:
0.0037 - mae: 0.0470 - val_loss: 0.0065 - val_mae: 0.0599 - learning_rate:
3.5000e-04
Epoch 8/50
14/14           0s 9ms/step - loss:
0.0035 - mae: 0.0461 - val_loss: 0.0063 - val_mae: 0.0587 - learning_rate:
3.5000e-04
Epoch 9/50
14/14           0s 9ms/step - loss:
0.0034 - mae: 0.0446 - val_loss: 0.0062 - val_mae: 0.0583 - learning_rate:
3.5000e-04
Epoch 10/50
14/14           0s 9ms/step - loss:
0.0032 - mae: 0.0432 - val_loss: 0.0061 - val_mae: 0.0582 - learning_rate:
3.5000e-04
Epoch 11/50
14/14           0s 10ms/step -
loss: 0.0030 - mae: 0.0418 - val_loss: 0.0062 - val_mae: 0.0587 - learning_rate:
3.5000e-04
Epoch 12/50
 9/14           0s 7ms/step - loss:
0.0029 - mae: 0.0407
Epoch 12: ReduceLROnPlateau reducing learning rate to 0.00024500001163687554.
14/14           0s 9ms/step - loss:
0.0029 - mae: 0.0409 - val_loss: 0.0061 - val_mae: 0.0582 - learning_rate:
3.5000e-04
Epoch 13/50
14/14           0s 10ms/step -
loss: 0.0030 - mae: 0.0421 - val_loss: 0.0061 - val_mae: 0.0584 - learning_rate:
2.4500e-04
Epoch 14/50
14/14           0s 9ms/step - loss:
0.0030 - mae: 0.0425 - val_loss: 0.0061 - val_mae: 0.0587 - learning_rate:
2.4500e-04
Epoch 15/50
14/14           0s 9ms/step - loss:
0.0031 - mae: 0.0433 - val_loss: 0.0061 - val_mae: 0.0583 - learning_rate:
2.4500e-04
Epoch 16/50
14/14           0s 9ms/step - loss:
```

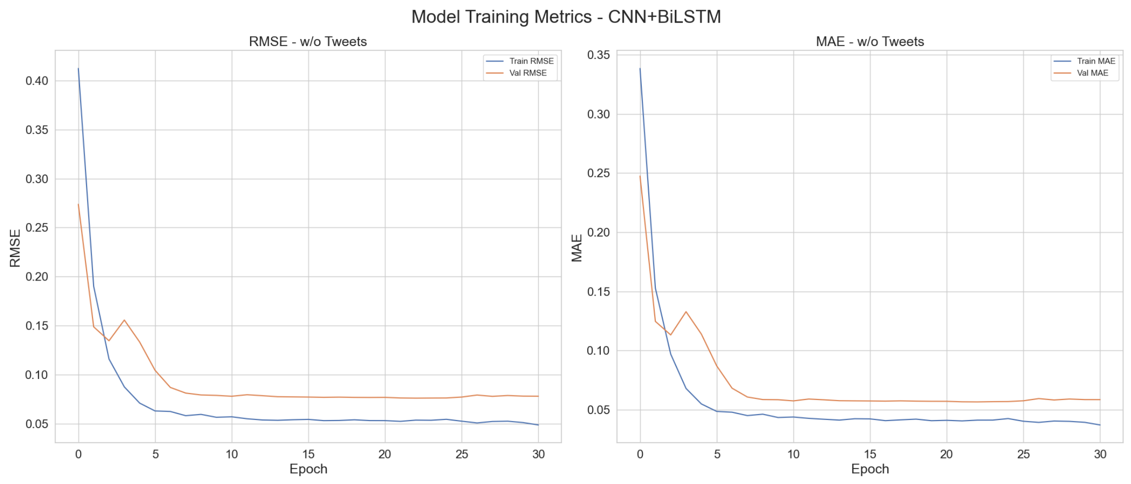
```
0.0029 - mae: 0.0412 - val_loss: 0.0060 - val_mae: 0.0576 - learning_rate:
2.4500e-04
Epoch 17/50
14/14            0s 9ms/step - loss:
0.0029 - mae: 0.0423 - val_loss: 0.0062 - val_mae: 0.0588 - learning_rate:
2.4500e-04
Epoch 18/50
14/14            0s 9ms/step - loss:
0.0031 - mae: 0.0422 - val_loss: 0.0059 - val_mae: 0.0575 - learning_rate:
2.4500e-04
Epoch 19/50
 9/14            0s 7ms/step - loss:
0.0028 - mae: 0.0409
Epoch 19: ReduceLROnPlateau reducing learning rate to 0.00017150000203400848.
14/14            0s 9ms/step - loss:
0.0027 - mae: 0.0403 - val_loss: 0.0060 - val_mae: 0.0578 - learning_rate:
2.4500e-04
Epoch 20/50
14/14            0s 9ms/step - loss:
0.0027 - mae: 0.0402 - val_loss: 0.0059 - val_mae: 0.0575 - learning_rate:
1.7150e-04
Epoch 21/50
14/14            0s 10ms/step -
loss: 0.0029 - mae: 0.0408 - val_loss: 0.0058 - val_mae: 0.0568 - learning_rate:
1.7150e-04
Epoch 22/50
14/14            0s 9ms/step - loss:
0.0028 - mae: 0.0404 - val_loss: 0.0059 - val_mae: 0.0575 - learning_rate:
1.7150e-04
Epoch 23/50
14/14            0s 10ms/step -
loss: 0.0031 - mae: 0.0427 - val_loss: 0.0060 - val_mae: 0.0583 - learning_rate:
1.7150e-04
Epoch 24/50
 9/14            0s 7ms/step - loss:
0.0032 - mae: 0.0441
Epoch 24: ReduceLROnPlateau reducing learning rate to 0.00012004999734926967.
14/14            0s 10ms/step -
loss: 0.0031 - mae: 0.0434 - val_loss: 0.0059 - val_mae: 0.0572 - learning_rate:
1.7150e-04
Epoch 25/50
14/14            0s 10ms/step -
loss: 0.0030 - mae: 0.0422 - val_loss: 0.0059 - val_mae: 0.0573 - learning_rate:
1.2005e-04
Epoch 26/50
14/14            0s 9ms/step - loss:
0.0030 - mae: 0.0419 - val_loss: 0.0062 - val_mae: 0.0591 - learning_rate:
1.2005e-04
```

```
Epoch 27/50
 9/14             0s 6ms/step - loss:
0.0027 - mae: 0.0399
Epoch 27: ReduceLROnPlateau reducing learning rate to 8.403499814448878e-05.
14/14             0s 9ms/step - loss:
0.0027 - mae: 0.0396 - val_loss: 0.0063 - val_mae: 0.0595 - learning_rate:
1.2005e-04
Epoch 28/50
14/14             0s 10ms/step -
loss: 0.0028 - mae: 0.0410 - val_loss: 0.0063 - val_mae: 0.0598 - learning_rate:
8.4035e-05
Epoch 29/50
14/14             0s 10ms/step -
loss: 0.0027 - mae: 0.0402 - val_loss: 0.0059 - val_mae: 0.0574 - learning_rate:
8.4035e-05
```

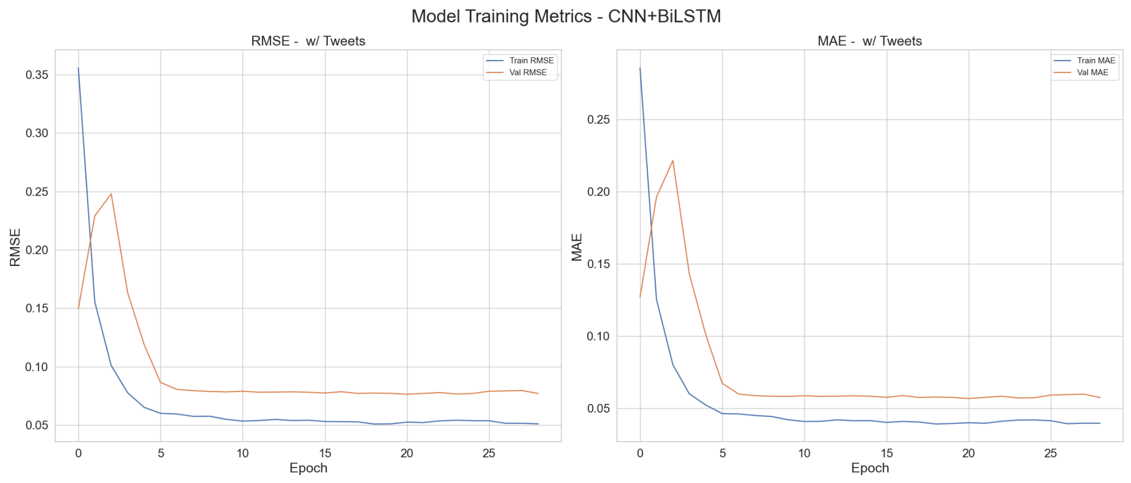
[156]:
```
plot_metrics(cnnBiLSTM_woSent, history_cnnBiLSTM_woSent,
             trainX_wo_tweet, trainY_wo_tweet,
             testX_wo_tweet, testY_wo_tweet,
             "without")

plot_metrics(cnnBiLSTM_withSent, history_cnnBiLSTM_withSent,
             trainX_with_tweet, trainY_with_tweet,
             testX_with_tweet, testY_with_tweet,
             "with")
```

```
2025-05-10 21:50:11.310491: E tensorflow/core/framework/node_def_util.cc:680]
NodeDef mentions attribute use_unbounded_threadpool which is not in the op
definition: Op<name=MapDataset; signature=input_dataset:variant,
other_arguments: -> handle:variant; attr=f:func;
attr=Targuments:list(type),min=0; attr=output_types:list(type),min=1;
attr=output_shapes:list(shape),min=1;
attr=use_inter_op_parallelism:bool,default=true;
attr=preserve_cardinality:bool,default=false;
attr=force_synchronous:bool,default=false; attr=metadata:string,default=""> This
may be expected if your graph generating binary is newer  than this binary.
Unknown attributes will be ignored. NodeDef: {{node ParallelMapDatasetV2/_14}}
2025-05-10 21:50:11.310805: E tensorflow/core/framework/node_def_util.cc:680]
NodeDef mentions attribute use_unbounded_threadpool which is not in the op
definition: Op<name=MapDataset; signature=input_dataset:variant,
other_arguments: -> handle:variant; attr=f:func;
attr=Targuments:list(type),min=0; attr=output_types:list(type),min=1;
attr=output_shapes:list(shape),min=1;
attr=use_inter_op_parallelism:bool,default=true;
attr=preserve_cardinality:bool,default=false;
attr=force_synchronous:bool,default=false; attr=metadata:string,default=""> This
may be expected if your graph generating binary is newer  than this binary.
Unknown attributes will be ignored. NodeDef: {{node ParallelMapDatasetV2/_14}}
```

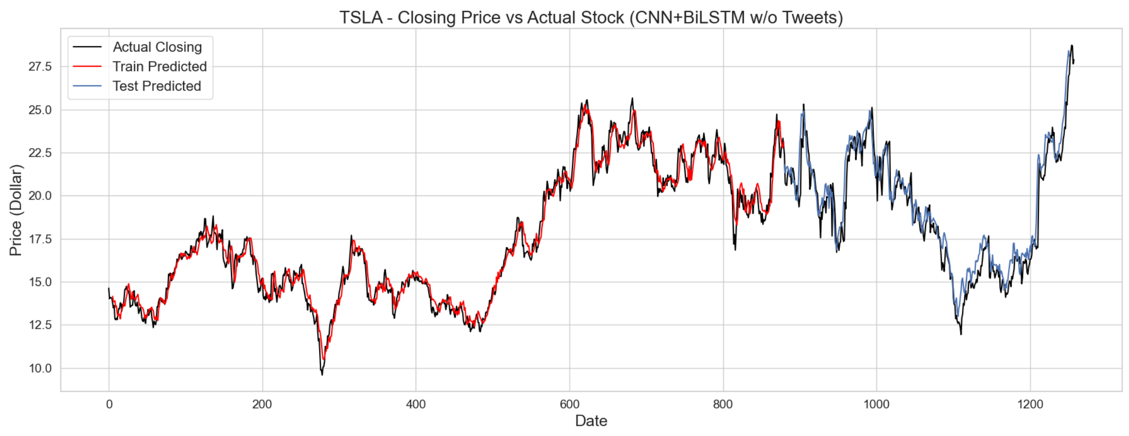
```
w/o Tweets RMSE: Train = 0.0465, Test = 0.0780
w/o Tweets MAE : Train = 0.0361, Test = 0.0586
```

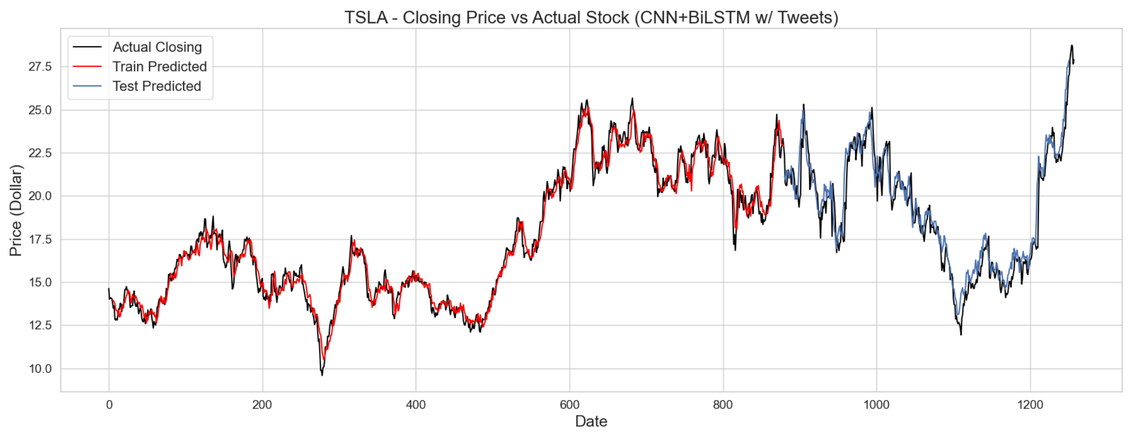


```
 w/ Tweets RMSE: Train = 0.0454, Test = 0.0770
 w/ Tweets MAE : Train = 0.0350, Test = 0.0574
```



[157]:
```
plot_pred_vs_actual(
    model=cnnBiLSTM_woSent,
    X_train=trainX_wo_tweet, y_train=trainY_wo_tweet,
    X_test=testX_wo_tweet,   y_test=testY_wo_tweet,
    model_name="CNN+BiLSTM",
    sentiment_mode="without"
)

plot_pred_vs_actual(
    model=cnnBiLSTM_withSent,
    X_train=trainX_with_tweet, y_train=trainY_with_tweet,
    X_test=testX_with_tweet,   y_test=testY_with_tweet,
    model_name="CNN+BiLSTM",
    sentiment_mode="with"
)
```





2.0.3 Transformer

Pre-load Plotting Functions

```
[93]: def plot_transformer_metrics(model, history, X_train, y_train, X_val, y_val,␣
       ↪sentiment_mode):
          """
          Plot training and validation RMSE/MAE metrics and loss curves for␣
       ↪Transformer models.
          """
          history_data = history.history
```

```
    # 1. Predictions
    y_train_pred = model.predict(X_train, verbose=0).squeeze()
    y_val_pred   = model.predict(X_val, verbose=0).squeeze()

    y_train_true = y_train.squeeze()
    y_val_true   = y_val.squeeze()

    # 2. Metrics
    train_rmse = np.sqrt(mean_squared_error(y_train_true, y_train_pred))
    val_rmse   = np.sqrt(mean_squared_error(y_val_true, y_val_pred))
    train_mae  = mean_absolute_error(y_train_true, y_train_pred)
    val_mae    = mean_absolute_error(y_val_true, y_val_pred)

    title_suffix = " w/ Tweets" if sentiment_mode == "with" else "w/o Tweets"

    # 3. Print metrics
    print(f"  [Transformer{title_suffix}] RMSE: Train = {train_rmse:.4f}, Test␣
↪= {val_rmse:.4f}")
    print(f"  [Transformer{title_suffix}] MAE : Train = {train_mae:.4f}, Test =␣
↪{val_mae:.4f}")

    # 4. Plot
    plt.figure(figsize=(21, 9))

    # ---- RMSE subplot
    plt.subplot(1, 2, 1)
    plt.plot(np.sqrt(history_data['loss']), label='Train RMSE')
    plt.plot(np.sqrt(history_data['val_loss']), label='Val RMSE')
    plt.title(f'RMSE - Transformer {title_suffix}', fontsize=18)
    plt.xlabel('Epoch', fontsize=18)
    plt.ylabel('RMSE', fontsize=18)
    plt.xticks(fontsize=16)
    plt.yticks(fontsize=16)
    plt.legend()

    # ---- MAE subplot
    plt.subplot(1, 2, 2)
    if 'mae' in history_data:
        plt.plot(history_data['mae'], label='Train MAE')
        plt.plot(history_data['val_mae'], label='Val MAE')
    else:
        plt.plot([train_mae] * len(history_data['loss']), label='Train MAE␣
↪(flat)')
        plt.plot([val_mae]   * len(history_data['val_loss']), label='Val MAE␣
↪(flat)')
    plt.title(f'MAE - Transformer {title_suffix}', fontsize=18)
    plt.xlabel('Epoch', fontsize=18)
    plt.ylabel('MAE', fontsize=18)
    plt.xticks(fontsize=16)
    plt.yticks(fontsize=16)
    plt.legend()

    plt.suptitle('Model Training Metrics - Transformer', fontsize=24)
    plt.tight_layout()
    plt.show()
```

[94]:
```
def plot_transformer_predictions(dates, true_prices, train_preds, test_preds,␣
 ↪train_size, title):
    """
    Plot true prices vs. transformer predictions using seaborn styling.

    Args:
        dates: full date index
        true_prices: full true values (1D)
        train_preds: model predictions on training data
        test_preds: model predictions on test data
        train_size: number of training samples (for separation)
        title: chart title
    """
    # Set seaborn style
    sns.set_theme(style="whitegrid")

    # Create figure
    plt.figure(figsize=(18, 7))

    # Plot actual prices
    sns.lineplot(x=dates, y=true_prices, color='black', label='Actual Closing')

    # Plot transformer predictions
    sns.lineplot(x=dates[:train_size], y=train_preds, color='red', label='Train␣
 ↪Predicted')

    # Plot test predictions with markers
    sns.lineplot(x=dates[train_size:], y=test_preds,
                 label='Test Predicted')

    # Add shaded region to distinguish train/test split
    plt.axvline(x=dates[train_size-1], color='gray', linestyle='--', alpha=0.7)

    # Styling
    plt.title(title, fontsize=20)
    plt.xlabel('Date (Jan 2020 - July 2022)', fontsize=18)
    plt.ylabel('Price (Dollar)', fontsize=18)
    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)
    plt.legend(fontsize=16)
    plt.grid(True)
    plt.tight_layout()

    return plt
```

Model Presetting & Building

[95]:
```
class PositionalEncoding(tf.keras.layers.Layer):
    def __init__(self, sequence_len, d_model):
        super().__init__()
        self.pos_encoding = self.positional_encoding(sequence_len, d_model)

    def get_config(self):
        return {"sequence_len": self.pos_encoding.shape[0], "d_model": self.
 ↪pos_encoding.shape[1]}

    def positional_encoding(self, position, d_model):
        angle_rads = self.get_angles(np.arange(position)[:, np.newaxis],
                                     np.arange(d_model)[np.newaxis, :],
                                     d_model)
        angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])
        angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])
        return tf.cast(angle_rads[np.newaxis, ...], dtype=tf.float32)

    def get_angles(self, pos, i, d_model):
        angle_rates = 1 / np.power(10000, (2 * (i // 2)) / np.float32(d_model))
        return pos * angle_rates

    def call(self, x):
        return x + self.pos_encoding[:, :tf.shape(x)[1], :]
```

[96]:
```
# ========== 1. Transformer Encoder ==========
def transformer_encoder(inputs, head_size, num_heads, ff_dim, dropout):
    """
    Builds a single Transformer encoder block.
    """
    # Multi-head self-attention
    attention_output = MultiHeadAttention(num_heads=num_heads,␣
 ↪key_dim=head_size, dropout=dropout)(inputs, inputs)
    attention_output = Add()([inputs, attention_output])
    attention_output = LayerNormalization()(attention_output)

    # Feed-forward network
    ffn_output = Dense(ff_dim, activation='relu')(attention_output)
    # ffn_output = Dropout(dropout)(ffn_output)  # <--- Dropout after first FFN␣
↪layer
    ffn_output = Dense(inputs.shape[-1])(ffn_output)
    ffn_output = Add()([attention_output, ffn_output])
    output = LayerNormalization()(ffn_output)

    return output
```

[97]:
```
def build_transformer_model(input_shape, head_size=64, num_heads=4, ff_dim=128,␣
↪num_layers=2, dropout=0.1):
    inputs = Input(shape=input_shape)
    x = PositionalEncoding(input_shape[0], input_shape[1])(inputs)  # add␣
↪positional encoding

    for _ in range(num_layers):
        x = transformer_encoder(x, head_size, num_heads, ff_dim, dropout)

    x = GlobalAveragePooling1D()(x)
    outputs = Dense(1)(x)
    return Model(inputs, outputs)
```

[98]:
```
def directional_loss(y_true, y_pred):
    sign_true = K.sign(y_true[1:] - y_true[:-1])
    sign_pred = K.sign(y_pred[1:] - y_pred[:-1])
    correct_direction = K.cast(K.equal(sign_true, sign_pred), dtype=tf.float32)
    return 1.0 - K.mean(correct_direction)

def integrated_loss(delta=0.01, lambda_dir=0.5):
    huber = Huber(delta=delta)

    def loss(y_true, y_pred):
        huber_loss = huber(y_true, y_pred)
        dir_loss = directional_loss(y_true, y_pred)
        return huber_loss + lambda_dir * dir_loss

    return loss
```

[99]:
```
# For Transformer *without* sentiment (cleaner input, stop sooner)
early_stop_wo = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)

# For Transformer *with* sentiment (noisier input, allow more time)
early_stop_with = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.75,         # ← good default
    patience=3,
    min_lr=1e-5,
    verbose=1
)
```

Model Training & Evaluating

[100]:
```
# ========== Train model without sentiment ==========
model_wo_sent = build_transformer_model((trainX_wo_tweet.shape[1],␣
  ↪trainX_wo_tweet.shape[2]))
model_wo_sent.compile(
    optimizer=Adam(0.001),
    # loss=integrated_loss(),
    loss=Huber(0.01),
    # loss='mse',
    metrics=['mae']
)
history_wo_sent = model_wo_sent.fit(  #   save history here
    trainX_wo_tweet, trainY_wo_tweet,
    validation_data=(testX_wo_tweet, testY_wo_tweet),
    epochs=50, batch_size=64, verbose=1,
    callbacks=[early_stop_wo]
)

# ========== Train model with sentiment ==========
model_with_sent = build_transformer_model((trainX_with_tweet.shape[1],␣
  ↪trainX_with_tweet.shape[2]))
model_with_sent.compile(
    optimizer=Adam(0.001),
    # loss=integrated_loss(),
    loss=Huber(0.01),
    # loss='mse',
    metrics=['mae']
)
history_with_sent = model_with_sent.fit(  #   save history here
    trainX_with_tweet, trainY_with_tweet,
    validation_data=(testX_with_tweet, testY_with_tweet),
    epochs=50, batch_size=64, verbose=1,
    callbacks=[early_stop_with]
)
```

```
Epoch 1/50
14/14            2s 19ms/step -
loss: 0.0032 - mae: 0.3247 - val_loss: 0.0049 - val_mae: 0.4907
Epoch 2/50
14/14            0s 6ms/step - loss:
0.0011 - mae: 0.1165 - val_loss: 0.0038 - val_mae: 0.3814
Epoch 3/50
14/14            0s 6ms/step - loss:
6.6015e-04 - mae: 0.0709 - val_loss: 0.0030 - val_mae: 0.3041
Epoch 4/50
14/14            0s 6ms/step - loss:
5.4238e-04 - mae: 0.0590 - val_loss: 0.0030 - val_mae: 0.3053
Epoch 5/50
14/14            0s 7ms/step - loss:
4.4475e-04 - mae: 0.0492 - val_loss: 0.0023 - val_mae: 0.2378
Epoch 6/50
14/14            0s 6ms/step - loss:
3.6234e-04 - mae: 0.0410 - val_loss: 0.0020 - val_mae: 0.2000
Epoch 7/50
14/14            0s 6ms/step - loss:
3.4177e-04 - mae: 0.0389 - val_loss: 0.0018 - val_mae: 0.1817
Epoch 8/50
14/14            0s 6ms/step - loss:
3.4357e-04 - mae: 0.0391 - val_loss: 0.0017 - val_mae: 0.1751
Epoch 9/50
14/14            0s 6ms/step - loss:
2.7825e-04 - mae: 0.0324 - val_loss: 0.0015 - val_mae: 0.1581
Epoch 10/50
14/14            0s 6ms/step - loss:
2.9995e-04 - mae: 0.0346 - val_loss: 0.0017 - val_mae: 0.1704
Epoch 11/50
14/14            0s 6ms/step - loss:
3.0187e-04 - mae: 0.0349 - val_loss: 0.0015 - val_mae: 0.1548
Epoch 12/50
14/14            0s 6ms/step - loss:
2.9833e-04 - mae: 0.0345 - val_loss: 0.0018 - val_mae: 0.1818
Epoch 13/50
14/14            0s 6ms/step - loss:
3.3264e-04 - mae: 0.0379 - val_loss: 0.0017 - val_mae: 0.1767
Epoch 14/50
14/14            0s 6ms/step - loss:
2.6231e-04 - mae: 0.0308 - val_loss: 0.0016 - val_mae: 0.1627
Epoch 15/50
14/14            0s 6ms/step - loss:
3.2832e-04 - mae: 0.0376 - val_loss: 0.0016 - val_mae: 0.1661
Epoch 16/50
14/14            0s 7ms/step - loss:
2.5449e-04 - mae: 0.0301 - val_loss: 0.0016 - val_mae: 0.1694
```

```
Epoch 17/50
14/14            0s 6ms/step - loss:
2.7043e-04 - mae: 0.0317 - val_loss: 0.0015 - val_mae: 0.1599
Epoch 18/50
14/14            0s 6ms/step - loss:
3.1782e-04 - mae: 0.0365 - val_loss: 0.0013 - val_mae: 0.1374
Epoch 19/50
14/14            0s 6ms/step - loss:
3.7832e-04 - mae: 0.0426 - val_loss: 0.0013 - val_mae: 0.1358
Epoch 20/50
14/14            0s 6ms/step - loss:
2.9102e-04 - mae: 0.0337 - val_loss: 0.0016 - val_mae: 0.1673
Epoch 21/50
14/14            0s 6ms/step - loss:
2.6692e-04 - mae: 0.0313 - val_loss: 0.0017 - val_mae: 0.1762
Epoch 22/50
14/14            0s 6ms/step - loss:
2.8483e-04 - mae: 0.0331 - val_loss: 0.0016 - val_mae: 0.1664
Epoch 23/50
14/14            0s 6ms/step - loss:
2.4354e-04 - mae: 0.0290 - val_loss: 0.0017 - val_mae: 0.1749
Epoch 24/50
14/14            0s 6ms/step - loss:
2.1451e-04 - mae: 0.0261 - val_loss: 0.0017 - val_mae: 0.1715
Epoch 25/50
14/14            0s 6ms/step - loss:
2.3525e-04 - mae: 0.0282 - val_loss: 0.0018 - val_mae: 0.1808
Epoch 26/50
14/14            0s 6ms/step - loss:
2.2279e-04 - mae: 0.0269 - val_loss: 0.0017 - val_mae: 0.1799
Epoch 27/50
14/14            0s 5ms/step - loss:
2.1414e-04 - mae: 0.0259 - val_loss: 0.0017 - val_mae: 0.1731
Epoch 28/50
14/14            0s 6ms/step - loss:
2.0664e-04 - mae: 0.0252 - val_loss: 0.0020 - val_mae: 0.2022
Epoch 29/50
14/14            0s 5ms/step - loss:
2.5003e-04 - mae: 0.0297 - val_loss: 0.0019 - val_mae: 0.1957
Epoch 1/50

2025-05-10 13:51:53.273245: E tensorflow/core/framework/node_def_util.cc:680]
NodeDef mentions attribute use_unbounded_threadpool which is not in the op
definition: Op<name=MapDataset; signature=input_dataset:variant,
other_arguments: -> handle:variant; attr=f:func;
attr=Targuments:list(type),min=0; attr=output_types:list(type),min=1;
attr=output_shapes:list(shape),min=1;
attr=use_inter_op_parallelism:bool,default=true;
```

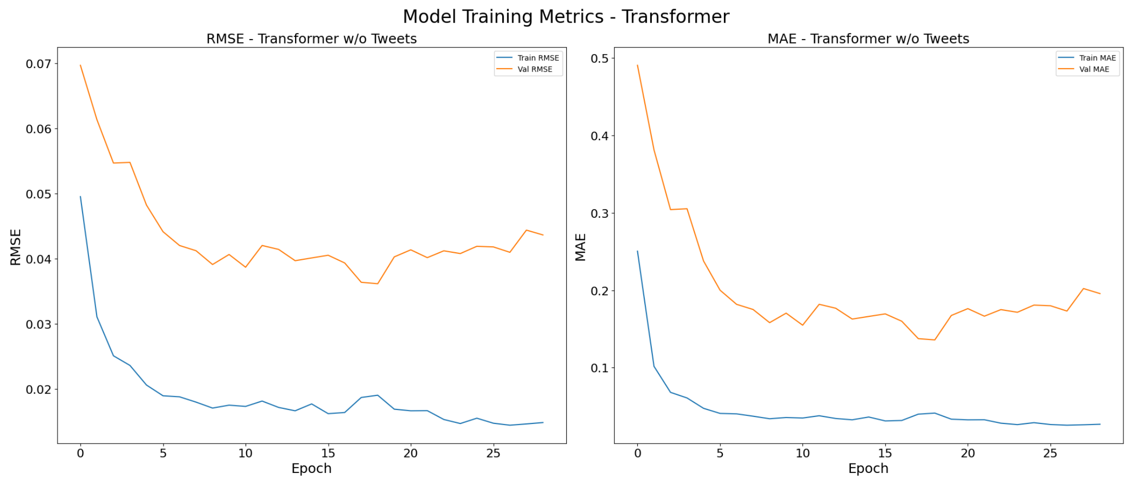
```
attr=preserve_cardinality:bool,default=false;
attr=force_synchronous:bool,default=false; attr=metadata:string,default=""> This
may be expected if your graph generating binary is newer  than this binary.
Unknown attributes will be ignored. NodeDef: {{node ParallelMapDatasetV2/_15}}
2025-05-10 13:51:53.273545: E tensorflow/core/framework/node_def_util.cc:680]
NodeDef mentions attribute use_unbounded_threadpool which is not in the op
definition: Op<name=MapDataset; signature=input_dataset:variant,
other_arguments: -> handle:variant; attr=f:func;
attr=Targuments:list(type),min=0; attr=output_types:list(type),min=1;
attr=output_shapes:list(shape),min=1;
attr=use_inter_op_parallelism:bool,default=true;
attr=preserve_cardinality:bool,default=false;
attr=force_synchronous:bool,default=false; attr=metadata:string,default=""> This
may be expected if your graph generating binary is newer  than this binary.
Unknown attributes will be ignored. NodeDef: {{node ParallelMapDatasetV2/_15}}
```

```
14/14            2s 18ms/step -
loss: 0.0030 - mae: 0.3078 - val_loss: 0.0059 - val_mae: 0.5982
Epoch 2/50
14/14            0s 7ms/step - loss:
0.0011 - mae: 0.1116 - val_loss: 0.0015 - val_mae: 0.1580
Epoch 3/50
14/14            0s 7ms/step - loss:
6.0356e-04 - mae: 0.0652 - val_loss: 0.0018 - val_mae: 0.1837
Epoch 4/50
14/14            0s 7ms/step - loss:
4.0895e-04 - mae: 0.0457 - val_loss: 0.0012 - val_mae: 0.1277
Epoch 5/50
14/14            0s 7ms/step - loss:
3.1472e-04 - mae: 0.0362 - val_loss: 8.5065e-04 - val_mae: 0.0900
Epoch 6/50
14/14            0s 7ms/step - loss:
2.8180e-04 - mae: 0.0328 - val_loss: 7.5442e-04 - val_mae: 0.0804
Epoch 7/50
14/14            0s 6ms/step - loss:
2.4849e-04 - mae: 0.0295 - val_loss: 8.4502e-04 - val_mae: 0.0893
Epoch 8/50
14/14            0s 6ms/step - loss:
2.8955e-04 - mae: 0.0337 - val_loss: 0.0012 - val_mae: 0.1210
Epoch 9/50
14/14            0s 6ms/step - loss:
2.3295e-04 - mae: 0.0279 - val_loss: 8.5472e-04 - val_mae: 0.0904
Epoch 10/50
14/14            0s 6ms/step - loss:
2.6701e-04 - mae: 0.0314 - val_loss: 0.0014 - val_mae: 0.1423
Epoch 11/50
14/14            0s 6ms/step - loss:
2.3613e-04 - mae: 0.0282 - val_loss: 0.0012 - val_mae: 0.1222
```

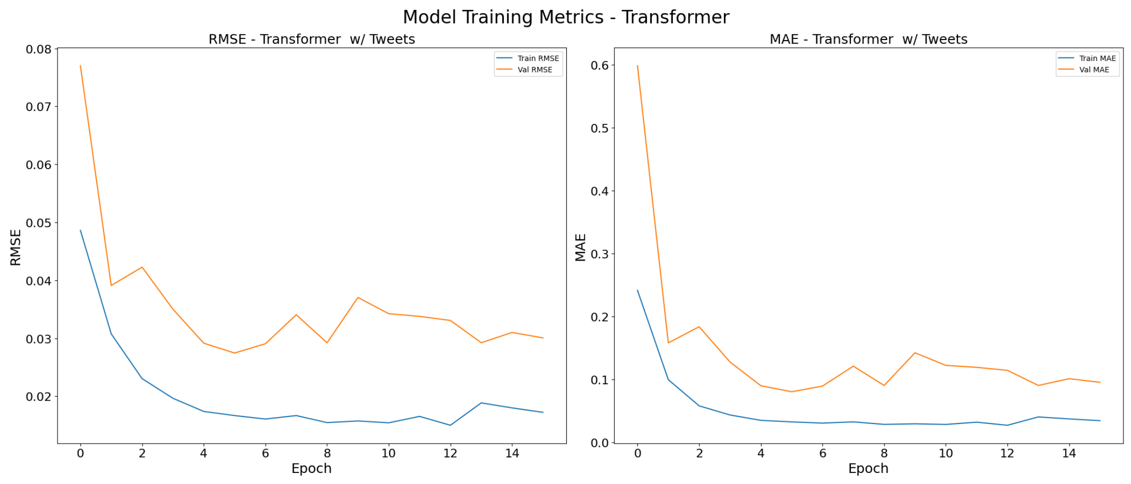
```
Epoch 12/50
14/14          0s 6ms/step - loss:
2.8167e-04 - mae: 0.0329 - val_loss: 0.0011 - val_mae: 0.1191
Epoch 13/50
14/14          0s 6ms/step - loss:
2.0412e-04 - mae: 0.0250 - val_loss: 0.0011 - val_mae: 0.1144
Epoch 14/50
14/14          0s 6ms/step - loss:
3.3401e-04 - mae: 0.0381 - val_loss: 8.5530e-04 - val_mae: 0.0903
Epoch 15/50
14/14          0s 6ms/step - loss:
3.3361e-04 - mae: 0.0381 - val_loss: 9.6195e-04 - val_mae: 0.1011
Epoch 16/50
14/14          0s 6ms/step - loss:
2.9512e-04 - mae: 0.0342 - val_loss: 9.0472e-04 - val_mae: 0.0954
```

[101]:
```
plot_transformer_metrics(
    model=model_wo_sent,
    history=history_wo_sent,
    X_train=trainX_wo_tweet, y_train=trainY_wo_tweet,
    X_val=testX_wo_tweet,   y_val=testY_wo_tweet,
    sentiment_mode="without"
)

plot_transformer_metrics(
    model=model_with_sent,
    history=history_with_sent,
    X_train=trainX_with_tweet, y_train=trainY_with_tweet,
    X_val=testX_with_tweet,   y_val=testY_with_tweet,
    sentiment_mode="with"
)
```

```
[Transformerw/o Tweets] RMSE: Train = 0.0376, Test = 0.1780
[Transformerw/o Tweets] MAE : Train = 0.0295, Test = 0.1358
```



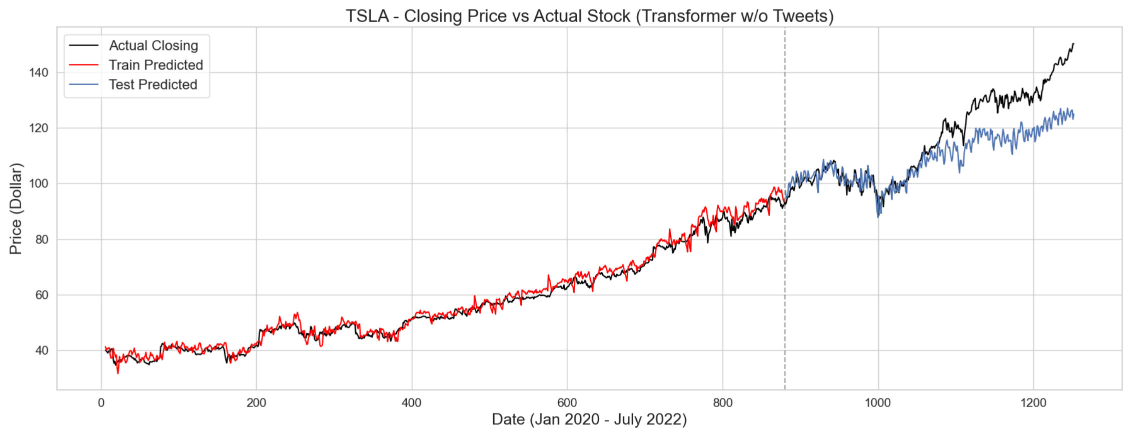
```
[Transformer w/ Tweets] RMSE: Train = 0.0345, Test = 0.0944
[Transformer w/ Tweets] MAE : Train = 0.0259, Test = 0.0804
```

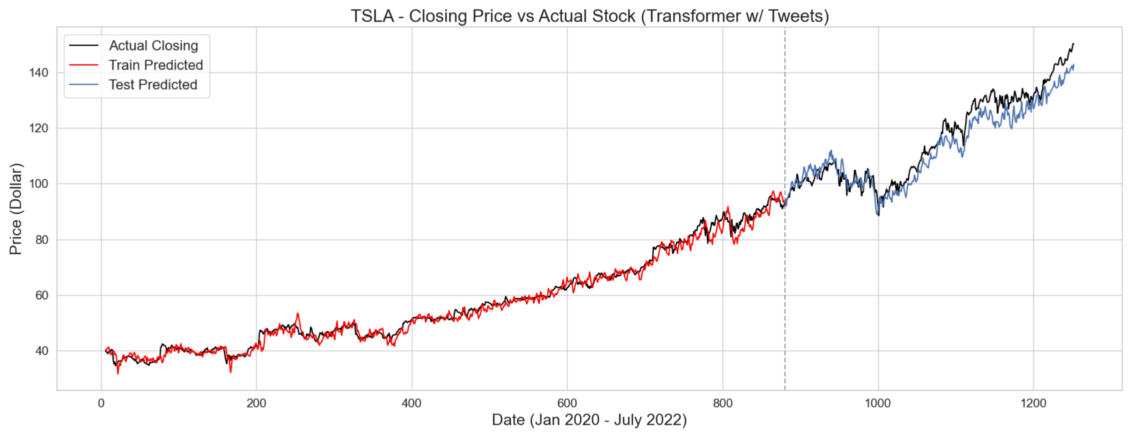


[102]:
```python
# ========== 1. Get Raw Predictions ==========

# Model without sentiment
train_preds_raw_wo_sent = model_wo_sent.predict(trainX_wo_tweet).reshape(-1, 1)
test_preds_raw_wo_sent  = model_wo_sent.predict(testX_wo_tweet).reshape(-1, 1)

# Model with sentiment
train_preds_raw_with_sent = model_with_sent.predict(trainX_with_tweet).
  ↪reshape(-1, 1)
test_preds_raw_with_sent  = model_with_sent.predict(testX_with_tweet).
  ↪reshape(-1, 1)

# ========== 2. Inverse Transform Predictions ==========

def safe_inverse(preds_scaled, full_scaler, target_index=0):
    """
    Inverse-transform predictions scaled with a full-feature MinMaxScaler.
    Assumes predictions correspond to feature at `target_index`.
    """
    pad_width = full_scaler.n_features_in_ - 1
    padded = np.concatenate([preds_scaled, np.zeros((preds_scaled.shape[0],␣
  ↪pad_width))], axis=1)
    return full_scaler.inverse_transform(padded)[:, target_index].reshape(-1, 1)

# Use the safe inverse
```

```python
inv_train_preds_wo_sent = safe_inverse(train_preds_raw_wo_sent, scaler,␣
 ↪target_index=0)
inv_test_preds_wo_sent  = safe_inverse(test_preds_raw_wo_sent,  scaler,␣
 ↪target_index=0)

inv_train_preds_with_sent = safe_inverse(train_preds_raw_with_sent, scaler,␣
 ↪target_index=0)
inv_test_preds_with_sent  = safe_inverse(test_preds_raw_with_sent,  scaler,␣
 ↪target_index=0)

# ========== 3. Prepare Date Range and Ground Truth ==========

# For model without sentiment
train_size_wo = len(inv_train_preds_wo_sent)
total_preds_wo = train_size_wo + len(inv_test_preds_wo_sent)
dates_wo = df_filtered.index[n_past:n_past + total_preds_wo]
true_adj_close_wo = df_filtered['Adj Close'].values[n_past:n_past +␣
 ↪total_preds_wo]

# For model with sentiment
train_size_with = len(inv_train_preds_with_sent)
total_preds_with = train_size_with + len(inv_test_preds_with_sent)
dates_with = df_filtered.index[n_past:n_past + total_preds_with]
true_adj_close_with = df_filtered['Adj Close'].values[n_past:n_past +␣
 ↪total_preds_with]

# ========== 4. Plot Predictions ==========

# Plot: WITHOUT Sentiment
plot_transformer_predictions(
    dates=dates_wo,
    true_prices=true_adj_close_wo,
    train_preds=inv_train_preds_wo_sent.flatten(),
    test_preds=inv_test_preds_wo_sent.flatten(),
    train_size=train_size_wo,
    title="TSLA - Closing Price vs Actual Stock (Transformer w/o Tweets)"
).show()

# Plot: WITH Sentiment
plot_transformer_predictions(
    dates=dates_with,
    true_prices=true_adj_close_with,
    train_preds=inv_train_preds_with_sent.flatten(),
    test_preds=inv_test_preds_with_sent.flatten(),
    train_size=train_size_with,
    title="TSLA - Closing Price vs Actual Stock (Transformer w/ Tweets)"
).show()
```

```
28/28                 0s 2ms/step
12/12                 0s 2ms/step
28/28                 0s 2ms/step
12/12                 0s 2ms/step
```





```
[103]: def evaluate_prediction_volatility(pred1, pred2, true=None, name1="Model A",␣
        ↪name2="Model B"):
           """




           - pred1, pred2:        1D numpy array   list
           - true:
           """
```

```
    def metrics(arr):
        arr = np.array(arr)
        first_diff = np.diff(arr)
        std = np.std(first_diff)
        mad = np.mean(np.abs(first_diff))
        smooth = np.mean(np.abs(arr[2:] - 2 * arr[1:-1] + arr[:-2]))
        return std, mad, smooth

    std1, mad1, smooth1 = metrics(pred1)
    std2, mad2, smooth2 = metrics(pred2)

    print(f" {name1}    :")
    print(f"   ·      std(diff)         = {std1:.6f}")
    print(f"   ·       mean(abs) = {mad1:.6f}")
    print(f"   ·      smoothness         = {smooth1:.6f}")
    print()

    print(f" {name2}    :")
    print(f"   ·   std(diff)          = {std2:.6f}")
    print(f"   ·   mean(abs)          = {mad2:.6f}")
    print(f"   ·   smoothness         = {smooth2:.6f}")
    print()

    if true is not None:
        std_t, mad_t, smooth_t = metrics(true)
        print(f"     :")
        print(f"   · std(diff)         = {std_t:.6f}")
        print(f"   · mean(abs)         = {mad_t:.6f}")
        print(f"   · smoothness        = {smooth_t:.6f}")
        print()
```

[104]:
```
pred1 = np.concatenate([inv_train_preds_wo_sent, inv_test_preds_wo_sent]).
  ↪flatten()
pred2 = np.concatenate([inv_train_preds_with_sent, inv_test_preds_with_sent]).
  ↪flatten()

evaluate_prediction_volatility(
    pred1=pred1,
    pred2=pred2,
    true=true_adj_close_wo,
    name1="Transformer w/o sent",
    name2="Transformer w/ sent"
)
```

```
 Transformer w/o sent    :
   ·      std(diff)         = 1.677407
   ·       mean(abs) = 1.160032
   ·      smoothness         = 1.674059
 Transformer w/ sent    :
   ·    std(diff)         = 1.314947
   ·    mean(abs)         = 1.017639
   ·    smoothness        = 1.420668

       :
   · std(diff)         = 1.129569
   · mean(abs)         = 0.742417
   · smoothness        = 1.118966
```

[105]:
```
pred1 = np.concatenate([inv_train_preds_wo_sent, inv_test_preds_wo_sent]).
 ↪flatten()
pred2 = np.concatenate([inv_train_preds_with_sent, inv_test_preds_with_sent]).
 ↪flatten()

evaluate_prediction_volatility(
    pred1=pred1,
    pred2=pred2,
    true=true_adj_close_wo,
    name1="Transformer w/o sent",
    name2="Transformer w/ sent"
)
```

```
 Transformer w/o sent    :
  ·      std(diff)         = 1.677407
  ·        mean(abs) = 1.160032
  ·      smoothness         = 1.674059

 Transformer w/ sent    :
  ·    std(diff)         = 1.314947
  ·    mean(abs)         = 1.017639
  ·    smoothness        = 1.420668

      :
  · std(diff)         = 1.129569
  · mean(abs)         = 0.742417
  · smoothness        = 1.118966
```

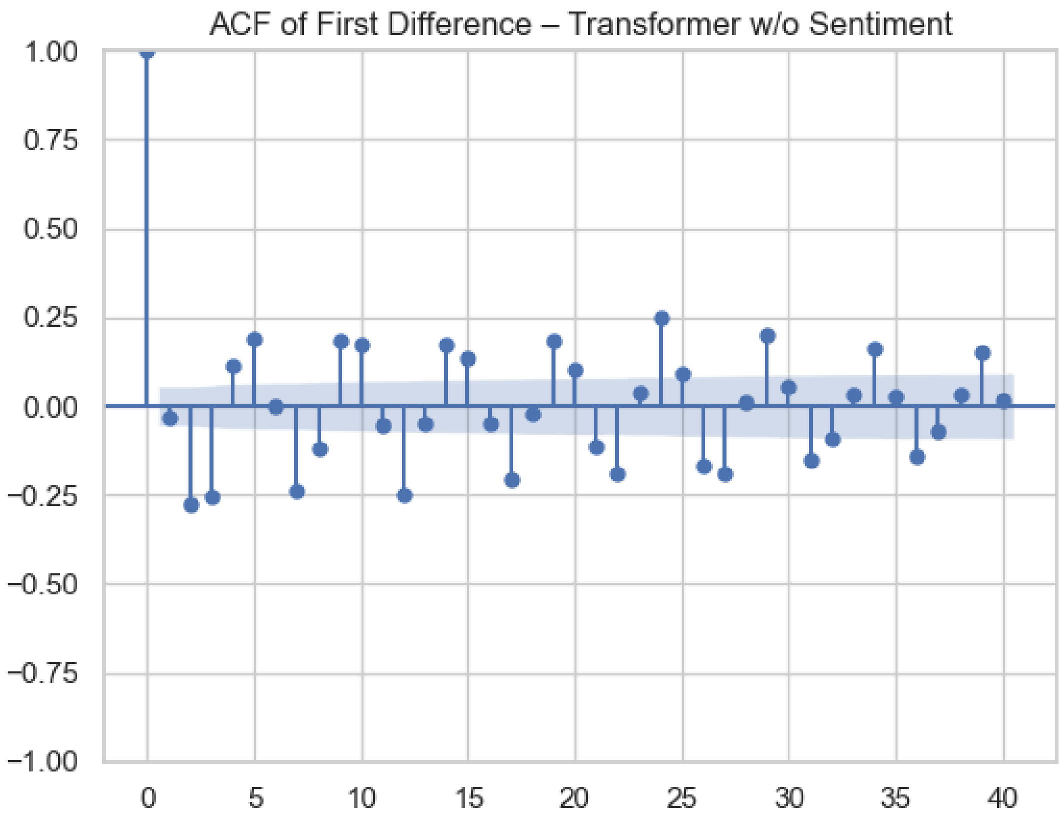
[106]:
```
pred_close_train_wo, _ = inv_preds(cnnBiLSTM_woSent, trainX_wo_tweet,␣
 ↪trainY_wo_tweet)
pred_close_test_wo,  _ = inv_preds(cnnBiLSTM_woSent, testX_wo_tweet,␣
 ↪testY_wo_tweet)

pred_close_train_w, _ = inv_preds(cnnBiLSTM_withSent, trainX_with_tweet,␣
 ↪trainY_with_tweet)
```

```
pred_close_test_w,  _ = inv_preds(cnnBiLSTM_withSent, testX_with_tweet,␣
 ↪testY_with_tweet)

pred3 = np.concatenate([pred_close_train_wo, pred_close_test_wo]).flatten()
pred4 = np.concatenate([pred_close_train_w, pred_close_test_w]).flatten()

evaluate_prediction_volatility(
    pred1=pred3,
    pred2=pred4,
    true=true_adj_close_wo,
    name1="cnnBiLSTM w/o sent",
    name2="cnnBiLSTM w/ sent"
)
```

```
 cnnBiLSTM w/o sent    :
  ·      std(diff)         = 0.689062
  ·        mean(abs) = 0.514961
  ·      smoothness         = 0.676101

 cnnBiLSTM w/ sent    :
  ·    std(diff)         = 1.031721
  ·    mean(abs)         = 0.802738
  ·    smoothness        = 1.240502

     :
  · std(diff)         = 1.129569
  · mean(abs)         = 0.742417
  · smoothness        = 1.118966
```

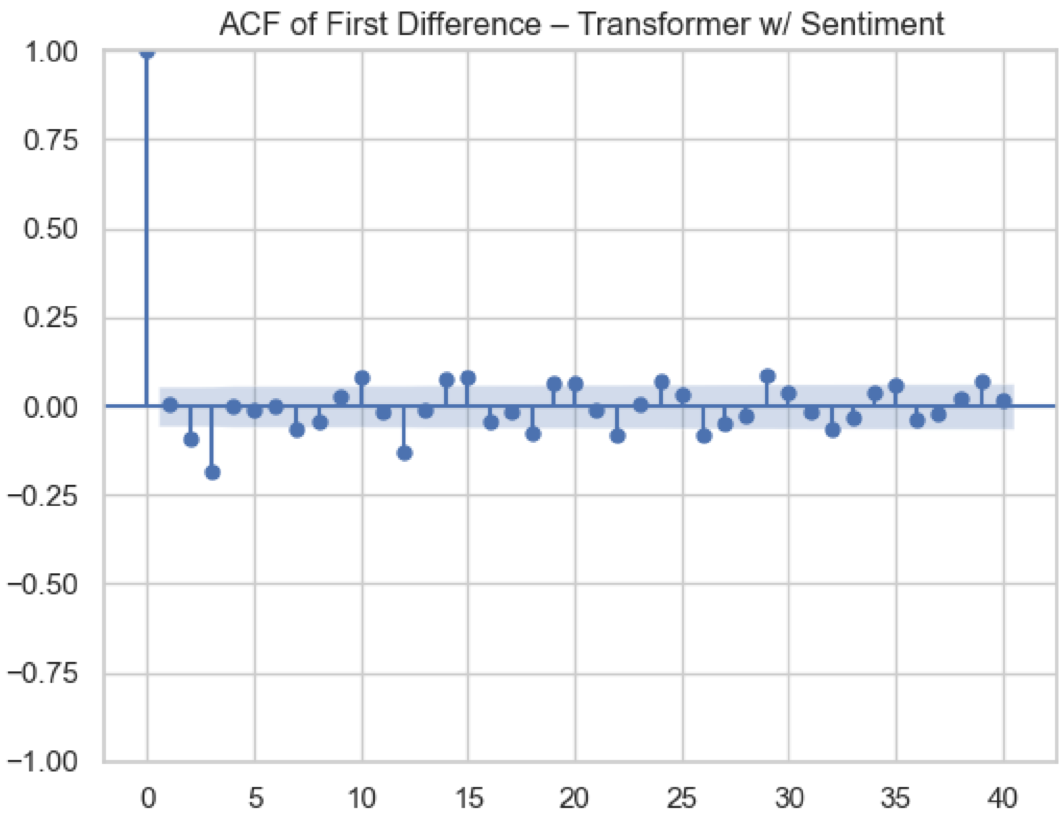
[107]:
```
from statsmodels.graphics.tsaplots import plot_acf
import matplotlib.pyplot as plt
import numpy as np

def plot_acf_diff(series, label):
    diff = np.diff(series)
    plt.figure(figsize=(10, 4))
    plot_acf(diff, lags=40, title=f"ACF of First Difference - {label}")
    plt.grid(True)
    plt.show()

plot_acf_diff(pred1, "Transformer w/o Sentiment")
plot_acf_diff(pred2, "Transformer w/ Sentiment")
plot_acf_diff(true_adj_close_wo, "True Closing Price")
```

```
<Figure size 1000x400 with 0 Axes>
```

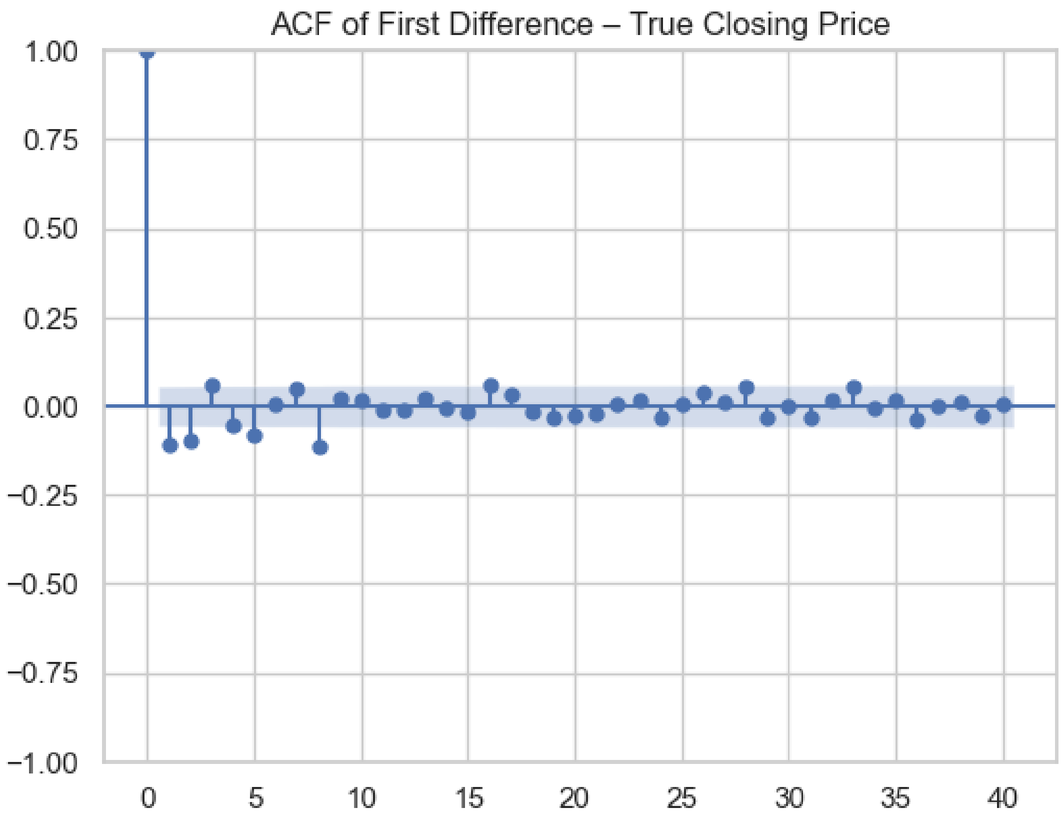


ACF of First Difference – Transformer w/o Sentiment
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30
35
40

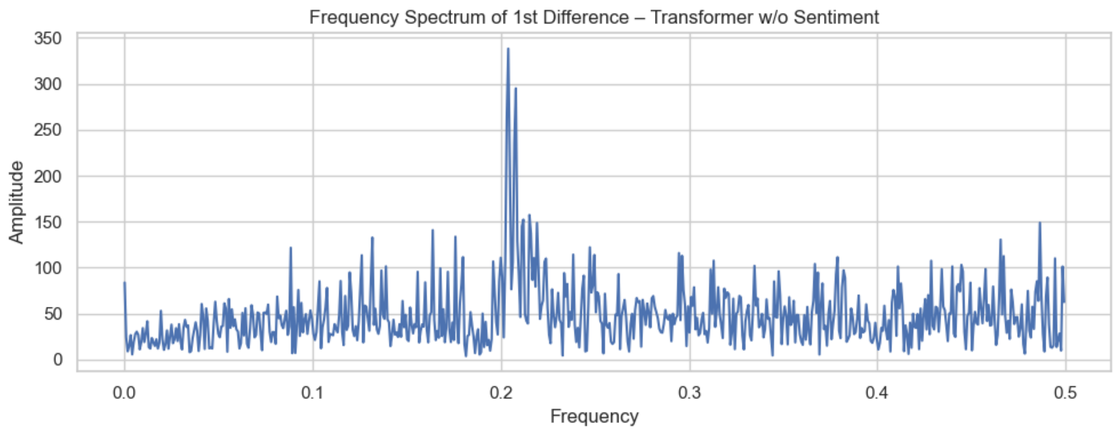
```
<Figure size 1000x400 with 0 Axes>
```

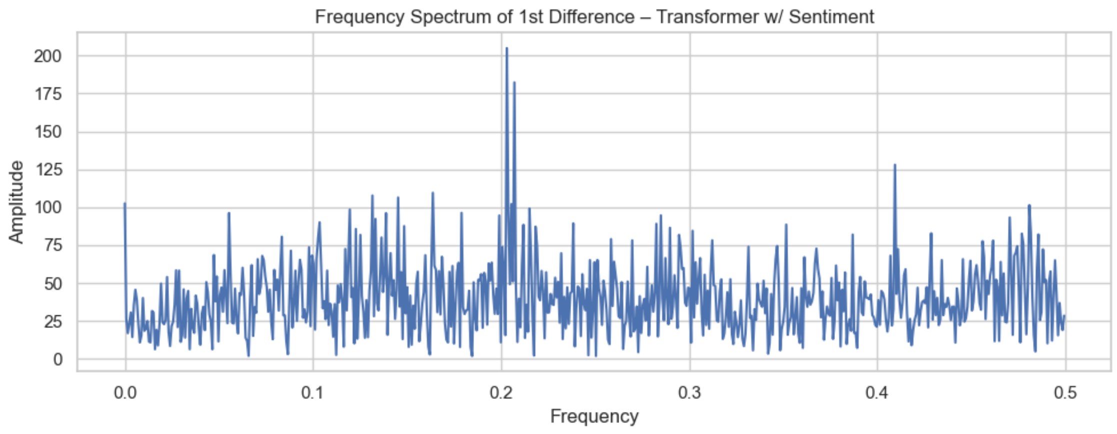

ACF of First Difference – Transformer w/ Sentiment
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30
35
40

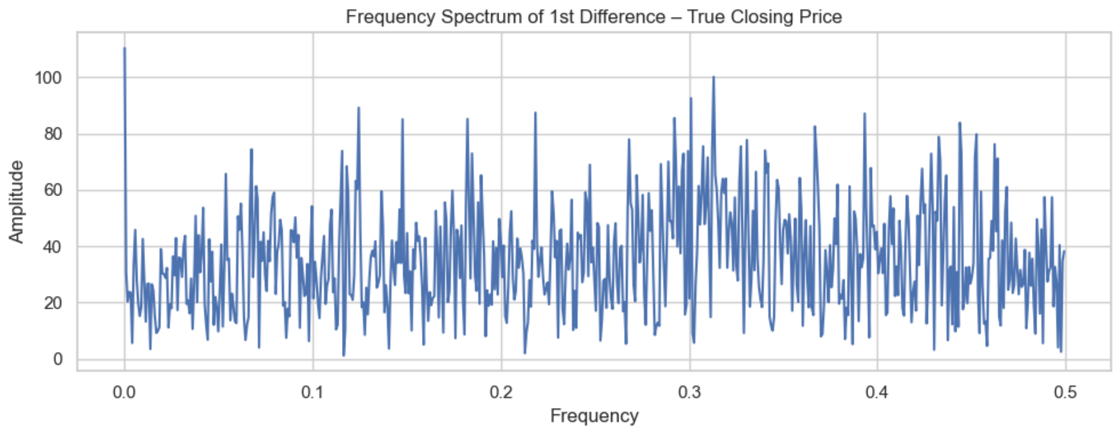
```
<Figure size 1000x400 with 0 Axes>
```



[108]:
```
from scipy.fft import fft, fftfreq
import matplotlib.pyplot as plt

def plot_frequency_spectrum(series, label, sample_rate=1):
    diff = np.diff(series)
    N = len(diff)
    yf = np.abs(fft(diff))
    xf = fftfreq(N, d=sample_rate)[:N // 2]

    plt.figure(figsize=(10, 4))
    plt.plot(xf, yf[:N // 2])
    plt.title(f"Frequency Spectrum of 1st Difference - {label}")
    plt.xlabel("Frequency")
    plt.ylabel("Amplitude")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

plot_frequency_spectrum(pred1, "Transformer w/o Sentiment")
plot_frequency_spectrum(pred2, "Transformer w/ Sentiment")
plot_frequency_spectrum(true_adj_close_wo, "True Closing Price")
```


Frequency Spectrum of 1st Difference – Transformer w/o Sentiment
Amplitude
Frequency


Frequency Spectrum of 1st Difference – Transformer w/ Sentiment
Amplitude
Frequency


Frequency Spectrum of 1st Difference – True Closing Price
Amplitude
Frequency