Forecasting Next-Day Market Prices of Stocks and Bitcoin Using Social Media Sentiment Analysis

Daniel Malvin Janesch

Dissertation written under the supervision of
Professor Pedro Afonso Fernandes

Dissertation submitted in partial fulfilment of requirements for the MSc in Business Analytics, at the Universidade Católica Portuguesa, 02.06.2025

Abstract

This thesis investigates whether sentiment scores derived from social media can improve the prediction of next-day market prices for financial assets, with a focus on Tesla ($TSLA) and Bitcoin ($BTC). Using freely available X (Twitter) datasets from Kaggle, three pre-trained Natural Language Processing (NLP) tools—VADER, TextBlob, and FinTwitBERT—were applied to individually score the sentiment of each post. After extensive cleaning, filtering, and sampling, a total of 47,800 tweets were used for analysis—33,200 for $TSLA and 14,600 for $BTC. Sentiment scores were aggregated into daily averages and merged with financial data from Bloomberg. These features were then used as explanatory variables in three forecasting models: ARIMA/ARIMAX, XGBoost, and Long Short-Term Memory (LSTM) neural networks. The results show that for $TSLA, the introduction of sentiment scores—especially with FinTwitBERT—substantially improved forecasting performance. The strongest model, an LSTM with sentiment and financial variables, achieved a test R^2 of 0.76 compared to a baseline model with R^2 of 0.49. In contrast, all models performed poorly on $BTC, likely due to its higher volatility, optimistically biased sentiment, and large data gaps (about 50% of tweet days missing due to structural issues). These findings suggest that sentiment analysis can enhance short-term price forecasting for traditional stocks, while its value for highly volatile assets like Bitcoin remains unclear. Additionally, the results highlight the superior predictive performance of transformer-based sentiment models like FinTwitBERT over lexicon-based alternatives. This study lays the groundwork for future research in real-time or intraday prediction using more complex models and diverse social media sources.

Keywords: sentiment analysis, financial forecasting, Bitcoin, Tesla, NLP, LSTM

Title: Forecasting Next-Day Market Prices of Stocks and Bitcoin Using Social Media Sentiment Analysis

Author: Daniel Malvin Janesch

Resumo

A presente dissertação encerra uma investigação sobre a possibilidade de o sentimento expresso nas redes soci-ais poder melhorar a previsão dos preços de mercado no dia seguinte de ativos financeiros, em particular, dos títulos da Tesla ($TSLA) e Bitcoin ($BTC). Partindo de dados do Twitter, disponíveis gratuitamente na plata-forma Kaggle, foram aplicadas três ferramentas de aprendizagem automática de Processamento de Linguagem Natural (NLP) – VADER, TextBlob e FinTwitBERT – para avaliar, individualmente, o sentimento expresso em cada publicação. Após um exaustivo processo de limpeza, filtragem e amostragem, foram analisados 47.800 tweets, 33.200 relativos à $TSLA e 14.600 à $BTC. Os scores de sentimento foram agregados em médias diá-rias e integrados com dados financeiros, também diários, provenientes da Bloomberg. Estas variáveis foram, por seu turno, integradas como fatores explicativos em três modelos de previsão: ARIMA/ARIMAX, XGBoost e redes neuronais LSTM (Long Short-Term Memory). Os resultados mostram que, no caso da $TSLA, a intro-dução da variável de sentimento – especialmente com FinTwitBERT – melhora substancialmente o desempe-nho da previsão. O modelo mais preciso, um LSTM com variáveis independentes de sentimento e financeiras, obteve um R^2 de teste de 0,76, comparando com o modelo de base, com um R^2 de apenas 0,49. No entanto, todos os modelos apresentaram fraco desempenho na previsão da cotação da $BTC, possivelmente devido à respetiva volatilidade, enviesamento otimista em termos de sentimento e lacunas nos dados (cerca de 50% dos dias com tweets não foram considerados devido a problemas estruturais). Estes resultados sugerem que a análi-se de sentimento de redes sociais pode melhorar a previsão das cotações de curto prazo de ações tradicionais, mas não tanto no caso de ativos muito voláteis como acontece com a Bitcoin. Adicionalmente, os resultados destacam o desempenho superior de modelos de sentimento baseados em transformers, como é o caso do FinTwitBERT, em comparação com modelos lexicográficos. Este estudo estabeleceu, ainda, uma base para in-vestigação futura no tópico da previsão em tempo real ou diária das cotações de ativos financeiros, utilizando modelos complexos de aprendizagem automática e fontes de dados diversificadas, incluindo as redes sociais.

Palavras-chave: análise de sentimento, previsão financeira, Bitcoin, Tesla, NLP, LSTM

Título: Previsão de Preços de Mercado no Dia Seguinte para Ações e Bitcoin Usando Análise de Sentimento de Redes Sociais

Autor: Daniel Malvin Janesch

Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Meaning
AAPL	Apple Inc.
ANN	Artificial Neural Network
AUC	Area Under the Curve
BERT	Bidirectional Encoder Representations from Transformers
BTC	Bitcoin
DJIA	Dow Jones Industrial Average
EMH	Efficient Market Hypothesis
GPOMS	Google Profile of Mood States
GRU	Gated Recurrent Unit
LSTM	Long Short Term Memory
MAE	Mean Absolute Error
MSFT	Microsoft Corporation
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
PACF	Partial Autocorrelation Function
R^2	Coefficient of Determination
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SEC	Securities and Exchange Commission
SOFFN	Self-Organizing Fuzzy Neural Network
SVM	Support Vector Machine
TSLA	Tesla Inc.
VADER	Valence Aware Dictionary and sEntiment Reasoner
XGBoost	Extreme Gradient Boosting

Acknowledgements

I would like to sincerely thank all the professors of my Master's program at Católica Lisbon for maintaining a high academic standard and consistently pushing us to deliver our best. In particular, I am grateful to Filipa Reis, Nicolò Bertani, and Miguel Godinho de Matos for their dedication and inspiring teaching.

I owe special thanks to my thesis supervisor, Professor Pedro Afonso Fernandes, for his feedback and continuous support throughout this project.

To my family, especially my mother and my sister — thank you for always having my back, even from far away. Your support has been incredibly important during this journey.

I would also like to thank my good friend Alexander Hebeis for providing access to Bloomberg market data, which, as I found out, is surprisingly difficult to obtain for free.

To all the amazing new friends I made during these two years: thank you! You turned this academic experience into the best two years of my life.

I am especially grateful for the opportunity to live and study in another country and even go on an academic exchange. These international experiences have shaped me both personally and professionally. Enrolling in this Master's program has been one of the best decisions I have ever made, and I am deeply thankful to Católica Lisbon for making this opportunity possible.

1 Introduction

According to the Efficient Market Hypothesis (EMH) derived from ideas by Eugene Fama of the University of Chicago in the 1960s, all relevant information about the value of a financial asset is rapidly spread among investors. These investors are constantly competing to find even the smallest differences between current prices and their own estimations of an asset's true value. If such differences are identified, they are quickly exploited and thus removed. This behavior results in efficient financial markets, where all assets are traded at a fair market value. New information is widely available for free and is released in the market randomly, making it almost impossible to consistently outperform the stock market(CFI-Team, 2021). This is in line with the Random Walk Theory, which was first proposed by the French mathematician and stock broker Julies Regnault in the 19th century and has since been adopted and reformulated by several researchers.

The Random Walk Theory states that if stock markets are truly efficient, then stock prices follow a random stochastic process and are therefore not autocorrelated, challenging the idea that patterns or trends identified through technical analysis can give individual investors a competitive edge in the form of higher returns without taking on more risk(CFI-Team, 2020). The Efficient Market Hypothesis remains an important aspect of financial literature, and accomplished economists like Malkiel (2003) argue that capital markets are still largely efficient. However, modern research focuses on insights from fields such as psychology, machine learning, and big data analytics, questioning the status quo by arguing that stock prices can be predicted to a certain extent in the very short term.

One particularly interesting area of research examines how alternative data sources, such as social media activity, could provide valuable signals about investor sentiment and market movements. In the last two decades, social networks have become more than just a creative outlet or a means of sharing and consuming content. Their widespread use and availability have turned social networks into massive data centers that almost function as a mirror of today's society, or at least public mood. Since financial markets are also a reflection of the public's mood, this leads to the assumption that the information derived from these data giants could be used to make predictions about future market movements, if analyzed correctly.

There have been several instances in recent history where we have seen social media activity directly impact financial markets, both from influential personas as well as ordinary users. In 2018, billionaire Elon Musk posted on X (formerly Twitter) that he was considering taking his company Tesla private at $420 per share, briefly increasing the stock price by 6%. A few days later, when it became clear that there was no such deal in sight, the stock price plummeted. This action led to an SEC fraud investigation that ultimately cost him and his company $40 million (Price and Schroeder, 2018).

In 2021, investors on the Reddit forum r/WallStreetBets worked together to trigger a short squeeze of the GameStop stock, driving it up more than 2,800% in only a few weeks. This resulted in billions of dollars in losses for large hedge funds and led to a series of congressional hearings about market manipulation, while also raising public awareness about the power of social media over financial markets (Morrow, 2021).

In the cryptocurrency (and especially in the meme coin) community, these fluctuations are a well-known phenomenon, as the crypto market is famously sensitive to public sentiment. Many investors constantly monitor X or Reddit feeds in order to be the first to react to new trends. Tweets from figures such as Elon Musk, Donald Trump, and other celebrities have a history of causing significant spikes and drops in the prices of Bitcoin and other cryptocurrencies. In early 2021 for example, the price of Bitcoin jumped by 20% after Tesla announced that it would purchase $1.5 billion in Bitcoin. Later that year, tweets reporting that the Chinese government was banning crypto trading and mining caused Bitcoin's price to drop by 30% (John et al., 2021).

I have been personally following the news about events like these for a couple of years, and while most of these examples were triggered by public figures or major events, they have sparked my interest in the impact of social media on financial markets. What if more subtle shifts in public mood could be measured by analyzing hundreds of thousands of X posts about a particular company or about Bitcoin using Natural Language Processing? Could these insights be used in combination with machine learning algorithms to predict market movements shortly before they occur? While the cryptocurrency market is known to be highly sentiment-driven, it might be interesting to test whether the stock price of a blue-chip company like Tesla is less affected by public sentiment.

This interest led to my research question: Can sentiment scores derived from X using NLP tools like VADER and TextBlob and FinTwitBert be used to improve the prediction of next-day market prices, and how does their predictive power differ when applied to blue-chip stocks (like Tesla) versus Bitcoin?

2 Literature Review

2.1 Context

The idea of using online discussions as a way to capture investor sentiment has been explored for more than 25 years. One of the first studies in this area by Tumarkin and Whitelaw (2001) examined whether investor discussions on financial message boards contained valuable predictive information or mostly represented "noise". Their findings indicated that, while extreme sentiment in online discussions was sometimes associated with stock price movements, these effects are often reversed over time, suggesting that the market might over-react to extreme investor sentiment. This implies that Internet sentiment might influence stock prices in the short term, potentially providing valuable insights before corrections occur. Similar results were found by researchers Antweiler and Frank (2004, 2006), who examined more than 180,000 corporate news stories from the Wall Street Journal spanning nearly three decades. Their study revealed that stock prices tend to follow an initial reaction followed by a reversal, suggesting that the market overreacts to news at first and then gradually corrects itself. This early work laid the foundation for later studies focusing on Natural Language Processing and Machine Learning techniques to systematically analyze investor sentiment on the Internet and improve traditional stock price prediction methods.

Building on these insights, the ever-growing popularity of social networks, as well as the rapid advances in Machine Learning and AI technology have created the opportunity for a new field of interest in financial research, essentially offering an intersection of behavioral finance and alternative data in finance. Researchers in this relatively young field have explored the use of NLP and sentiment analysis to measure public mood, by processing large datasets of social media posts. In combination with advanced Machine Learning algorithms, this data is used to gain insights that aim to challenge traditional views of market efficiency that are aligned with the Efficient Market Hypothesis. Early work in this area suggested that public mood, reflected in social media, could provide predictive insights into financial markets(Bollen et al., 2011). More recent studies have extended this approach to the cryptocurrency market, particularly Bitcoin, exploring whether social media sentiment can similarly predict market movements(Valencia et al., 2019). The following sections review key contributions in sentiment analysis, financial prediction models, and research exploring both traditional stock markets and the cryptocurrency market.

2.2 Natural Language Processing for Stock Market Predictions

One of the most influential studies in this field is by Bollen et al. (2011), who analyzed whether public mood, as reflected in social networks, could improve predictions of stock market movements. Their study introduced the idea that large-scale sentiment data from X could serve as an information advantage, challenging the idea of market efficiency. The authors applied NLP techniques on a dataset of 9.8 million posts from 2008, using a combination of the mood tracking tools OpinionFinder and Google Profile of Mood States (GPOMS) to classify sentiment into six emotional states: calm, alert, sure, vital, kind, and happy. They built two predictive models: one baseline model that relied solely on historical stock market data and a second model that included sentiment information. Using Granger causality tests and Self-Organizing Fuzzy Neural Networks (SOFNN), the study examined whether fluctuations in public sentiment were correlated with movements in the Dow Jones Industrial Average (DJIA). Their findings suggested that certain emotional states (calmness in particular) were significantly correlated with stock market performance. Their sentiment model achieved a prediction accuracy of 87.6% for daily DJIA movements, outperforming the baseline model.

The authors made significant progress in connecting public mood to market movements. However, while their findings challenged the EMH, their study is limited in several aspects. First, they focused exclusively on the DJIA rather than individual stocks, making it unclear whether sentiment influences specific assets differently and leaving room for further research in this area. Second, their sentiment classification relied on custom-built mood-tracking tools (OpinionFinder and GPOMS), which are based on a predefined lexicon. More recent studies utilize pre-trained NLP models like VADER, TextBlob, or FinBERT, which may provide a more robust sentiment analysis. Third, their approach to filtering posts is not ideal. They only analyzed tweets that explicitly contained phrases such as 'I feel' or 'I am feeling,' thereby excluding a vast number of posts that express sentiment in more subtle ways. They also removed all tweets containing links, potentially omitting valuable market-relevant discussions and news-driven sentiment.

Another study of interest was conducted by Pagolu et al. (2016), who applied NLP techniques to predict changes in Microsoft's stock price. They collected a dataset of 250,000 X posts about Microsoft and implemented a supervised learning method, training a sentiment classifier on 3,216 human-labeled tweets. To represent text data, they experimented with two different techniques: N-grams, which capture short sequences of words without considering deeper meaning, and the more advanced Word2Vec, which converts words into numerical vectors that reflect their meaning based on surrounding words. Once trained, their model achieved a sentiment classification accuracy of approximately 70%.

The authors used their sentiment classifier to label tweets as negative, neutral, or positive. To incorporate sentiment into their predictive model, they aggregated sentiment scores into three-day averages, which were then used as input for a logistic regression model and a Support Vector Machine (SVM) to predict stock price movements as up or down. Their best-performing model, based on SVM, achieved a prediction accuracy of approximately 72%.

Pagolu et al. (2016) demonstrated a promising link between X sentiment and stock market movements, but their approach has room for improvement. They treated stock price movements as independent data points rather than modeling them as time series, despite the frequent assumption in research that financial market data are autocorrelated. Their sentiment classifier was trained on a relatively small dataset of just 3,216 manually labeled tweets, which limits generalizability and would require the model to be retrained for any other asset. Finally, while their best model outperformed the 50% threshold suggested by the EMH, their results remain relatively unimpressive, suggesting that sentiment alone may not be sufficient for significant and accurate market predictions.

In a newer study, Kolasani and Assaf (2020) employed a more sophisticated deep learning approach to predict market movements of both the DJIA and Apple Inc. (AAPL). The authors first trained sentiment classification models using the "Sentiment140" dataset from Kaggle, which contains 1.6 million pre-labeled X posts categorized as positive (1) or negative (0). They evaluated five machine learning algorithms for this classification task: Logistic Regression, SVM, Decision Tree, Boosted Tree, and Random Forests. The SVM model proved to be the most effective and achieved approximately 83% accuracy in sentiment classification.

To capture only relevant posts, the researchers filtered X data from January to December 2016 using three distinct keywords: "stock market" (for general market sentiment), "StockTwits" (targeting posts about the largest social media platform for investors), and "AAPL" (for company-specific posts about Apple). The researchers implemented several text preprocessing techniques on these posts, including lemmatization (reducing words to their base forms), and removing short and stop words (which typically carry less meaning). After classifying the sentiment of these processed posts, they aggregated the results into daily average sentiment scores, aligning them with days the stock market was open for trade and leaving out the days it was closed.

The daily sentiment averages were combined with historical price data from Yahoo Finance to create a comprehensive predictive model. The researchers implemented both Boosted Regression Trees and Multilayer Perceptron Neural Networks to forecast stock price movements. Their comparative analysis revealed that neural networks consistently outperformed traditional methods across all three keyword datasets, with company-specific posts about Apple yielding the most accurate predictions.

Notably, their methodology demonstrated that incorporating social media sentiment significantly improved the predictive power of stock market forecasting models compared to using historical price data alone.

Kolasani and Assaf (2020) present a novel approach by integrating sentiment analysis of X feeds with historical stock price data into a deep-learning model to forecast market movements. While their model does use historic prices as input, they are not modeled as a time series, potentially missing the autocorrelation that is often assumed in research. Additionally, their sentiment classifier is trained on a fixed dataset that only labels posts as positive or negative, which may not capture the nuanced variations in sentiment as effectively as more recent pre-trained models such as VADER, TextBlob, or FinBERT. Lastly, regarding DJIA prediction, filtering only based on two specific keywords ("stock market" and "StockTwits") could exclude valuable input from a large number of valuable social media discussions without these two keywords.

One of the most recent and state-of-the-art studies in this field is by Koukaras et al. (2022) from the International Hellenic University. The researchers collected X and StockTwits data related to Microsoft ($MSFT) from July 2020 to October 2020 and applied a series of preprocessing techniques to the text data. They then performed sentiment analysis using the advanced tools VADER (Valence Aware Dictionary and sEntiment Reasoner) and TextBlob, assigning each post a sentiment score between -1 and 1. The daily averages of the scores were standardized, outliers were removed to reduce bias, and missing values were interpolated. For predictive modeling, the authors combined daily sentiment scores with several stock-related variables from Yahoo Finance, testing seven different classification algorithms, including Logistic Regression, Support Vector Machines (SVM), Decision Trees, and a Neural Network, to classify stock movements as up or down. Their best-performing model, a combination of VADER and SVM, yielded an F-score of 76.3% with an AUC of 67%.

Unlike earlier studies that relied on simpler prediction techniques or lexicon-based sentiment alone, this research compares multiple classification algorithms to optimize predictive accuracy and uses some of the most modern applications available for sentiment analysis. However, it is worth noting that the stock price changes were again modeled as independent data points, potentially missing out on their time-series nature. The authors themselves noted several limitations, including sentiment analysis tools that sometimes misinterpret sarcasm, the presence of spam accounts that potentially affect sentiment scores, and class imbalance in the dataset (most instances are stock price increases).

2.3 Natural Language Processing for Cryptocurrency Market Predictions

While sentiment analysis has shown promise in stock market prediction, its role in cryptocurrency markets may be even more significant. Traditional stocks are often influenced by financial metrics of their companies, macroeconomic factors, and the activity of institutional investors. Cryptocurrencies like Bitcoin, on the other hand, are highly sentiment-driven and volatile, with prices often reacting strongly to social media trends, public discourse, and public sentiment(Bakas et al., 2022). Additionally, the crypto market operates 24/7, meaning that

sentiment shifts can trigger price movements at any time, without the constraints of traditional trading hours. Despite these unique characteristics, there has been comparatively less research on using sentiment analysis for Bitcoin price prediction, making it an exciting area of study.

As early as 2015, well before Bitcoin prices soared to record-breaking highs in subsequent years, researchers identified X sentiment as one of the factors that is positively correlated with Bitcoin price in the short term (Georgoula et al., 2015).

Early attempts to use this insight to predict prices followed a couple of years later, when Pant et al. (2018) applied two feature extraction techniques (Bag-of-Words and Word2Vec) to a dataset of more than 4,000 manually labeled X posts. These techniques transform raw text into numerical representations that machine learning models can understand. Using the extracted features, the authors trained multiple classification algorithms and achieved an accuracy of approximately 81% for sentiment classification. They then combined sentiment scores with historical Bitcoin prices as input for a Recurrent Neural Network (RNN), which showed a price prediction accuracy of around 78%, presenting further evidence that Bitcoin prices can indeed be predicted using sentiment data. Although these results are respectable, the dataset was relatively small, and the manual labeling process may have introduced bias, potentially affecting the accuracy of the classification.

One of the most promising recent approaches to cryptocurrency price prediction was presented by Asemi et al. (2024), who combined both price data and sentiment features to improve their deep learning models. Their methodology approach consisted of first collecting historical Bitcoin price data and X posts related to cryptocurrency; second, applying pre-trained models TweetNLP and BERTweet (both trained on Google's BERT LLM model) to extract sentiment scores that were then aggregated into 1-hour bins; and finally, integrating both price data and sentiment features into various recurrent neural network architectures.

Their best performing model (a GRU model) outperformed other architectures and achieved an impressive 90.3% accuracy when incorporating sentiment analysis with a 16-hour lag. This was approximately 3% higher than the models using price data alone, which revealed an intriguing insight: X sentiment seems to become increasingly influential for price predictions beyond a 12-hour window, with optimal prediction accuracy occurring 16-17 hours after the expression of the sentiment on social media. These findings suggest that the cryptocurrency market requires time to absorb and react to sentiment signals, resulting in the delayed impact of Twitter sentiment on price movements.

My research will aim to address the limitations of previous studies on both traditional stocks and Bitcoin by applying several pre-trained NLP models to a large set of posts after targeted filtering. Each tweet will be individually evaluated and assigned a sentiment score using these models. The resulting sentiment scores will then be aggregated into daily averages and used as input features for a range of prediction models. This approach is intended to offer a more accurate and scalable method of sentiment analysis across a diverse set of financial assets.

3 Methodology

This chapter outlines the methodological framework used to investigate whether sentiment derived from social media posts can enhance the prediction of financial market movements. The process consists of several key stages:

- Collection of large sets of X data from Kaggle.
- Collection of financial data from Bloomberg.
- Pre-Processing steps (data cleaning, filtering, and sampling).
- Sentiment analysis using natural language processing tools.
- Prediction model training and tuning.
- Evaluation and Interpretation of Results.

Figure 3.1 provides an overview of the entire pipeline, from raw data acquisition to the final evaluation of predictive models.

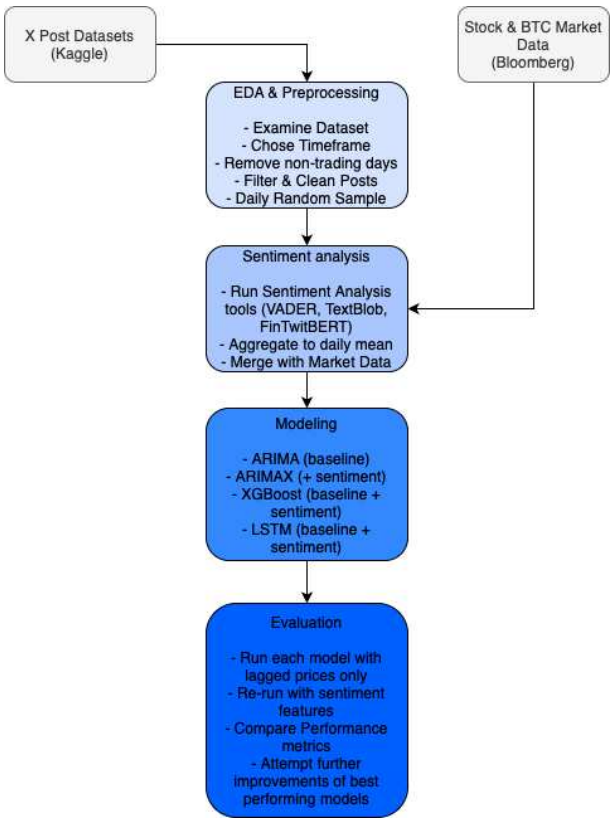

Figure 3.1: Overview of the modeling pipeline.

3.1 Natural Language Processing Background

Natural Language Processing (NLP) refers to a set of techniques used to extract structure and meaning from unstructured text data. As outlined by Taddy (2019), standard NLP pipelines typically include several preprocessing steps:

- Tokenization: splitting text into individual words or subword units so that it can be processed further. Without tokenization, models cannot identify individual terms or phrases.
- Stopword removal: eliminating common words (e.g., "the", "and", "is") that carry little semantic value
- Stemming or lemmatization: reducing words to their base form (e.g., "running" → "run").
- Feature vectorization: converting text into numerical representations that machine learning models can process.

All three tools used in this study (VADER, TextBlob, and FinTwitBERT) handle tokenization automatically as part of their internal pipelines. While VADER and TextBlob use traditional word-level tokenization, FinTwitBERT breaks text into smaller word fragments (subwords), allowing it to better handle rare or misspelled words and to interpret meaning based on context. Unlike the other tools, FinTwitBERT is a machine learning model and therefore requires feature vectorization. It handles this internally through a method called contextual embedding, which transforms words into complex numerical vectors based on the surrounding text. For example, the word "*bank*" should be represented differently in "*river bank*" compared to "*investment bank*", because the model interprets meaning in context.

Additionally, lowercasing was intentionally avoided because both VADER and FinTwitBERT use capitalization as a signal of sentiment intensity (e.g., "GREAT" vs. "great"). Emojis were also kept, since they often carry emotional value. VADER explicitly includes emojis and emoticons in its sentiment lexicon, and FinTwitBERT can interpret them in context as it was trained on finance-related tweets.

3.2 Data Collection

3.2.1 X (Twitter) Data

Two separate datasets of historical tweets were obtained from Kaggle: one focused on blue-chip stocks such as **Apple ($AAPL)**, **Tesla ($TSLA)**, and **Microsoft ($MSFT)**, and the other on **Bitcoin ($BTC)**. This section describes the pre-processing steps applied to the stock dataset; similar procedures were later applied to the Bitcoin dataset with minor adjustments where necessary. The raw stock dataset contained over **3.3 million entries**, consisting of tweet text, timestamps (in Unix format), and various fields with metadata. The data was first filtered to retain only tweets concerning $TSLA, and cleaned by removing unnecessary columns such as comment counts, retweets, likes, and user identifiers. Timestamps were converted to New York local time to better align with U.S. market trading hours. To ensure sufficient training data, the period from **January 1 to October 2, 2018** was selected based on tweet volume and relative market stability, in an attempt to identify a time frame within the available data that was not affected by major economic crises (like COVID-19) that could skew the results.

Tweets were filtered to:

- Include only posts containing the $TSLA cashtag.
- Exclude weekends and U.S. market holidays, using the official U.S. federal holiday calendar.
- Remove duplicate entries and retweets.
- Filter out non-English content (using the FastText language identifier).
- Filter out very short posts (under 25 characters), and posts containing promotional, spam-related keywords or pure price updates (e.g., “giveaway”, “free”, “current Tesla price”).
- Eliminate tweets with more than three hashtags or more than two cashtags, those containing phone numbers or emails, and tweets consisting of more than 30% numeric characters.

The tweet text was further cleaned by removing mentions, URLs, and excessive whitespace. A time series plot of daily tweet counts was generated to examine tweet volume over time. After the removal of weekends and market holidays, 190 trading days remained. To ensure data quality and model robustness, days with fewer than 200 posts were excluded from the final dataset, resulting in 166 trading days suitable for sentiment analysis. To ensure balance across days, 200 posts per day were randomly sampled from the remaining filtered data.

The Bitcoin dataset contained approximately **3.6 million entries** covering the period from 2021 to 2023. However, structural issues such as mismatched or corrupt entries resulted in many days appearing as missing during aggregation. In an effort to maximize usable data while minimizing gaps, a 150-day window from **February 14 to July 13, 2022** was selected based on the lowest number of missing tweet days. Since Bitcoin is traded continuously, no filtering for weekends or holidays was required.

Due to a higher volume of spam and bot activity, stricter filtering criteria were applied compared to the stock dataset, including an extended list of cryptocurrency-specific spam keywords. After applying the same pre-processing pipeline as for the stock dataset and removing days with fewer than 200 tweets, only 73 days within the 150-day window remained suitable for sentiment analysis. Again, a random sample of 200 posts per day was used for sentiment analysis. In total, **47,800 posts** (33,200 for $TSLA and 14,600 for $BTC) were analyzed across both datasets.

3.2.2 $BTC & $TSLA Market Price Data

Daily market data for both Tesla ($TSLA) and Bitcoin ($BTC) was obtained from Bloomberg, including open, high, low, and close prices for each asset. The dataset was cleaned by standardizing decimal formatting and converting date fields to a consistent datetime format.

For $TSLA, the analysis focused on the period from **January 1 to October 2, 2018**, and for $BTC on the period from **February 14 to July 13, 2022**, aligning with the corresponding X datasets.

Since Bitcoin is traded continuously and does not have an official closing time, the price of $BTC at the close of the New York Stock Exchange (NYSE) was used to maintain comparability with $TSLA's daily closing price. Since the data was sourced from Bloomberg, all timestamps were assumed to already reflect New York local time.

As part of exploratory data analysis, graphs were created to visualize the relationship between sentiment scores and asset prices (the sentiment analysis process is detailed in Section 3.2). These visual inspections were performed out of curiosity and served to inform the modeling approach later on.

The final market datasets were subsequently merged with the aggregated daily sentiment scores to support both time series and machine learning-based prediction models.

3.3 Sentiment Analysis

This section describes the methods used to extract sentiment from social media posts related to **$TSLA** and **$BTC**. Three different sentiment analysis tools were used: **VADER**, **TextBlob**, and **FinTwitBERT**. These tools were selected to provide a mix of lexicon-based and transformer-based approaches, allowing for comparison between different sentiment extraction techniques.

The output of each tool was aggregated at the daily level and later merged with the corresponding market data to serve as input features for the forecasting models. In addition, exploratory plots of daily sentiment trends and score distributions were created to visually examine the behavior of the sentiment over time. The following subsections outline the characteristics and implementation of each sentiment tool.

Figures 3.2 and 3.3 present the distributions of the sentiment scores produced by the three tools across the two datasets ($TSLA and $BTC). A more detailed discussion of the output of each tool follows in the respective subsections.

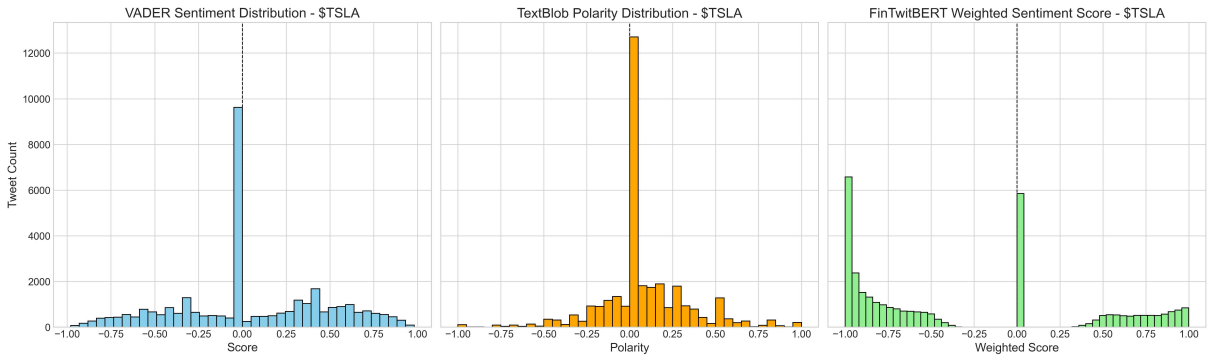

Figure 3.2: Distributions of sentiment scores across VADER, TextBlob, and FinTwitBERT - $TSLA.

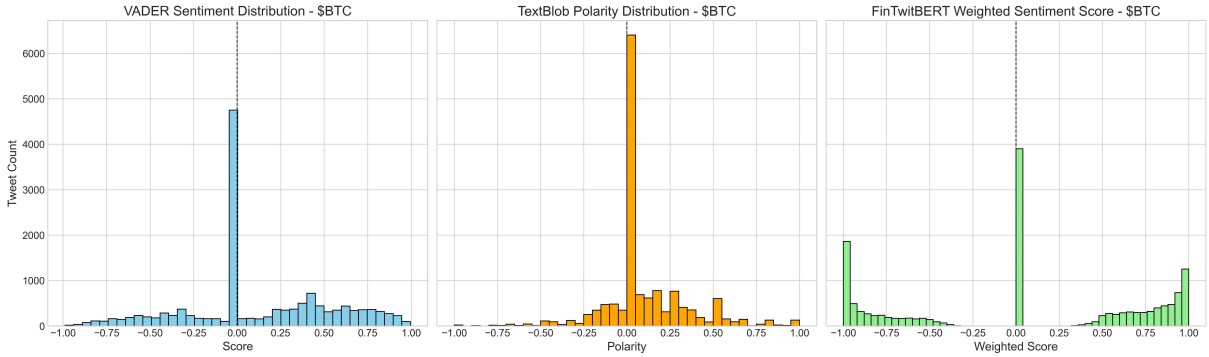

Figure 3.3: Distributions of sentiment scores across VADER, TextBlob, and FinTwitBERT - $BTC

3.3.1 VADER

VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon- and rule-based sentiment analysis tool specifically developed for analyzing sentiment in social media text introduced by Hutto and Gilbert (2014). It combines a manually created sentiment lexicon with five heuristic rules that account for syntactic and grammatical cues such as punctuation (e.g., "!"), capitalization (e.g., "GREAT"), degree modifiers (e.g., "extremely"), negation, and so called contrastive conjunctions (e.g., "but"). It can also handle Emjoiis. These features allow VADER to better capture sentiment intensity in informal, short-form text like tweets.

VADER calculates sentiment scores based on a predefined dictionary of words, each assigned a score ranging from -4 (most negative) to +4 (most positive). The tool then applies this data to adjust the sentiment of each sentence accordingly. The final output includes four scores: positive, neutral, negative, and compound. The compound score, which is a normalized, weighted aggregate ranging from -1 to +1, is the most commonly used metric for overall sentiment polarity. In this study, VADER was implemented using the *SentimentIntensityAnalyzer* class from the *vaderSentiment* Python package.

As shown in Figures 3.2 and 3.3, the distributions of VADER sentiment scores for both the $TSLA and $BTC datasets are quite similar. In both cases, a sharp spike occurs at a compound score of 0.0, indicating that a large number of tweets were classified as neutral. The overall distribution is centered around zero, with relatively few tweets receiving highly positive or highly negative scores. This pattern suggests that VADER often gives a neutral classification when sentiment is not clearly expressed using words from its predefined lexicon. In practice, this can lead to the wrong classification of tweets that are emotionally or contextually meaningful but do not contain certain keywords. A manual review of a sample of "neutral" tweets confirmed this limitation.

> For example, the following tweet about Tesla was assigned a neutral score:
>
> *$TSLA factory could burn down, orders could fall off a cliff, all c-level execs could walk, and their stock price would still probably only take a point or two hit as long as #elonmusk is still in charge!*
>
> Similarly, a Bitcoin-related tweet was also classified as neutral despite containing emotionally charged language:
>
> *f*ck these banks! we have the power to overthrow them! use #bitcoin !!!*

While human readers would likely interpret these posts as highly positive toward Bitcoin and Tesla, VADER fails to capture the sentiment correctly. This illustrates its limitations in detecting sarcasm, exaggeration, and emotionally loaded language, which are common in financial social media.

3.3.2 TextBlob

TextBlob is a lexicon-based sentiment analysis tool built on top of the Python natural language toolkit (*nltk*) and *pattern* libraries. While *nltk* provides natural language processing tools such as tokenization and part-of-speech tagging (categorizing words into grammatical categories), *pattern* supplies the sentiment lexicon used to calculate polarity and subjectivity scores. It uses a predefined sentiment lexicon and a rule-based polarity scoring system to assess the sentiment of a given text. Each tweet is assigned two key values: polarity, ranging from –1 (most negative) to +1 (most positive), and subjectivity, ranging from 0 (objective) to 1 (subjective) (Loria, 2025). In this study, only the polarity score was used.

In contrast to VADER, TextBlob applies a simpler averaging of word-level polarity scores based on a general-purpose lexicon. It was included in this study as an additional baseline for VADER, allowing comparison between two different lexicon-based approaches. Despite their limitations, these tools are computationally highly efficient, especially on very large datasets, and widely used in sentiment research, especially when labeled training data is unavailable.

As shown in Figures 3.2 and 3.3, the polarity scores generated by TextBlob are heavily concentrated around zero, suggesting that many tweets were classified as neutral or weakly polarized. Similarly to VADER, a manual review of sample tweets revealed that this often reflects a lack of contextual understanding, rather than true neutrality. This illustrates a key limitation of lexicon-based tools like TextBlob and VADER: their reliance on explicit sentiment words, rather than the underlying context. As discussed in the next section, FinTwitBERT attempts to address this gap by applying transformer-based deep learning to capture sentiment more accurately.

3.3.3 FinTwitBERT

FinTwitBERT is a transformer-based sentiment analysis tool designed specifically for financial social media text developed by Akkerman and Koornstra (2023). It builds on BERT (Bidirectional Encoder Representations from Transformers), a deep learning architecture introduced by Google. Unlike previous models, which process one word one at a time, transformer-based models are able to analyze relationships between all words in a sentence simultaneously. They can interpret both short- and long-range dependencies in context which makes them particularly effective for detecting sentiment nuances in informal text. FinBERT, introduced by Yang et al. (2020), uses the BERT architecture for the financial domain by pre-training on formal financial documents such as SEC filings and analyst reports. While this is effective for clearly structured and professional text, FinBERT may not perform as well on the informal, emotional, or sarcastic language common on financial Twitter. To address this, FinTwitBERT was developed by further fine-tuning FinBERT on a large set of finance-related tweets, enabling it to better understand sentiment in noisy and chaotic social media environments.

In this study, FinTwitBERT was implemented using the *StephanAkkerman/FinTwitBERT* sentiment model with the *HuggingFace transformers* pipeline. Each tweet was classified as BULLISH, BEARISH, or NEUTRAL, along with a confidence score for the predicted label. To compute a standardized sentiment score, labels were first mapped to numeric values (1 for BULLISH, 0 for NEUTRAL, –1 for BEARISH), and then multiplied by the model's confidence for each classification:

$$\textit{Weighted Score} = \textit{Numeric Label} \times \textit{Confidence}$$

This produces a continuous score between –1 and +1, similar to the output of VADER and Textblob. While FinTwitBERT is much more computationally demanding than VADER and TextBlob, it offers superior performance in detecting sentiment that depends on sarcasm, context, or phrasing nuances. As shown in Figures 3.2 and 3.3, the FinTwitBERT sentiment scores show a more polarized distribution, generally more bearish for $TSLA and more bullish for $BTC with fewer neutral scores. This shows its ability to extract deeper semantic information and suggests its potential as a meaningful input for financial prediction models.

In contrast to the earlier misclassifications by VADER and TextBlob, FinTwitBERT also demonstrated the ability to correctly identify genuinely neutral tweets, particularly those that were informative or curious in tone, rather than emotionally expressive.

> For example, the following tweet was classified as neutral by FinTwitBERT:
> *I wonder how many people have accumulated 1 full bitcoin? #btc yes no* 👇👇

> Similarly, this tweet about Tesla was also labeled as neutral:
> *tesla $tsla scheduled to post quarterly earnings on wednesday*

Neither of the tweets expresses a clear negative or positive sentiment, which aligns well with a neutral classification.

To visually assess the relationship between sentiment and market behavior, FinTwitBERT sentiment scores were overlaid with daily closing prices for both $TSLA and $BTC (see Figures 3.4 and 3.5). For $TSLA, the sentiment shows some alignment with price movements, with bearish sentiment preceding or coinciding with sharp price drops, particularly in April and September 2018. This suggests that FinTwitBERT may be capable of capturing sentiment shifts that correlate with market movements.

In contrast, $BTC sentiment remains more erratic and frequently optimistic, despite the consistent downward trend in price during the observed period in 2022. This observation comes as a bit of a surprise, as I had initially assumed that $BTC and other cryptocurrencies would mirror sentiment more closely than traditional stocks. However, it seems that, at least during this period of observation, public sentiment often remained bullish even in periods of negative price movement.

It should be noted that the $BTC sentiment dataset contained a significant number of missing days (around 50% of the total window) due to corrupted or incomplete tweet data. This causes the irregular structure and limits the interpretability of the $BTC sentiment overlay.

Although these visual comparisons are not sufficient to draw conclusions, they highlight the potential value of sentiment as an external variable, particularly in assets where price changes appear to correlate with sentiment shifts.

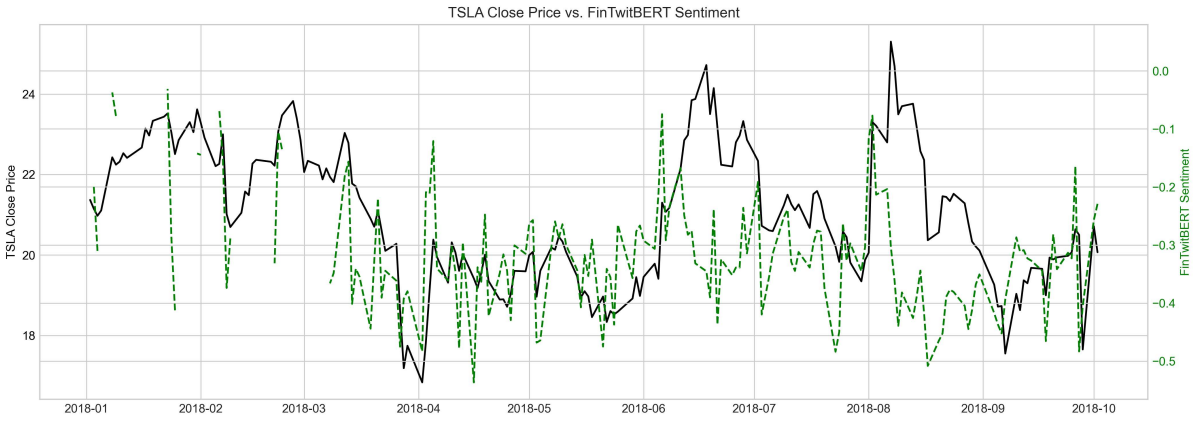

Figure 3.4: Daily closing price of $TSLA compared to average daily FinTwitBERT sentiment scores

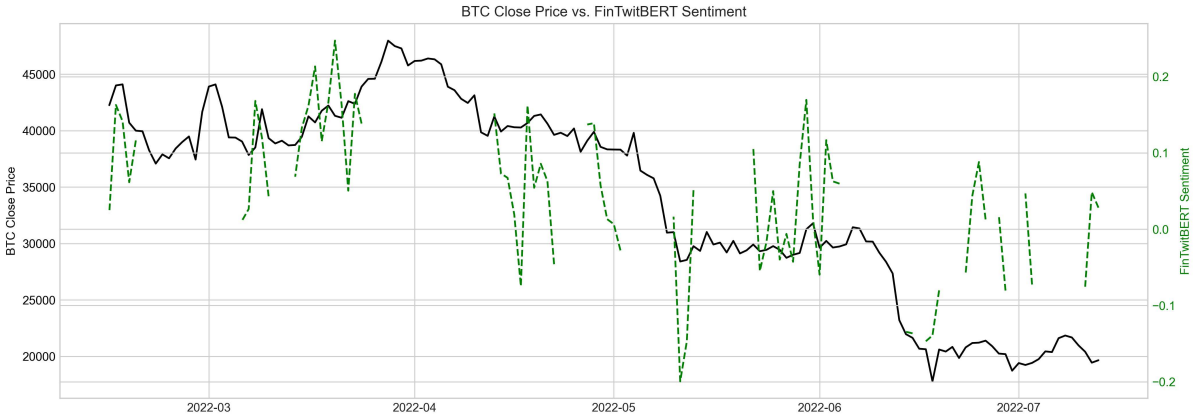

Figure 3.5: Daily closing price of $BTC compared to average daily FinTwitBERT sentiment scores

3.4 Predictive Modeling

This section outlines the forecasting models used to predict the next-day closing price of $TSLA and $BTC. Three types of models were selected with the aim of comparing different methodological approaches: a classical time series model (ARIMA/ARIMAX), a tree-based machine learning model (XGBoost), and a neural network (LSTM). These models differ in complexity and interpretability, allowing for a comparative assessment of their effectiveness in the use of sentiment inputs.

My setup for this project was structured as a regression problem, with the goal of forecasting the actual next-day price rather than classifying the direction of movement. A standard time-aware 80/20 train/test split was used by reserving the final 20% of the time series for out-of-sample predictions.

Each model was trained to predict the closing price on Day *D+1*, using different types of inputs depending on the model architecture. For ARIMA and ARIMAX, which are designed to model temporal dependencies internally, no manually lagged features were needed. In contrast, XGBoost and LSTM models were provided with lagged values (closing prices of the previous days).

Two main setups were tested for each model type: (1) a baseline using only price-based features, and (2) an extended version that incorporated daily aggregated sentiment scores from VADER, TextBlob, and FinTwitBERT. After filtering and aligning the data, some trading days had missing sentiment scores, due to structural problems of the dataset or low tweet volume. To address this, a neutral score (zero) was assigned to preserve continuity as best as possible without creating misleading trends. Forward-filling was avoided, as sentiment can change quickly, and carrying over old values (especially across multiple missing days) risked distorting the true sentiment. This imputation was especially relevant for the $BTC dataset, where nearly 50% of all days had missing sentiment data.

In those cases, Model performance was assessed using root mean squared error (**RMSE**), mean absolute error (**MAE**), and coefficient of determination ($\mathbf{R^2}$) on both the training and test sets. **MAE** indicates how far off the predictions are on average, treating all errors equally. **RMSE** also measures prediction error, but penalizes larger deviations more heavily by squaring them before averaging, making it more sensitive to outliers. $\mathbf{R^2}$ reflects how well the model explains the variance in the actual values, with a value of 1 indicating perfect predictions and values below 0 indicating performance worse than simply predicting the mean of the training values.

Forecasts were visualized alongside actual prices to assess the quality of the predictions over time. For the most promising model configurations, the feature set was further expanded to include additional price-derived indicators such as:

- Daily return: the percentage change between open and close, capturing momentum within one day.
- High–low spread: the difference between daily high and low prices, serving as a measure of volatility.
- Lagged return: the price change from the previous day, capturing short-term trends.

These final models were also tuned using grid search or dropout regularization where applicable. In addition, time series cross-validation was applied to evaluate the consistency of model performance over different time windows and to reduce the risk of overfitting.

3.4.1 ARIMA/ARIMAX

The first set of models applied were classical time series models: ARIMA (Autoregressive Integrated Moving Average) and ARIMAX, which extends ARIMA by including exogenous variables. ARIMA is a widely used statistical model for time series forecasting, based solely on the past values of the target variable. While it might be too simplistic for capturing the non-linear and volatile behavior we observe in financial markets, it can serve as a comparative baseline. Compared to more complex machine learning models or neural networks, these models are relatively transparent and easier to interpret. Previous studies have shown that ARIMA can offer accurate results for short-term stock price forecasts (Adebiyi et al., 2014).

In this study, ARIMA was used to forecast the next-day closing price of $TSLA and $BTC based solely on their historical prices. The optimal model structure selected by the *auto_arima()* function was (0,1,0), which corresponds to a model in *first differences*, without autoregressive (AR) or moving average (MA) components. This means the model has no persistence.

Differencing is done to make a non-stationary time series stationary. A model in first differences means that we do not model the price levels p_t directly, but rather their changes from one period to the next:

$$\Delta p_t = p_t - p_{t-1}$$

With an ARIMA(0,1,0) specification, these changes are assumed to be purely random:

$$\Delta p_t = \varepsilon_t \quad \Rightarrow \quad p_t = p_{t-1} + \varepsilon_t$$

where ε_t is white noise with a mean of zero and constant variance. This is equivalent to a *random walk* process, which implies that prices evolve unpredictably over time and cannot be forecasted from past values alone.

This result is consistent with the Efficient Market Hypothesis (EMH), which suggests that all available information is already reflected in current prices. Therefore, the best prediction for tomorrow's price is simply today's price:

$$\mathbb{E}[p_{t+1}] = p_t$$

The fact that the optimal ARIMA model for both $TSLA and $BTC turned out to be (0,1,0) indicates that (at least for the time period analyzed) the prices for both assets behave like random walks, and no predictable pattern could be extracted from past prices alone.

The same (0,1,0) structure was then applied to the ARIMAX models, which included the VADER, TextBlob, an FinTwitBERT daily averages as exogenous variables to assess whether sentiment data could improve predictive performance beyond the random walk baseline.

3.4.2 XGBoost

The second set of models applied are called XGBoost (Extreme Gradient Boosting), a powerful prediction method based on decision trees. XGBoost builds a series of trees, where each new tree is trained to correct the errors of the previous one, making it a so-called ensemble method. This strategy allows it to efficiently capture non-linear relationships, feature interactions, and subtle patterns in the data. Compared to classical time-series models, XGBoost is more flexible, supports multiple input features, and generally performs well on structured data. This makes it a suitable choice for modeling noisy and complex financial time series like $TSLA and $BTC. Similar ensemble approaches have been successfully applied in sentiment-based market prediction (Koukaras et al., 2022).

As a starting point, the model was trained using only the previous day's closing price (lag-1) as an input feature. This choice was based on a Partial Autocorrelation Function (PACF) analysis of the closing price series, which indicated significant correlation only at the first lag (see Figures 3.6 and 3.7). The model setup was then expanded to include daily sentiment scores from VADER, TextBlob, and FinTwitBERT, both individually and in combination. In the final configuration, additional engineered financial features (daily return, the high–low spread and lagged returns) were also included as predictors.

Performance was evaluated using RMSE, MAE, and R^2 on both training and test sets. To improve generalization and reduce overfitting, hyperparameter tuning was conducted using *GridSearchCV* with time series cross-validation (*TimeSeriesSplit*). Feature importance plots and residual diagnostics were also generated to assess the model's behavior and interpretability.

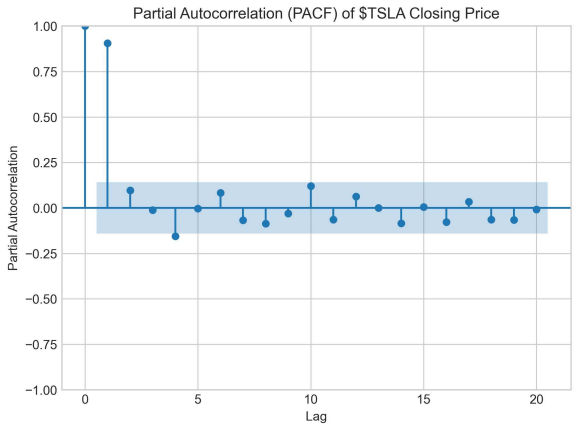

Figure 3.6: PACF of $TSLA Closing Price

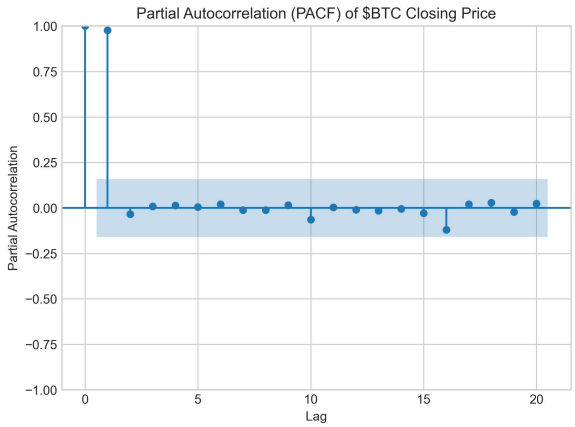

Figure 3.7: PACF of $BTC Closing Price

3.4.3 LSTM

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) designed to handle sequential data. Unlike traditional RNN models, LSTMs can retain information from previous time steps through a memory structure controlled by gates. This allows them to recognize patterns over time and is particularly useful in financial time series forecasting, where price movements can be influenced by trends or events from earlier in the sequence. The selection of an LSTM architecture is supported by previous research that used recurrent models to incorporate sentiment and price data in financial forecasting tasks (e.g., Pant et al. 2018)

In this study, a standard LSTM architecture was implemented to predict the next-day closing price of $TSLA and $BTC. The model consists of a single LSTM layer with 50 units, followed by a dense output layer for direct regression. This structure is commonly used in financial forecasting tasks due to its simplicity, interpretability, and effectiveness on small- to medium-scale datasets. The design was chosen to minimize complexity and reduce the risk of overfitting.

Like before, the basic model input included only lagged closing prices (lag1) as input features. Sentiment scores from VADER, TextBlob, and FinTwitBERT were later added individually and in combination. The most promising models were further enhanced by including financial indicators such as daily return, high-low spread, and lagged return. All input features were scaled using MinMax normalization and reshaped into 3D arrays with the shape (samples, timesteps, features), where each sample consists of a single time step and several input features. This format is required by *Keras* (a Python library that makes it easy to build and train deep learning models) to correctly train the LSTM model.

To further improve performance, the strongest model was tuned using a random search strategy via the *Keras Tuner*. Hyperparameters such as the number of units, dropout rate, L2 regularization strength, and learning rate were optimized to minimize validation loss. The tuned model was evaluated on both the training and test sets using RMSE, MAE, and R^2.

4 Results

4.1 ARIMA/ARIMAX

4.1.1 ARIMA

The basic ARIMA model's predictive performance was evaluated for both $TSLA and $BTC. While in-sample (training) performance was strong for both assets—reflected in high R^2 values of 0.84 for $TSLA and 0.95 for $BTC—the out-of-sample results show a different picture. On the test set, the models performed poorly, with RMSEs of 4.51 for $TSLA and 2926.52 for $BTC, and negative R^2 scores (–7.82 and –8.30, respectively), indicating that the forecasts were worse than simply predicting the mean.

The forecast plots (Figures 4.1 and 4.2) confirm this: the predicted values for both assets appear flat and disconnected from the actual price movements. This is because ARIMA(0,1,0) out-of-sample forecasts simply predict the last known price from the training period, because the model assumes that future changes are random with a mean of zero. In other words, the forecast remains constant over the test period, reflecting the random walk behavior discussed earlier.

These results suggest that price-only ARIMA models are insufficient for capturing the complexity of financial markets, especially during volatile periods. The poor generalization performance underscores the limitations of linear, univariate models and highlights the potential value of incorporating external variables such as sentiment, which is explored in the following sections.

This performance was not unexpected, as ARIMA was primarily used as a baseline model to establish a benchmark for forecasting accuracy using only historical price data. The goal was not to achieve high predictive performance with ARIMA, but rather to compare its limitations against more flexible models that incorporate sentiment and other features.

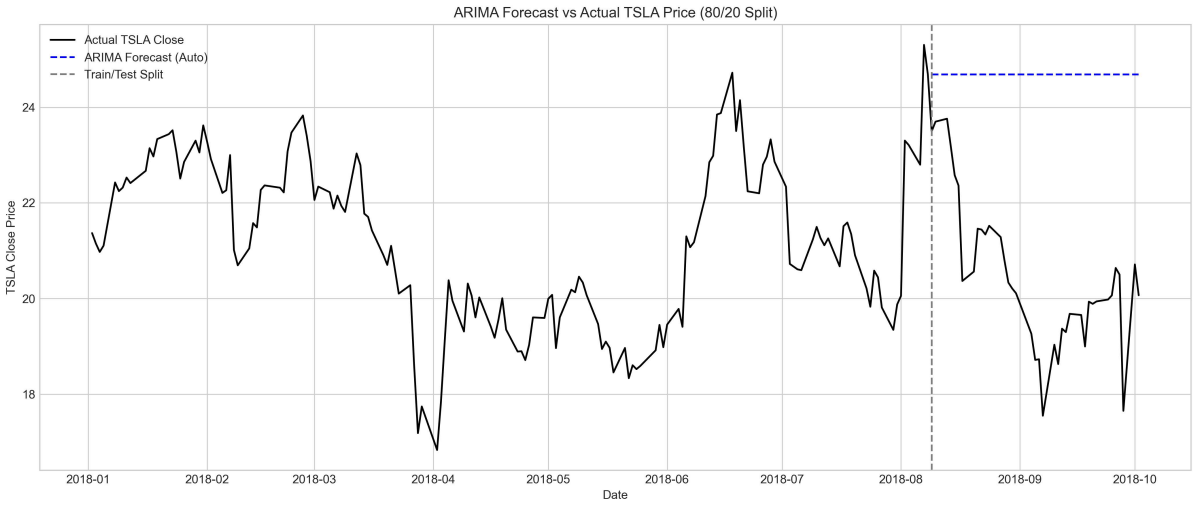

Figure 4.1: ARIMA Forecast vs Actual $TSLA Close Price

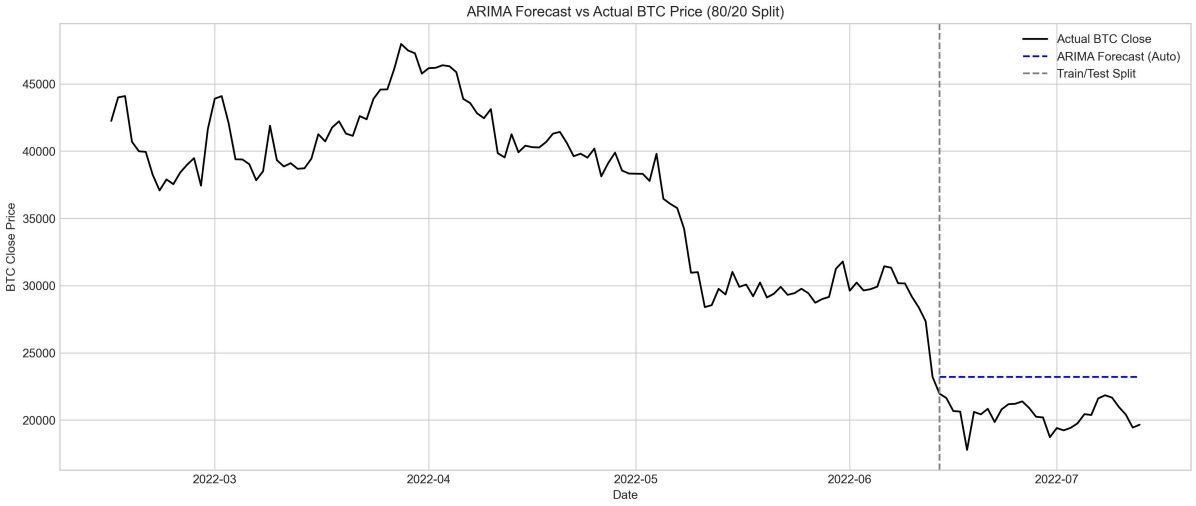

Figure 4.2: ARIMA Forecast vs Actual $BTC Close Price

4.1.2 ARIMAX

To assess whether sentiment data could enhance predictive performance, three ARIMAX models were tested for each asset, incorporating VADER, TextBlob, and FinTwitBERT sentiment scores as exogenous variables. As summarized in Table 1, none of the models achieved meaningful improvements over the baseline ARIMA. For both $TSLA and $BTC, test set RMSEs remained high and R^2 scores were strongly negative across all configurations. Among the six models, ARIMAX with TextBlob sentiment for $TSLA performed marginally better, with the lowest RMSE (4.47) and the least negative R^2 (–7.67). However, as illustrated in Figure 4.3, even this model produced a largely flat forecast that failed to track actual price movements. These findings suggest that adding sentiment as a linear external variable within the ARIMAX framework is insufficient to capture the non-linear and complex dynamics in financial markets. More flexible, non-linear models may be required to meaningfully integrate sentiment into price forecasting.

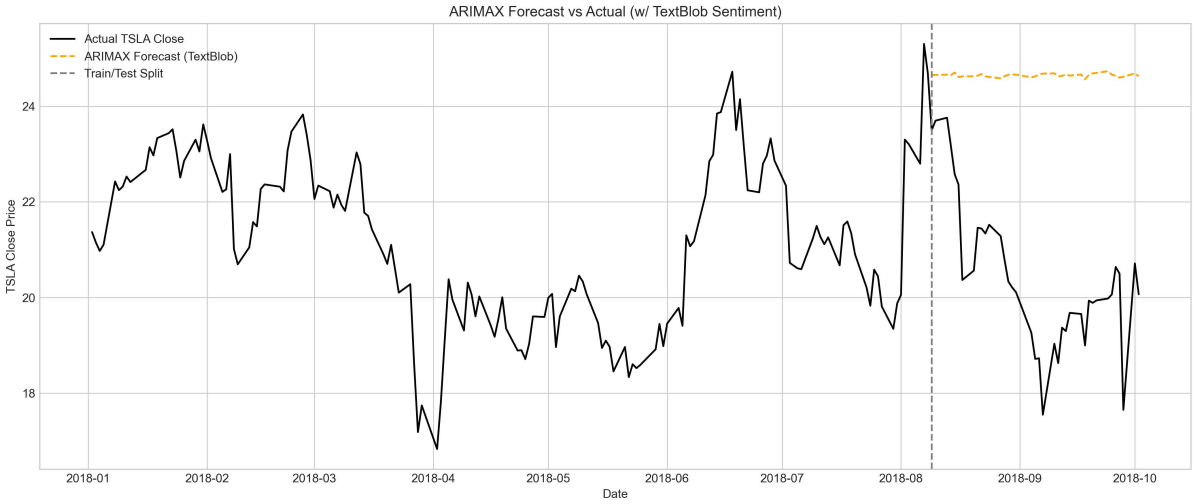

Figure 4.3: ARIMAX Forecast vs Actual $TSLA Close Price (with TextBlob Sentiment)

Table 1: ARIMAX Test Set Performance with Sentiment Features ($TSLA and $BTC)

Asset	Sentiment Model	RMSE	MAE	R^2
$TSLA	VADER	4.58	4.33	–8.10
	TextBlob	**4.47**	**4.21**	**–7.67**
	FinTwitBERT	4.50	4.22	–7.77
$BTC	VADER	3198.84	3036.72	–10.11
	TextBlob	3127.86	2966.21	–9.62
	FinTwitBERT	2883.95	2725.99	–8.03

4.2 XGBoost

The XGBoost models trained on $TSLA showed promising out-of-sample performance, particularly when sentiment features were included. We can see in Table 2 that the final model using lagged prices and all three sentiment scores (VADER, TextBlob, and FinTwitBERT) achieved a test RMSE of 0.89 and R^2 of 0.65 (see also Figure 4.4). An earlier version achieved a similar test performance but also displayed strong signs of overfitting, with a near perfect training R^2 of 0.99. To address this, hyperparameter tuning was performed using randomized search with time series cross-validation. The resulting model produced a more balanced training R^2 of 0.91, suggesting stronger generalization. Feature importance analysis (Figure 4.5) confirmed that the model relied most heavily on a 1 day-price lag (which makes sense given our PACF plot 3.6), followed by FinTwitBERT sentiment, while VADER and TextBlob had minimal influence. Residual diagnostics indicated little to no autocorrelation of the residuals, approximate normality, and no visible time-dependent error patterns. Overall, this suggests that when properly tuned, XGBoost, can effectively use sentiment, particularly from FinTwitBERT, to improve short-term price forecasting.

Table 2: XGBoost Test Set Performance on $TSLA

Feature Set	**RMSE**	**MAE**	**R^2**
Lagged Price Only	1.40	1.05	0.15
Lag + VADER	1.05	0.82	0.52
Lag + TextBlob	1.42	1.07	0.13
Lag + FinTwitBERT	0.93	0.69	0.63
Lag + All Sentiment	0.90	0.72	0.65
Lag + All Sentiment + Financial	0.92	0.67	0.64
Lag + All Sentiment (tuned)	**0.89**	**0.70**	**0.65**

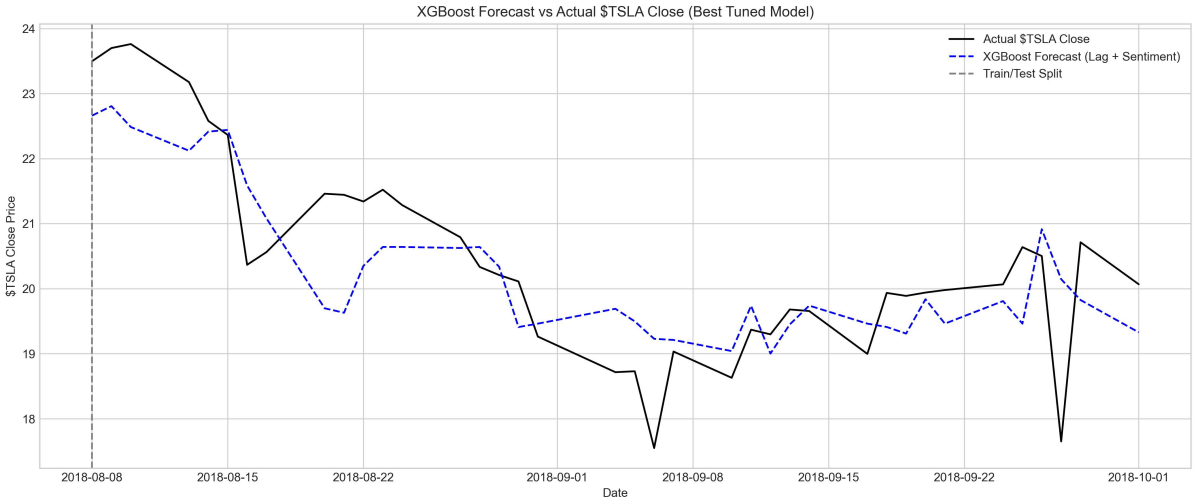

Figure 4.4: XGBoost Forecast vs Actual $TSLA Close - Tuned (Lag + All Sentiment)

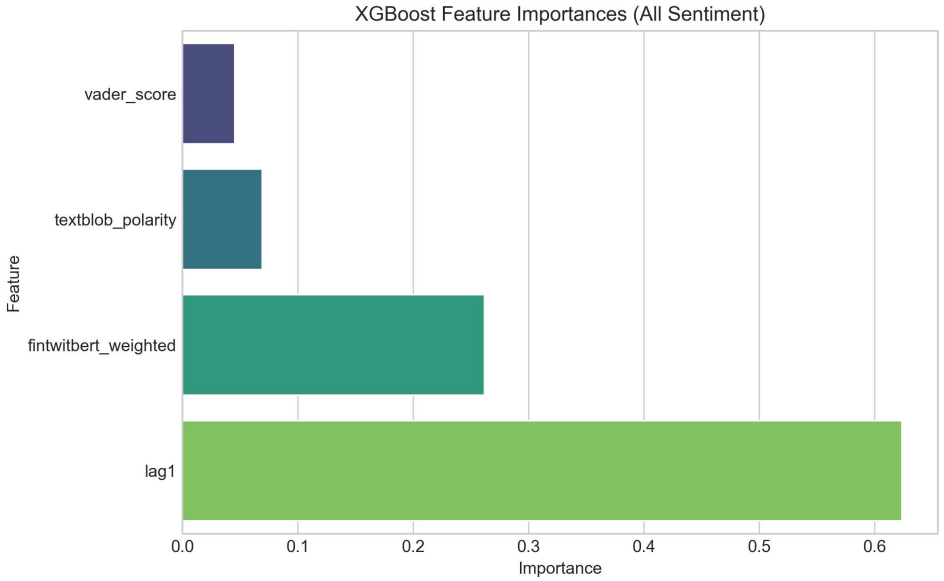

Figure 4.5: Feature Importance for Best XGBoost $TSLA Model (Lag + All Sentiment)

In contrast to the results for $TSLA, the XGBoost models for $BTC failed to produce accurate predictions. As shown in Table 3, the inclusion of sentiment features did not improve test performance significantly. All sentiment models resulted in strongly negative R^2 values, indicating forecasts worse than simply predicting the mean. The best configuration was achieved using FinTwitBERT. While it lowered RMSE to 2057.75, the corresponding R^2 remained negative (–3.87), and the model still overfit the training set (R^2 = 0.99). As shown in Figure 4.6, even the best model failed to follow the actual price trajectory. These results suggest that for $BTC, the XGBoost models were unable to extract meaningful predictive information from sentiment data, possibly due to higher noise levels or gaps in the sentiment data.

Table 3: XGBoost Test Set Performance on $BTC

Feature Set	**RMSE**	**MAE**	**R^2**
Lagged Price Only	3175.65	3027.16	-9.95
+ VADER	2207.85	1975.83	-4.60
+ TextBlob	2191.04	1944.21	-4.52
+ FinTwitBERT	**2057.75**	**1825.09**	**-3.87**
+ All Sentiment	2065.39	1825.74	-3.90
+ All Sentiment + Financial	2118.17	1902.65	-4.16

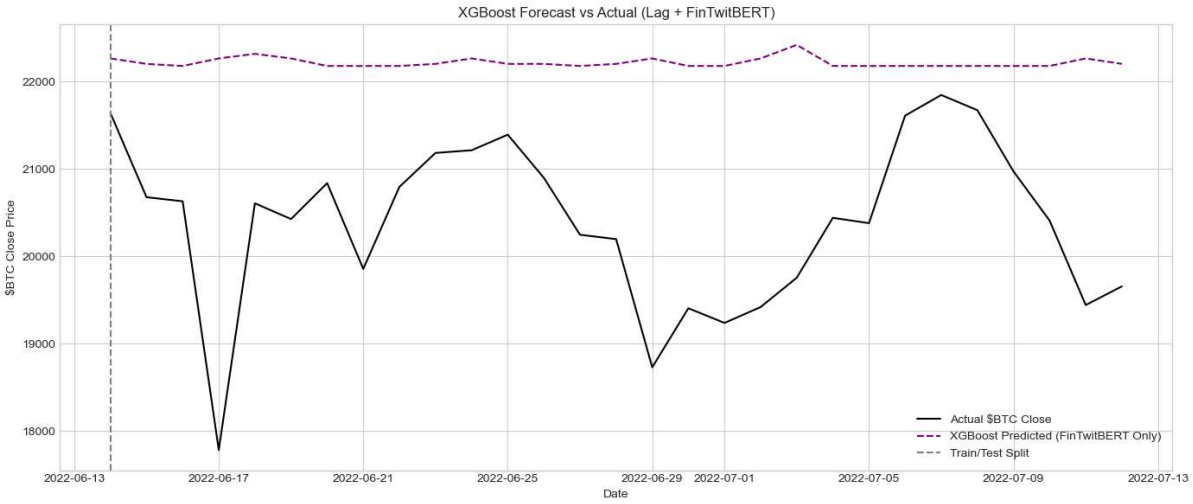

Figure 4.6: XGBoost Forecast vs Actual $BTC Close Price (Best Model with Lag + FinTwitBERT)

4.3 LSTM

The LSTM models achieved strong results when applied to $TSLA data. While the baseline model using only lagged prices reached a test R^2 of 0.49, performance improved steadily with the inclusion of FinTwitBERT and additional sentiment sources. The best performance was obtained using a tuned LSTM model with dropout and L2 regularization, trained on a feature set combining lagged prices, all sentiment scores, and financial indicators. As shown in Table 4, this model achieved an RMSE of 0.74, a test R^2 of 0.76, and a MAE of 0.53—indicating that, on average, the model's daily closing price predictions were off by just 53 cents. To further assess the robustness of the final LSTM model, time series cross-validation was performed using three non-overlapping folds, resulting in a mean RMSE of 1.64 with a standard deviation of 0.43. This suggests that the model generalizes well across different time segments. These results highlight the ability of LSTM architectures to capture patterns over time and benefit from different exogenous inputs, especially when regularization is applied to avoid overfitting (see Figure 4.7).

Table 4: LSTM Test Set Performance on $TSLA

Feature Set	**RMSE**	**MAE**	**R^2**
Lagged Price Only	1.08	0.82	0.49
+ VADER	1.06	0.83	0.51
+ TextBlob	1.03	0.76	0.54
+ FinTwitBERT	0.94	0.76	0.62
+ All Sentiment	0.92	0.74	0.63
+ Financial	0.99	0.75	0.58
+ All Sentiment + Financial	0.79	0.60	0.73
+ All Sentiment + Financial (tuned)	**0.74**	**0.53**	**0.76**

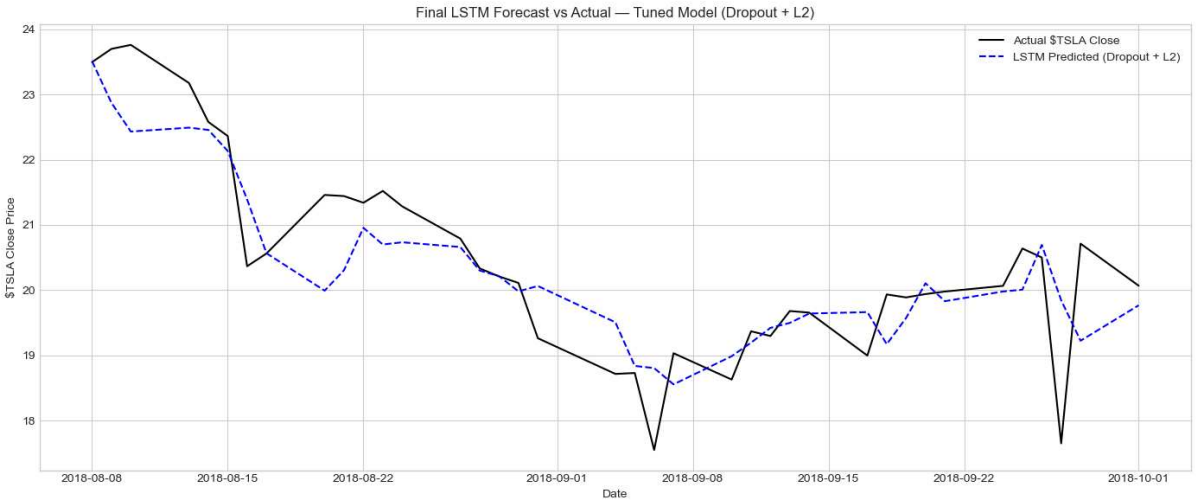

Figure 4.7: LSTM Forecast vs Actual $TSLA Close Price (Best Tuned Model with Sentiment + Financial Features)

Despite multiple attempts to improve performance, the LSTM models consistently failed to achieve acceptable results on the $BTC dataset. Initial models using lagged prices and sentiment scores from VADER, TextBlob, or FinTwitBERT performed poorly, with test set R^2 values ranging from –35.26 to –16.52. Even after incorporating financial indicators such as daily return, high-low spread, and lagged return, the models continued to underperform. As a final attempt, a more complex feature set was constructed by adding longer lag sequences, rolling averages, and short-term volatility measures and tuning steps such as L2 regularization and dropout were performed. While this reduced the RMSE to 1310.92, the test R^2 remained negative (–0.91), as shown in Table 5. The forecast plot in Figure 4.8 further illustrates the model's inability to follow actual price trends. These findings suggest that, for the selected period, neither sentiment data nor extended technical features enabled the LSTM to generate valuable forecasts for Bitcoin.

Table 5: LSTM Test Set Performance on $BTC

Feature Set	**RMSE**	**MAE**	**R^2**
Lagged Price Only	5169.78	5065.61	-28.02
+ VADER	4532.28	4389.97	-21.30
+ TextBlob	5778.74	5685.59	-35.26
+ FinTwitBERT	4560.22	4436.68	-21.58
+ All Sentiment	4016.54	3851.89	-16.52
+ Financial Only	4541.02	4458.86	-21.39
+ lags 1–7 + Financial	1909.85	1479.72	-3.05
+ lags 1–7 + Financial (tuned)	**1310.92**	**981.63**	**-0.91**

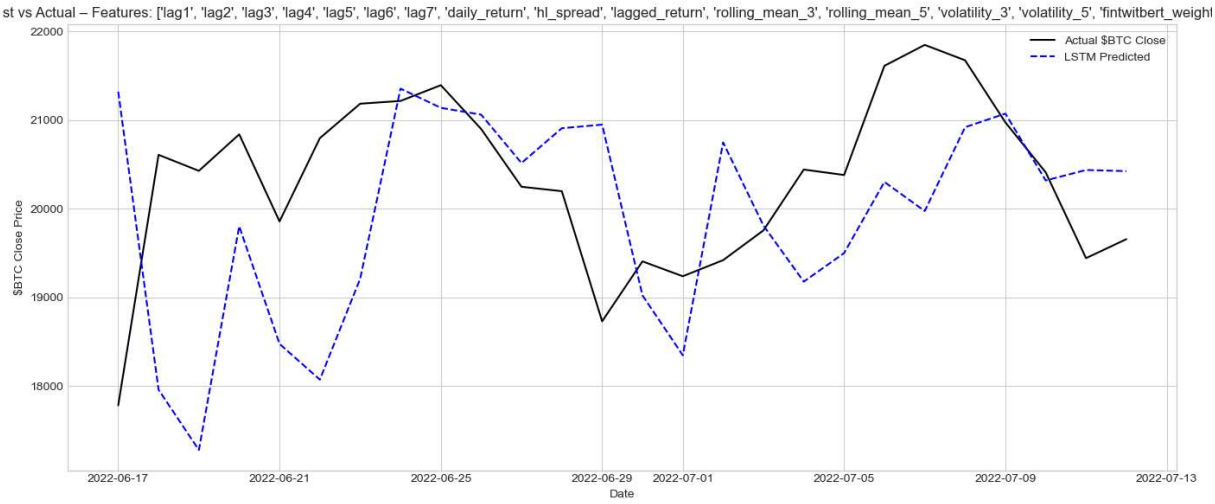

Figure 4.8: LSTM Forecast vs Actual $BTC Close Price (Best Feature Set: Extended Lags, Sentiment, and Financial Indicators)

5 Conclusion

5.1 Discussion

The results of this study show a clear contrast between the performance of forecasting models applied to $TSLA and those applied to $BTC. For $TSLA, the models (especially the tuned LSTM and XGBoost architectures) performed better than I expected, achieving test R^2 values close to 0.80. This suggests that sentiment scores can provide valuable predictive information when combined with traditional price features. Interestingly, the more advanced FinTwitBERT sentiment model consistently outperformed computationally simpler alternatives like VADER and TextBlob. While those lighter models are easier and faster to implement, FinTwitBERT appears to add significantly higher value. For any serious attempt at sentiment-based forecasting, using a more powerful transformer based model like FinTwitBERT is likely the most effective approach.

The findings indicate that even for blue-chip stocks like $TSLA, market sentiment expressed on social media contains information that can enhance short-term forecasting accuracy. In contrast, the performance of all models on $BTC was significantly weaker. This was surprising, as I initially assumed that sentiment-driven forecasting would work especially well for Bitcoin, given how much it is discussed online and its reputation for being influenced by hype and public opinion. However, the high volatility of Bitcoin, together with the rather bullish tone of tweets and the sharp price drop at the beginning of the observed time period, are factors that may have impacted the models' ability to make accurate predictions. Additionally, the large number of missing tweet days and potential noise in the sentiment dataset likely contributed to the poor results. Overall, while sentiment analysis appears to improve forecasting for traditional assets, its value for highly volatile assets like Bitcoin remains uncertain and may be dependent on context and time frame.

What surprised me during this project was the limited amount of high-quality academic research directly combining NLP-based sentiment with financial time-series forecasting. Given the increasing relevance of social media and the availability of open-source sentiment tools, I had expected more coverage in academic literature. After discussing this topic with friends currently working in finance, none reported the use of such tools in their daily work, partly due to the fact that they do not evaluate stocks based on the market prices and their employers are more interested in the price of a stock in 3-5 years rather than the next day. This highlights the fact that sentiment analysis tends to be more helpful for short-term rather than long-term predictions, which would naturally make it more relevant to day traders and other short-term investors.

When attempting to forecast market prices, we must keep in mind that we are competing with some of the most skilled, competitive, and resourceful players in the world.
It seems logical that advanced sentiment-based approaches, more refined than those explored in this thesis, are already being used by hedge funds or other trading firms. However, it appears that sentiment-based forecasting has not yet become common practice for many professional investors and could serve as a valuable addition to traditional valuation and forecasting methods.

These findings also show that there is much room for fruitful future research. One promising field is intraday or real-time forecasting, which might be able to better capture the short-term impact of sentiment changes and enable a more precise analysis of the correlation between tweet timing and price movements. Expanding the scope of social media data to include other platforms such as Reddit or financial news comment sections could provide valuable additional sentiment data and improve model performance. Furthermore, training models on larger datasets with more diverse assets and time periods could help assess the generalizability of sentiment-based forecasting. It would also be interesting to explore the role of sentiment in forecasting other cryptocurrencies, including so-called meme coins, which tend to be even more sentiment-driven and speculative. Finally, experimenting with more sophisticated model architectures may further improve predictive accuracy.

5.2 Limitations

This study faced several limitations that should be taken into account when interpreting the results. One of the main challenges was obtaining high-quality social media data. Accessing large amounts of up-to-date X data is very expensive, so I had to rely on freely available datasets from Kaggle. While these datasets were useful, there was little transparency regarding how exactly the data had been collected, filtered, or pre-processed. This raises concerns about potential bias and data quality.

Originally, I had planned to analyze the exact same time frame for both Bitcoin and the selected blue-chip stock to make the comparison more meaningful. However, the first Bitcoin dataset I used contained a large amount of spam and, after filtering, there were almost no posts left to analyze. I then switched to an alternative dataset, which contained less spam but came with structural issues and corrupted data entries. This forced me to change the original time frame. In the end, I manually identified a 150-day period with the fewest missing days, which became the final time frame for Bitcoin. The selection of the timeframe based on data availability may have impacted the results.

Another limitation relates to the sentiment models that were used. FinTwitBERT showed strong performance, but none of the sentiment outputs were manually validated, so any misclassifications could have impacted the model performance. Furthermore, the sentiment scores were calculated as simple daily averages without taking into account the influence or credibility of individual users or distinguishing between different types of content (e.g., news vs. opinion), this may have introduced noise.

The forecasting results for $TSLA are promising, but since only one time frame and one stock were analyzed, the generalizability of these findings remains limited. Similarly, the weak performance on $BTC may not apply to other time periods and cryptocurrencies. Finally, this study focused exclusively on prediction and did not attempt to explore causal relationships between sentiment and price movements, so no conclusions about causality can be drawn from the results.

References

A. A. Adebiyi, A. O. Adewumi, and C. K. Ayo. Stock price prediction using the arima model. In *Proceedings - UKSim-AMSS 16th International Conference on Computer Modelling and Simulation, UKSim 2014*, pages 106–112, 2014. ISBN 9781479949236. doi: 10.1109/UKSim.2014.67.

S. Akkerman and T. Koornstra. Fintwitbert: A specialized language model for financial tweets, 2023. URL https://github.com/TimKoornstra/FinTwitBERT.

W. Antweiler and M. Z. Frank. The market impact of corporate news stories. Technical report, UBC Sauder School of Business, 2004.

W. Antweiler and M. Z. Frank. Do u.s. stock markets typically overreact to corporate news stories? Technical report, UBC Sauder School of Business, 2006.

A. Asemi, H. N. Phuong, and M. Alshafeey. Predicting bitcoin price movement through sentiment analysis: A comprehensive study. In *InternationalWorkshop on Big Data in Emergent Distributed Environments ((BiDEDE)*. Association for Computing Machinery, Inc, 6 2024. ISBN 9798400706790. doi: 10.1145/3663741.3664791.

D. Bakas, G. Magkonis, and E. Y. Oh. What drives volatility in bitcoin market? *Finance Research Letters*, 50, 2022. ISSN 15446123. doi: 10.1016/j.frl.2022.103237.

J. Bollen, H. Mao, and X. Zeng. Twitter mood predicts the stock market. *Journal of Computational Science*, 2:1–8, 2011. ISSN 18777503. doi: 10.1016/j.jocs.2010.12.007.

CFI-Team. Random walk theory - definition, history, implications of the theory, 2020. URL https://corporatefinanceinstitute.com/resources/career-map/sell-side/capital-markets/what-is-the-random-walk-theory/.

CFI-Team. Efficient markets hypothesis - understanding and testing emh, 2021. URL https://corporatefinanceinstitute.com/resources/career-map/sell-side/capital-markets/efficient-markets-hypothesis/.

I. Georgoula, D. Pournarakis, C. Bilanakos, D. N. Sotiropoulos, and G. M. Giaglis. Using time-series and sentiment analysis to detect the determinants of bitcoin prices. In *Mediterranean Conference on Information Systems (MCIS)*, volume 20. Association for Information Systems AIS Electronic Library (AISeL), 2015. URL http://aisel.aisnet.org/mcis2015http://aisel.aisnet.org/mcis2015/20.

C. J. Hutto and E. Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of the International AAAI Conference on Web and Social Media*, pages 216–225, 2014. URL http://sentic.net/.

A. John, S. Shen, and T. Wilson. China's top regulators ban crypto trading and mining, sending bitcoin tumbling | reuters. *Reuters*, 2021. URL https://www.reuters.com/world/china/china-central-bank-vows-crackdown-cryptocurrency-trading-2021-09-24/.

S. V. Kolasani and R. Assaf. Predicting stock movement using sentiment analysis of twitter feed with neural networks. *Journal of Data Analysis and Information Processing*, 08:309–319, 2020. ISSN 2327-7211. doi: 10.4236/jdaip.2020.84018.

P. Koukaras, C. Nousi, and C. Tjortjis. Stock market prediction using microblogging sentiment analysis and machine learning. *Telecom*, 3:358–378, 2022. ISSN 26734001. doi: 10.3390/telecom3020019.

S. Loria. Textblob: Simplified text processing, 2025. URL https://textblob.readthedocs.io/en/dev/.

B. G. Malkiel. The efficient market hypothesis and its critics. *Journal of Economic Perspectives*, 17, 2003.

A. Morrow. Everything you need to know about how a reddit group blew up gamestop's stock | cnn business. *CNN Business*, 2021. URL https://edition.cnn.com/2021/01/27/investing/gamestop-reddit-stock/index.html.

V. S. Pagolu, K. N. R. Challa, G. Panda, and B. Majhi. Sentiment analysis of twitter data for predicting stock market movements. In *International conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*. IEEE, 2016. ISBN 9781509046201.

D. R. Pant, P. Neupane, and A. Poudel. Recurrent neural network based bitcoin price prediction by twitter sentiment analysis. In *IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)*. IEEE, 2018. ISBN 9781538662274.

M. Price and P. Schroeder. Tesla, musk pay $40 million to settle tweet charges, musk to resign as chairman | reuters. *Reuters*, 2018. URL https://www.reuters.com/article/technology/tesla-musk-pay-40-million-to-settle-tweet-charges-musk-to-resign-as-chairman-idUSKCN1MA03H/.

M. Taddy. *Business Data Science*, volume 1. McGraw-Hill Education LLC, 2019. ISBN 978-1-26-045278-5.

R. Tumarkin and R. F. Whitelaw. News or noise? internet postings and stock prices. *Financial Analysts Journal*, 57, 2001.

F. Valencia, A. Gómez-Espinosa, and B. Valdés-Aguirre. Price movement prediction of cryptocurrencies using sentiment analysis and machine learning. *Entropy*, 21, 6 2019. ISSN 10994300. doi: 10.3390/e21060589.

Y. Yang, M. C. S. UY, and A. Huang. Finbert: A pretrained language model for financial communications. Technical report, Hong Kong University of Science and Technology, 2020. URL http://arxiv.org/abs/2006.08097.

Appendix

File - /Users/malvin/DataspellProjects/MasterThesis/X Data TSLA.ipynb

```
#%%
# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from datetime import datetime, timezone
from datetime import timedelta

# VADER sentiment analysis
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

# TextBlob sentiment analysis
from textblob import TextBlob

# FinTwitBERT sentiment analysis via Hugging Face Transformers
from transformers import pipeline

# US holidays for trading day filtering
import holidays

#%%
# Defining data for the dataframe
data_stocks = pd.read_csv('stock_tweets.csv')

# Creating the dataframe
df_stocks = pd.DataFrame(data_stocks)

# Convert 'created_at' to datetime format
df_stocks['post_date'] = df_stocks['post_date'].apply(lambda x: datetime.fromtimestamp(x, tz=timezone.utc))

# Remove unwanted columns comment_num, retweet_num, like_num
df_stocks.drop(columns=["comment_num", "retweet_num", "like_num", "writer"], inplace=True)

#%%
# Step 1: Define the date range for filtering
start_date = datetime(2018, 1, 1)
end_date = datetime(2018, 10, 3)  # end is exclusive, so this includes Oct 2

# Step 2: Convert timestamp to New York-local datetime
df_stocks['post_date'] = df_stocks['post_date'].dt.tz_convert('America/New_York')

# Step 3: Filter tweets between Start and End dates based on NY-local date
df_trading_period = df_stocks[
    (df_stocks["post_date"].dt.date >= start_date.date()) &
    (df_stocks["post_date"].dt.date < end_date.date())
    ]

# Step 4: Filter for Tweets containing TSLA cashtag
df_trading_period_TSLA = df_trading_period[
    df_trading_period["body"].str.contains(r"\$TSLA", case=False, na=False)
].copy()

# Step 5: Convert datetime to just date (NY-local), but keep column name
df_trading_period_TSLA['post_date'] = df_trading_period_TSLA['post_date'].dt.date

# Step 6: Get US holidays and add Good Friday
us_holidays = holidays.UnitedStates(years=[2018])
extra_market_holidays = {datetime(2018, 3, 30).date()}  # Good Friday
holiday_dates = set(us_holidays.keys()) | extra_market_holidays

# Step 7: Summary of trading days
start_date_only = start_date.date()
end_date_only = (end_date - timedelta(days=1)).date()
total_weekdays = sum(1 for d in pd.date_range(start=start_date, end=end_date - timedelta(days=1), freq='B'))
holidays_in_period = sum(1 for day in holiday_dates if start_date_only <= day <= end_date_only and day.
weekday() < 5)

print(f"Total weekdays (Mon-Fri) between {start_date_only} and {end_date_only}: {total_weekdays}")
print(f"US holidays on weekdays (incl. Good Friday): {holidays_in_period}")
print(f"Expected trading days after holidays removed: {total_weekdays - holidays_in_period}")

# Step 8: Remove tweets from weekends and market holidays
df_trading_period_TSLA = df_trading_period_TSLA[
    df_trading_period_TSLA['post_date'].apply(lambda x: x.weekday() < 5)  # Monday-Friday
]
df_trading_period_TSLA = df_trading_period_TSLA[
    ~df_trading_period_TSLA['post_date'].isin(holiday_dates)  # Remove holidays
]

# Clean memory
del df_stocks
#%%
print(f"Total TSLA Tweets before filtering:", len(df_trading_period_TSLA))
#%%
# Preprocessing the TSLA tweets
df_trading_period_TSLA['body'] = (
    df_trading_period_TSLA['body']
    .str.lower()
```

```
    .str.replace(r'@\w+', '', regex=True)  # remove mentions
    .str.replace(r'http\S+|www\S+|https\S+', '', regex=True)  # remove URLs
    .str.replace(r'\s+', ' ', regex=True)  # normalize spaces
    .str.strip()  # remove leading/trailing whitespace
)

# Remove very short posts
df_trading_period_TSLA = df_trading_period_TSLA[df_trading_period_TSLA['body'].str.len() > 25]

# Define spammy keywords (TSLA-relevant version)
spam_keywords = [
    'giveaway', 'retweet', 'follow', 'airdrop', 'bonus',
    'free', 'guaranteed', 'whatsapp', 'dm me', 'click here',
    'price alert', 'stock alert', 'current tsla price'
]

# Remove tweets containing spam keywords
df_trading_period_TSLA = df_trading_period_TSLA[
    ~df_trading_period_TSLA['body'].str.contains('|'.join(spam_keywords), na=False)
]

# Remove tweets with phone numbers or emails
df_trading_period_TSLA = df_trading_period_TSLA[
    ~df_trading_period_TSLA['body'].str.contains(r'\+?\d{7,}', regex=True)
]
df_trading_period_TSLA = df_trading_period_TSLA[
    ~df_trading_period_TSLA['body'].str.contains(r'\S+@\S+', regex=True)
]

# Drop duplicate tweets
df_trading_period_TSLA = df_trading_period_TSLA.drop_duplicates(subset='body')

# Remove tweets with more than 3 hashtags or more than 2 cashtags
df_trading_period_TSLA = df_trading_period_TSLA[
    (df_trading_period_TSLA['body'].str.count('#') <= 3) &
    (df_trading_period_TSLA['body'].str.count(r'\$') <= 2)
    ]

# remove tweets that consist of more than 30% numbers, but allow normal text with some numbers
df_trading_period_TSLA = df_trading_period_TSLA[
    df_trading_period_TSLA['body'].apply(lambda x: sum(c.isdigit() for c in x) / len(x) < 0.3)
]
#%%
# Group by date and count tweets per day
daily_counts_TSLA = df_trading_period_TSLA.groupby(df_trading_period_TSLA["post_date"]).size()

# Average tweets per day
average_tsla_per_day = daily_counts_TSLA.mean()
print(f"Average $TSLA tweets per day: {average_tsla_per_day:.2f}")

# Min/max tweet counts and corresponding dates
min_tweets = daily_counts_TSLA.min()
min_date = daily_counts_TSLA.idxmin()
max_tweets = daily_counts_TSLA.max()
max_date = daily_counts_TSLA.idxmax()
print(f"Maximum daily $TSLA tweets: {max_tweets} on {max_date}")
print(f"Minimum daily $TSLA tweets: {min_tweets} on {min_date}")

# Total tweets
print(f"Total TSLA Tweets after filtering:", len(df_trading_period_TSLA))
#%%
df_trading_period_TSLA
#%%
# Plot
plt.figure(figsize=(12, 6))
daily_counts_TSLA.plot(kind='line')
plt.title("Daily Number of $TSLA Tweets")
plt.xlabel("Date")
plt.ylabel("Number of Tweets")
plt.grid(True)
plt.tight_layout()
plt.show()
#%%
# Filter for days with at least 200 tweets
valid_days_TSLA = daily_counts_TSLA[daily_counts_TSLA >= 200].index

# Filter the dataframe to only include valid days
df_trading_period_TSLA_filtered = df_trading_period_TSLA[df_trading_period_TSLA["post_date"].isin(
    valid_days_TSLA)]

# Count and list dropped days
dropped_days_TSLA = daily_counts_TSLA[daily_counts_TSLA < 200]
removed_dates_list = dropped_days_TSLA.index.tolist()

print(f"Dropped {len(dropped_days_TSLA)} days due to low tweet volume (<200).")

# show the dropped days
print("Dropped days and tweet counts:")
print(dropped_days_TSLA)
```

```
177
178 # Save the dropped dates list to a CSV
179 pd.Series(removed_dates_list).to_csv('dropped_days_tsla.csv', index=False)
180 #%%
181 # Unique tweet dates after filtering
182 tweet_dates = sorted(df_trading_period_TSLA_filtered['post_date'].unique())
183 print(f"Number of tweet days: {len(tweet_dates)}")
184 #%%
185 # Plot
186 # Recalculate tweet counts for filtered days
187 daily_counts_TSLA_filtered = df_trading_period_TSLA_filtered.groupby('post_date').size()
188
189 # Re-plot the updated tweet volume chart
190 plt.figure(figsize=(12, 6))
191 daily_counts_TSLA_filtered.plot(kind='line')
192 plt.title("Daily Number of $TSLA Tweets (Filtered for ≥200)")
193 plt.xlabel("Date")
194 plt.ylabel("Number of Tweets")
195 plt.grid(True)
196 plt.tight_layout()
197 plt.show()
198 #%%
199 num_days_before = df_trading_period_TSLA['post_date'].nunique()
200 print(f"Number of trading days before filtering: {num_days_before}")
201
202 num_days_remaining = df_trading_period_TSLA_filtered['post_date'].nunique()
203 print(f"Remaining trading days with ≥200 tweets: {num_days_remaining}")
204
205 # create a df with random sample of 200 tweets for each day
206
207 df_TSLA_sampled = (
208     df_trading_period_TSLA_filtered
209     .groupby('post_date')
210     .apply(lambda x: x.sample(n=200, random_state=42))
211     .reset_index(drop=True)
212 )
213 #%%
214 # VADER Sentiment Analysis
215
216 # Initialize the VADER sentiment analyzer
217 analyzer = SentimentIntensityAnalyzer()
218 # Apply VADER to each tweet in the sampled dataframe
219 df_TSLA_sampled = df_TSLA_sampled.copy()
220 df_TSLA_sampled['vader_score'] = df_TSLA_sampled['body'].apply(
221     lambda x: analyzer.polarity_scores(x)['compound']
222 )
223
224 # Group by date and calculate daily average sentiment
225 daily_sentiment_TSLA = df_TSLA_sampled.groupby('post_date')['vader_score'].mean().reset_index()
226
227 # Plotting
228 plt.figure(figsize=(12, 6))
229 plt.plot(daily_sentiment_TSLA['post_date'], daily_sentiment_TSLA['vader_score'], marker='o')
230 plt.title("Daily Average VADER Sentiment of $TSLA Tweets (Sampled)")
231 plt.xlabel("Date")
232 plt.ylabel("Average Sentiment Score")
233 plt.xticks(rotation=45)
234 plt.grid(True)
235 plt.tight_layout()
236 plt.show()
237
238
239 #%%
240 df_TSLA_sampled
241 #%%
242 plt.figure(figsize=(12, 6))
243 plt.hist(df_TSLA_sampled['vader_score'], bins=50)
244 plt.title("Distribution of VADER Sentiment Scores ($TSLA Tweets)")
245 plt.xlabel("Sentiment Score")
246 plt.ylabel("Tweet Count")
247 plt.savefig("vader_sentiment_histogram_TSLA.png", dpi=300, bbox_inches='tight')
248 plt.show()
249 #%%
250 # Inspect tweets that received a 0.0 score
251 neutral_tweets = df_TSLA_sampled[df_TSLA_sampled['vader_score'] == 0.0]
252 neutral_tweets
253 #%%
254 # Apply TextBlob polarity to each tweet
255 df_TSLA_sampled['textblob_polarity'] = df_TSLA_sampled['body'].apply(
256     lambda x: TextBlob(x).sentiment.polarity
257 )
258
259 # Group by date and calculate daily average polarity
260 daily_textblob_sentiment_TSLA = (
261     df_TSLA_sampled.groupby('post_date')['textblob_polarity']
262     .mean()
263     .reset_index()
264 )
265
```

```
# Plotting
plt.figure(figsize=(12, 6))
plt.plot(daily_textblob_sentiment_TSLA['post_date'], daily_textblob_sentiment_TSLA['textblob_polarity'], marker='o', color='orange')
plt.title("Daily Average TextBlob Sentiment of $TSLA Tweets (Sampled)")
plt.xlabel("Date")
plt.ylabel("Average Polarity Score")
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
#%%
# Plot a histogram of TextBlob polarity
plt.figure(figsize=(10, 6))
plt.hist(df_TSLA_sampled['textblob_polarity'], bins=50, color='orange', edgecolor='black')
plt.title('Distribution of TextBlob Polarity Scores ($TSLA Tweets)')
plt.xlabel('Polarity Score')
plt.ylabel('Tweet Count')
plt.grid(True)
plt.tight_layout()
plt.savefig("textblob_sentiment_histogram_TSLA.png", dpi=300, bbox_inches='tight')
plt.show()
#%%
# Sort by polarity
sorted_textblob = df_TSLA_sampled.sort_values(by='textblob_polarity', ascending=True)

# 10 most negative tweets
print("\n--- 10 Most Negative Tweets (TextBlob) ---")
for idx, row in sorted_textblob.head(10).iterrows():
    print(f"Score: {row['textblob_polarity']:.4f} | Tweet: {row['body']}\n")

# 10 most positive tweets
print("\n--- 10 Most Positive Tweets (TextBlob) ---")
for idx, row in sorted_textblob.tail(10).iterrows():
    print(f"Score: {row['textblob_polarity']:.4f} | Tweet: {row['body']}\n")


    # Filter tweets with neutral polarity
neutral_tweets_textblob = df_TSLA_sampled[df_TSLA_sampled['textblob_polarity'] == 0]

# Show a few examples
neutral_tweets_textblob
#%%
# Step 1: Initialize the FinTwitBERT sentiment analysis pipeline
fintwitbert_pipe = pipeline("sentiment-analysis", model="StephanAkkerman/FinTwitBERT-sentiment")

# Step 2: Apply FinTwitBERT to a list of tweets (batched)
def get_fintwitbert_sentiments(texts, batch_size=64):
    results = []
    for i in range(0, len(texts), batch_size):
        batch = texts[i:i+batch_size]
        batch_results = fintwitbert_pipe(batch, truncation=True, max_length=512)
        results.extend(batch_results)
    return results

# Step 3: Apply batch sentiment function to sampled tweets
fintwitbert_batch_results = get_fintwitbert_sentiments(df_TSLA_sampled['body'].tolist())

# Step 4: Unpack the results
df_TSLA_sampled['fintwitbert_label'] = [r['label'] for r in fintwitbert_batch_results]
df_TSLA_sampled['fintwitbert_score'] = [r['score'] for r in fintwitbert_batch_results]

# Step 5: Convert labels to numeric
label_to_numeric = {'BULLISH': 1, 'NEUTRAL': 0, 'BEARISH': -1}
df_TSLA_sampled['fintwitbert_numeric'] = df_TSLA_sampled['fintwitbert_label'].map(label_to_numeric)

# Step 6: Multiply label by confidence to create a weighted sentiment score
df_TSLA_sampled['fintwitbert_weighted'] = df_TSLA_sampled['fintwitbert_numeric'] * df_TSLA_sampled['fintwitbert_score']

# Step 7: Aggregate to daily averages
daily_sentiment_TSLA_fintwitbert = df_TSLA_sampled.groupby('post_date')['fintwitbert_weighted'].mean().reset_index()
#%%
# Step 8: Plot the results
plt.figure(figsize=(12, 6))
plt.plot(daily_sentiment_TSLA_fintwitbert['post_date'], daily_sentiment_TSLA_fintwitbert['fintwitbert_weighted'], marker='o', color='green')
plt.title("Daily Average FinTwitBERT Sentiment of $TSLA Tweets")
plt.xlabel("Date")
plt.ylabel("Weighted Sentiment Score")
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()

plt.style.use('seaborn-v0_8-whitegrid')
plt.figure(figsize=(8, 6))

```

```
plt.hist(df_TSLA_sampled['fintwitbert_weighted'].dropna(), bins=50, color='lightgreen', edgecolor='black')
plt.title('FinTwitBERT Weighted Sentiment Score', fontsize=14)
plt.xlabel('Weighted Score', fontsize=12)
plt.ylabel('Tweet Count', fontsize=12)
plt.axvline(0, color='green', linestyle='--', linewidth=1)

plt.tight_layout()
plt.savefig("fintwitbert_weighted_sentiment_histogram.png", dpi=300, bbox_inches='tight')
plt.show()
#%%

plt.style.use('seaborn-v0_8-whitegrid')
fig, axes = plt.subplots(1, 3, figsize=(20, 6), sharey=True)  # wider + taller

# VADER
axes[0].hist(df_TSLA_sampled['vader_score'], bins=40, color='skyblue', edgecolor='black')
axes[0].set_title('VADER Sentiment Distribution - $TSLA', fontsize=16)
axes[0].set_xlabel('Score', fontsize=14)
axes[0].set_ylabel('Tweet Count', fontsize=14)
axes[0].tick_params(labelsize=12)
axes[0].axvline(0, color='black', linestyle='--', linewidth=1)

# TextBlob
axes[1].hist(df_TSLA_sampled['textblob_polarity'], bins=40, color='orange', edgecolor='black')
axes[1].set_title('TextBlob Polarity Distribution - $TSLA', fontsize=16)
axes[1].set_xlabel('Polarity', fontsize=14)
axes[1].tick_params(labelsize=12)
axes[1].axvline(0, color='black', linestyle='--', linewidth=1)

# FinTwitBERT
axes[2].hist(df_TSLA_sampled['fintwitbert_weighted'].dropna(), bins=50, color='lightgreen', edgecolor='
    black')
axes[2].set_title('FinTwitBERT Weighted Sentiment Score - $TSLA', fontsize=16)
axes[2].set_xlabel('Weighted Score', fontsize=14)
axes[2].tick_params(labelsize=12)
axes[2].axvline(0, color='black', linestyle='--', linewidth=1)

plt.tight_layout()
plt.savefig("sentiment_histograms_all_tools_horizontal_larger.png", dpi=300, bbox_inches='tight')
plt.show()
#%%
# TSLA - FinTwitBERT Sentiment Class Distribution
plt.style.use('seaborn-v0_8-whitegrid')
plt.figure(figsize=(8, 6))

plt.hist(df_TSLA_sampled['fintwitbert_numeric'], bins=[-1.5, -0.5, 0.5, 1.5],
         color='green', edgecolor='black', align='mid', rwidth=0.6)
plt.xticks([-1, 0, 1], ['Bearish', 'Neutral', 'Bullish'])
plt.title('TSLA - FinTwitBERT Sentiment Class Distribution', fontsize=14)
plt.xlabel('Sentiment Class', fontsize=12)
plt.ylabel('Tweet Count', fontsize=12)
plt.tight_layout()
plt.savefig("fintwitbert_sentiment_class_TSLA.png", dpi=300, bbox_inches='tight')
plt.show()
#%%
# Inspect tweets that received a neutral FinTwitBERT score
neutral_tweets = df_TSLA_sampled[df_TSLA_sampled['fintwitbert_label'] == 'NEUTRAL']
neutral_tweets
#%%
# Save sentiment DataFrame
df_TSLA_sampled.to_csv("TSLA_sentiment_sampled.csv", index=False)
```

```
#%% raw
```python
#%%
#  Data Handling
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

#  Visualization
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use('seaborn-v0_8-whitegrid')

# Sentiment Analysis
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from textblob import TextBlob
from transformers import pipeline  # For FinTwitBERT

# Time Series Analysis
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from pmdarima import auto_arima
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.graphics.tsaplots import plot_acf

import statsmodels.api as sm


# for XGBoost
from xgboost import XGBRegressor
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer
from sklearn.model_selection import GridSearchCV
import scipy.stats as stats

# for LSTM
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from kerastuner.tuners import RandomSearch
from kerastuner.engine.hyperparameters import HyperParameters
from tensorflow.keras.regularizers import l2
import itertools
#%%
# Load the market + sentiment data

df_vader = pd.read_csv("/Users/malvin/DataspellProjects/MasterThesis/merged_tsla_sentiment_vader.csv")
df_textblob = pd.read_csv("/Users/malvin/DataspellProjects/MasterThesis/merged_tsla_sentiment_textblob.csv")
df_fintwitbert = pd.read_csv("/Users/malvin/DataspellProjects/MasterThesis/merged_tsla_sentiment_fintwitbert
.csv")

#%%
#  Load merged sentiment files for TSLA (stock)
df_vader = pd.read_csv("/Users/malvin/DataspellProjects/MasterThesis/merged_tsla_sentiment_vader.csv")
df_textblob = pd.read_csv("/Users/malvin/DataspellProjects/MasterThesis/merged_tsla_sentiment_textblob.csv")
df_fintwitbert = pd.read_csv("/Users/malvin/DataspellProjects/MasterThesis/merged_tsla_sentiment_fintwitbert
.csv")

#  Process VADER sentiment
df_vader['date'] = pd.to_datetime(df_vader['date'])
df_vader.set_index('date', inplace=True)
tsla_vader = df_vader[['TSLA_Open', 'TSLA_High', 'TSLA_Low', 'TSLA_Close', 'vader_score']].asfreq('B')
tsla_vader['vader_score'] = tsla_vader['vader_score'].fillna(0)

#  Process TextBlob sentiment
df_textblob['date'] = pd.to_datetime(df_textblob['date'])
df_textblob.set_index('date', inplace=True)
tsla_textblob = df_textblob[['TSLA_Open', 'TSLA_High', 'TSLA_Low', 'TSLA_Close', 'textblob_polarity']].
asfreq('B')
tsla_textblob['textblob_polarity'] = tsla_textblob['textblob_polarity'].fillna(0)

#  Process FinTwitBERT sentiment
df_fintwitbert['date'] = pd.to_datetime(df_fintwitbert['date'])
df_fintwitbert.set_index('date', inplace=True)
tsla_fintwitbert = df_fintwitbert[['TSLA_Open', 'TSLA_High', 'TSLA_Low', 'TSLA_Close', 'fintwitbert_weighted
']].asfreq('B')
tsla_fintwitbert['fintwitbert_weighted'] = tsla_fintwitbert['fintwitbert_weighted'].fillna(0)

#%%
tsla_vader
#%%
# Define which columns represent price data
price_cols = ['TSLA_Open', 'TSLA_High', 'TSLA_Low', 'TSLA_Close']

# Drop rows in each sentiment DataFrame where any price column is NaN
```
```

```
 86
 87 tsla_vader = tsla_vader.dropna(subset=price_cols)
 88 tsla_textblob = tsla_textblob.dropna(subset=price_cols)
 89 tsla_fintwitbert = tsla_fintwitbert.dropna(subset=price_cols)
 90 #%%
 91 # Run ADF test to determine trend
 92 def run_adf_test(series, label):
 93     result = adfuller(series.dropna())
 94     print(f"\n--- ADF Test for {label} ---")
 95     print(f"ADF Statistic: {result[0]:.4f}")
 96     print(f"p-value: {result[1]:.4f}")
 97     if result[1] < 0.05:
 98         print("   Series is likely stationary (no differencing needed).")
 99     else:
100         print("⚠️ Series is likely non-stationary (consider differencing).")
101
102 # Run ADF tests
103 run_adf_test(tsla_vader['TSLA_Close'], 'TSLA_Close')
104 #%%
105 # ARIMA closing Price only
106 y_tsla = tsla_vader['TSLA_Close']
107 y_tsla.index = pd.to_datetime(y_tsla.index)
108
109 #  80/20 train/test split
110 split_idx = int(0.8 * len(y_tsla))
111 y_tsla_train = y_tsla[:split_idx]
112 y_tsla_test  = y_tsla[split_idx:]
113
114 #  Train ARIMA model
115 auto_model = auto_arima(
116     y_tsla_train,
117     start_p=1, start_q=1,
118     max_p=5, max_q=5,
119     d=1, seasonal=False,
120     stepwise=True, trace=True
121 )
122
123 print(f"\nBest ARIMA order: {auto_model.order}")
124
125 #  Forecast test period
126 forecast_auto = auto_model.predict(n_periods=len(y_tsla_test))
127
128 #  Evaluate performance
129 # Forecast on training set
130 forecast_train = auto_model.predict_in_sample(start=1, end=split_idx - 1)
131 y_tsla_train_trimmed = y_tsla_train[1:]
132
133 # Training metrics
134 rmse_train = mean_squared_error(y_tsla_train_trimmed, forecast_train, squared=False)
135 mae_train = mean_absolute_error(y_tsla_train_trimmed, forecast_train)
136 r2_train = r2_score(y_tsla_train_trimmed, forecast_train)
137
138 print("\n   ARIMA Performance (Train Set):")
139 print(f"  RMSE: {rmse_train:.4f}")
140 print(f"  MAE : {mae_train:.4f}")
141 print(f"  R²  : {r2_train:.4f}")
142
143 # Test metrics
144 rmse_test = mean_squared_error(y_tsla_test, forecast_auto, squared=False)
145 mae_test = mean_absolute_error(y_tsla_test, forecast_auto)
146 r2_test = r2_score(y_tsla_test, forecast_auto)
147
148 print("\n   ARIMA Performance (Test Set):")
149 print(f"  RMSE: {rmse_test:.4f}")
150 print(f"  MAE : {mae_test:.4f}")
151 print(f"  R²  : {r2_test:.4f}")
152
153 print(f"\nPrice Range (Test Set): Min = {y_tsla_test.min():.2f}, Max = {y_tsla_test.max():.2f}")
154
155 #  Plot forecast vs actual
156 plt.figure(figsize=(14, 6))
157 plt.plot(y_tsla, label='Actual TSLA Close', color='black')
158 plt.plot(y_tsla_test.index, forecast_auto, label='ARIMA Forecast (Auto)', color='blue', linestyle='--')
159 plt.axvline(y_tsla.index[split_idx], color='gray', linestyle='--', label='Train/Test Split')
160 plt.title('ARIMA Forecast vs Actual TSLA Price (80/20 Split)')
161 plt.xlabel('Date')
162 plt.ylabel('TSLA Close Price')
163 plt.legend()
164 plt.grid(True)
165 plt.tight_layout()
166 plt.savefig("arima_tsla_forecast.png", dpi=300)
167 plt.show()
168 #%%
169 print(auto_model.summary())
170 #%%
171 # Get residuals from the model (assuming pmdarima)
172 residuals = auto_model.resid()  # For pmdarima's auto_arima
173
174 # Line plot of residuals
```

```
plt.figure(figsize=(12, 4))
plt.plot(residuals)
plt.title("Residuals from ARIMA Model")
plt.xlabel("Time")
plt.ylabel("Residual")
plt.grid(True)
plt.tight_layout()
plt.show()

# Histogram with KDE
plt.figure(figsize=(8, 4))
sns.histplot(residuals, bins=30, kde=True)
plt.title("Distribution of Residuals")
plt.xlabel("Residual")
plt.tight_layout()
plt.show()

# Autocorrelation plot of residuals
sm.graphics.tsa.plot_acf(residuals.dropna(), lags=40)
plt.title("Autocorrelation of Residuals")
plt.tight_layout()
plt.show()
#%%
# Now using ARIMAX with the Vader Sentiment Score as an additional feature
X_vader = tsla_vader[['vader_score']]
X_vader_train = X_vader[:split_idx]
X_vader_test  = X_vader[split_idx:]

# Fitting ARIMAX with VADER
model_arimax_vader = ARIMA(endog=y_tsla_train, exog=X_vader_train, order=(0, 1, 0))
model_arimax_vader_fit = model_arimax_vader.fit()

# Forecasting
forecast_arimax_vader = model_arimax_vader_fit.forecast(steps=len(y_tsla_test), exog=X_vader_test)
#%%
plt.figure(figsize=(14, 6))
plt.plot(y_tsla, label='Actual TSLA Close', color='black')
plt.plot(y_tsla_test.index, forecast_arimax_vader, label='ARIMAX Forecast (VADER)', color='green',
    linestyle='--')
plt.axvline(y_tsla.index[split_idx], color='gray', linestyle='--', label='Train/Test Split')
plt.title('ARIMAX Forecast vs Actual (w/ VADER Sentiment)')
plt.xlabel('Date')
plt.ylabel('TSLA Close Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
#%%
#  Forecast on Training Set (ARIMAX VADER)
forecast_train_vader = model_arimax_vader_fit.predict(
    start=1, end=split_idx - 1, exog=X_vader_train
)

#  Trim y_tsla_train to match forecast length (due to d=1 differencing)
y_tsla_train_trimmed = y_tsla_train[1:]

#  Evaluate Training Set Performance
rmse_train_vader = mean_squared_error(y_tsla_train_trimmed, forecast_train_vader, squared=False)
mae_train_vader = mean_absolute_error(y_tsla_train_trimmed, forecast_train_vader)
r2_train_vader = r2_score(y_tsla_train_trimmed, forecast_train_vader)

print("   ARIMAX (VADER) Performance (Train Set):")
print(f"  RMSE: {rmse_train_vader:.4f}")
print(f"  MAE : {mae_train_vader:.4f}")
print(f"  R²  : {r2_train_vader:.4f}")

#  Forecast on Test Set (already computed as forecast_arimax_vader)
rmse_test_vader = mean_squared_error(y_tsla_test, forecast_arimax_vader, squared=False)
mae_test_vader = mean_absolute_error(y_tsla_test, forecast_arimax_vader)
r2_test_vader = r2_score(y_tsla_test, forecast_arimax_vader)

print("\n   ARIMAX (VADER) Performance (Test Set):")
print(f"  RMSE: {rmse_test_vader:.4f}")
print(f"  MAE : {mae_test_vader:.4f}")
print(f"  R²  : {r2_test_vader:.4f}")

#  Context: Price Range in Test Set
print(f"\nPrice Range (Test Set): Min = {y_tsla_test.min():.2f}, Max = {y_tsla_test.max():.2f}")
#%%
# ARIMAX for textblob
X_textblob = tsla_textblob[['textblob_polarity']]
X_textblob_train = X_textblob[:split_idx]
X_textblob_test  = X_textblob[split_idx:]

# Fit ARIMAX(0,1,0) model
model_arimax_textblob = ARIMA(endog=y_tsla_train, exog=X_textblob_train, order=(0, 1, 0))
model_arimax_textblob_fit = model_arimax_textblob.fit()

# Forecast using TextBlob sentiment
```

```
forecast_arimax_textblob = model_arimax_textblob_fit.forecast(steps=len(y_tsla_test), exog=X_textblob_test)
#%%
#plotting

plt.figure(figsize=(14, 6))
plt.plot(y_tsla, label='Actual TSLA Close', color='black')
plt.plot(y_tsla_test.index, forecast_arimax_textblob, label='ARIMAX Forecast (TextBlob)', color='orange', linestyle='--')
plt.axvline(y_tsla.index[split_idx], color='gray', linestyle='--', label='Train/Test Split')
plt.title('ARIMAX Forecast vs Actual (w/ TextBlob Sentiment)')
plt.xlabel('Date')
plt.ylabel('TSLA Close Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig("arimax_tsla_textblob.png", dpi=300)
plt.show()
#%%
#  Forecast on Training Set (TextBlob)
forecast_train_textblob = model_arimax_textblob_fit.predict(start=1, end=split_idx-1, exog=X_textblob_train)
y_tsla_train_trimmed_textblob = y_tsla_train[1:]

#  Evaluate Training Set Performance
rmse_train_textblob = mean_squared_error(y_tsla_train_trimmed_textblob, forecast_train_textblob, squared=False)
mae_train_textblob = mean_absolute_error(y_tsla_train_trimmed_textblob, forecast_train_textblob)
r2_train_textblob = r2_score(y_tsla_train_trimmed_textblob, forecast_train_textblob)

print("   ARIMAX (TextBlob) Performance — Train Set:")
print(f"  RMSE: {rmse_train_textblob:.4f}")
print(f"  MAE : {mae_train_textblob:.4f}")
print(f"  R²  : {r2_train_textblob:.4f}")

#  Evaluate Test Set Performance
rmse_test_textblob = mean_squared_error(y_tsla_test, forecast_arimax_textblob, squared=False)
mae_test_textblob = mean_absolute_error(y_tsla_test, forecast_arimax_textblob)
r2_test_textblob = r2_score(y_tsla_test, forecast_arimax_textblob)

print("\n   ARIMAX (TextBlob) Performance — Test Set:")
print(f"  RMSE: {rmse_test_textblob:.4f}")
print(f"  MAE : {mae_test_textblob:.4f}")
print(f"  R²  : {r2_test_textblob:.4f}")

#  Context: Price Range in Test Set
print(f"\nPrice Range (Test Set): Min = {y_tsla_test.min():.2f}, Max = {y_tsla_test.max():.2f}")
#%%
# Print model summary
print("\n   Model Summary (TextBlob):")
print(model_arimax_textblob_fit.summary())
#%%
# Define and split FinTwitBERT sentiment
X_fintwitbert = tsla_fintwitbert[['fintwitbert_weighted']]
X_fintwitbert_train = X_fintwitbert[:split_idx]
X_fintwitbert_test  = X_fintwitbert[split_idx:]

# Fit ARIMAX(0,1,0) model
model_arimax_fintwitbert = ARIMA(endog=y_tsla_train, exog=X_fintwitbert_train, order=(0, 1, 0))
model_arimax_fintwitbert_fit = model_arimax_fintwitbert.fit()

# Forecast using FinTwitBERT sentiment
forecast_arimax_fintwitbert = model_arimax_fintwitbert_fit.forecast(
    steps=len(y_tsla_test),
    exog=X_fintwitbert_test
)
#%%
# Plotting
plt.figure(figsize=(14, 6))
plt.plot(y_tsla, label='Actual TSLA Close', color='black')
plt.plot(y_tsla_test.index, forecast_arimax_fintwitbert, label='ARIMAX Forecast (FinTwitBERT)', color='purple', linestyle='--')
plt.axvline(y_tsla.index[split_idx], color='gray', linestyle='--', label='Train/Test Split')
plt.title('ARIMAX Forecast vs Actual (w/ FinTwitBERT Sentiment)')
plt.xlabel('Date')
plt.ylabel('TSLA Close Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
#%%
#  Forecast on Training Set (FinTwitBERT)
forecast_train_fintwitbert = model_arimax_fintwitbert_fit.predict(start=1, end=split_idx-1, exog=X_fintwitbert_train)
y_tsla_train_trimmed_fintwitbert = y_tsla_train[1:]

#  Evaluate Training Set Performance
rmse_train_fintwitbert = mean_squared_error(y_tsla_train_trimmed_fintwitbert, forecast_train_fintwitbert, squared=False)
mae_train_fintwitbert = mean_absolute_error(y_tsla_train_trimmed_fintwitbert, forecast_train_fintwitbert)
```

```
r2_train_fintwitbert = r2_score(y_tsla_train_trimmed_fintwitbert, forecast_train_fintwitbert)

print("   ARIMAX (FinTwitBERT) Performance — Train Set:")
print(f"  RMSE: {rmse_train_fintwitbert:.4f}")
print(f"  MAE : {mae_train_fintwitbert:.4f}")
print(f"  R²  : {r2_train_fintwitbert:.4f}")

#  Evaluate Test Set Performance
rmse_test_fintwitbert = mean_squared_error(y_tsla_test, forecast_arimax_fintwitbert, squared=False)
mae_test_fintwitbert = mean_absolute_error(y_tsla_test, forecast_arimax_fintwitbert)
r2_test_fintwitbert = r2_score(y_tsla_test, forecast_arimax_fintwitbert)

print("\n   ARIMAX (FinTwitBERT) Performance — Test Set:")
print(f"  RMSE: {rmse_test_fintwitbert:.4f}")
print(f"  MAE : {mae_test_fintwitbert:.4f}")
print(f"  R²  : {r2_test_fintwitbert:.4f}")

#  Context: Price Range in Test Set
print(f"\nPrice Range (Test Set): Min = {y_tsla_test.min():.2f}, Max = {y_tsla_test.max():.2f}")

# Print model summary
print("\n   Model Summary (FinTwitBERT):")
print(model_arimax_fintwitbert_fit.summary())
#%%
# PACF Plot to see how many lags to include for the XGBoost Algorithm setup

plot_pacf(tsla_vader['TSLA_Close'], lags=20, method='ywm')
plt.title('Partial Autocorrelation (PACF) of $TSLA Closing Price')
plt.xlabel('Lag')
plt.ylabel('Partial Autocorrelation')
plt.grid(True)
plt.tight_layout()
plt.savefig("pacf_tsla_closing_price.png", dpi=300, bbox_inches='tight')
plt.show()
#%%
# Create base DataFrame from tsla_vader
df_xgb_base = tsla_vader.copy()

# Lagged features
df_xgb_base['lag1'] = df_xgb_base['TSLA_Close'].shift(1)

# Target: next day's close
df_xgb_base['target'] = df_xgb_base['TSLA_Close'].shift(-1)

# Drop rows with NaNs from shift
df_xgb_base.dropna(inplace=True)

# Feature and target selection
X_xgb_base = df_xgb_base[['lag1']]
y_xgb_base = df_xgb_base['target']
#%%
# Time-aware split
split_idx_xgb = int(0.8 * len(df_xgb_base))

X_xgb_base_train = X_xgb_base[:split_idx_xgb]
X_xgb_base_test  = X_xgb_base[split_idx_xgb:]

y_xgb_base_train = y_xgb_base[:split_idx_xgb]
y_xgb_base_test  = y_xgb_base[split_idx_xgb:]
#%%
# Train the model
xgb_model_base = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_model_base.fit(X_xgb_base_train, y_xgb_base_train)

# Train predictions
preds_train = xgb_model_base.predict(X_xgb_base_train)

# Test predictions
preds_test = xgb_model_base.predict(X_xgb_base_test)

# Train set metrics
rmse_train = mean_squared_error(y_xgb_base_train, preds_train, squared=False)
mae_train = mean_absolute_error(y_xgb_base_train, preds_train)
r2_train = r2_score(y_xgb_base_train, preds_train)

# Test set metrics
rmse_test = mean_squared_error(y_xgb_base_test, preds_test, squared=False)
mae_test = mean_absolute_error(y_xgb_base_test, preds_test)
r2_test = r2_score(y_xgb_base_test, preds_test)

# Output results
print("   XGBoost Performance price only (Train Set):")
print(f"  RMSE: {rmse_train:.4f}")
print(f"  MAE : {mae_train:.4f}")
print(f"  R²  : {r2_train:.4f}")

print("\n   XGBoost Performance price only (Test Set):")
print(f"  RMSE: {rmse_test:.4f}")
print(f"  MAE : {mae_test:.4f}")
```

```
print(f"  R²  : {r2_test:.4f}")

plt.figure(figsize=(12,6))
plt.plot(y_xgb_base_test.values, label='Actual')
plt.plot(preds_test, label='Predicted', linestyle='--')
plt.title("XGBoost Prediction vs Actual (Base)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
#%%
# Build merged df with the sentiment scores from all 3 models and lagged values

# Start from VADER base
df_xgb_all_sent = tsla_vader[['TSLA_Open', 'TSLA_High', 'TSLA_Low', 'TSLA_Close', 'vader_score']].copy()

# Add sentiment features
df_xgb_all_sent = df_xgb_all_sent.join(tsla_textblob[['textblob_polarity']])
df_xgb_all_sent = df_xgb_all_sent.join(tsla_fintwitbert[['fintwitbert_weighted']])

# Create lag features
df_xgb_all_sent['lag1'] = df_xgb_all_sent['TSLA_Close'].shift(1)

# Create financial features for later use
df_xgb_all_sent['daily_return']   = df_xgb_all_sent['TSLA_Close'].pct_change()
df_xgb_all_sent['hl_spread']      = df_xgb_all_sent['TSLA_High'] - df_xgb_all_sent['TSLA_Low']
df_xgb_all_sent['lagged_return']  = df_xgb_all_sent['daily_return'].shift(1)

# Target: next day's closing price
df_xgb_all_sent['target'] = df_xgb_all_sent['TSLA_Close'].shift(-1)

# Drop rows with any missing values from shifting or pct_change
df_xgb_all_sent.dropna(inplace=True)

#%%
# Define all X and y
X_xgb_all_sent = df_xgb_all_sent[['lag1', 'vader_score', 'textblob_polarity', 'fintwitbert_weighted']]
y_xgb_all_sent = df_xgb_all_sent['target']

# Time-aware split
split_idx_xgb_all_sent = int(0.8 * len(df_xgb_all_sent))

X_xgb_all_sent_train = X_xgb_all_sent[:split_idx_xgb_all_sent]
X_xgb_all_sent_test  = X_xgb_all_sent[split_idx_xgb_all_sent:]

y_xgb_all_sent_train = y_xgb_all_sent[:split_idx_xgb_all_sent]
y_xgb_all_sent_test  = y_xgb_all_sent[split_idx_xgb_all_sent:]
#%%
# Use only lag1, and vader_score
features_vader = ['lag1', 'vader_score']

# Train model
xgb_model_vader_only = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_model_vader_only.fit(X_xgb_all_sent_train[features_vader], y_xgb_all_sent_train)

# Predict
y_pred_train_vader = xgb_model_vader_only.predict(X_xgb_all_sent_train[features_vader])
y_pred_test_vader  = xgb_model_vader_only.predict(X_xgb_all_sent_test[features_vader])

# Train set metrics
rmse_train = mean_squared_error(y_xgb_all_sent_train, y_pred_train_vader, squared=False)
mae_train = mean_absolute_error(y_xgb_all_sent_train, y_pred_train_vader)
r2_train = r2_score(y_xgb_all_sent_train, y_pred_train_vader)

# Test set metrics
rmse_test = mean_squared_error(y_xgb_all_sent_test, y_pred_test_vader, squared=False)
mae_test = mean_absolute_error(y_xgb_all_sent_test, y_pred_test_vader)
r2_test = r2_score(y_xgb_all_sent_test, y_pred_test_vader)

# Output results
print("   XGBoost (Lag + VADER) - Train Set:")
print(f"  RMSE: {rmse_train:.4f}")
print(f"  MAE : {mae_train:.4f}")
print(f"  R²  : {r2_train:.4f}")

print("\n   XGBoost (Lag + VADER) - Test Set:")
print(f"  RMSE: {rmse_test:.4f}")
print(f"  MAE : {mae_test:.4f}")
print(f"  R²  : {r2_test:.4f}")

# Plot
plt.figure(figsize=(14, 6))
plt.plot(y_xgb_all_sent_test.index, y_xgb_all_sent_test, label='Actual TSLA Close', color='black')
plt.plot(y_xgb_all_sent_test.index, y_pred_test_vader, label='XGBoost Predicted (VADER Only)', color='blue'
    , linestyle='--')
plt.axvline(y_xgb_all_sent_test.index[0], color='gray', linestyle='--', label='Train/Test Split')
plt.title('XGBoost Forecast vs Actual (Lag + VADER)')
plt.xlabel('Date')
plt.ylabel('$TSLA Close Price')
```

```
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
#%%
# Use only lag1, and textblob_polarity
features_textblob = ['lag1', 'textblob_polarity']

# Train
xgb_model_textblob_only = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_model_textblob_only.fit(X_xgb_all_sent_train[features_textblob], y_xgb_all_sent_train)

# Predict on Train and Test Sets
y_xgb_pred_textblob_train = xgb_model_textblob_only.predict(X_xgb_all_sent_train[features_textblob])
y_xgb_pred_textblob_test = xgb_model_textblob_only.predict(X_xgb_all_sent_test[features_textblob])

# Evaluate Training Set
rmse_train_tb = mean_squared_error(y_xgb_all_sent_train, y_xgb_pred_textblob_train, squared=False)
mae_train_tb = mean_absolute_error(y_xgb_all_sent_train, y_xgb_pred_textblob_train)
r2_train_tb = r2_score(y_xgb_all_sent_train, y_xgb_pred_textblob_train)

# Evaluate Test Set
rmse_test_tb = mean_squared_error(y_xgb_all_sent_test, y_xgb_pred_textblob_test, squared=False)
mae_test_tb = mean_absolute_error(y_xgb_all_sent_test, y_xgb_pred_textblob_test)
r2_test_tb = r2_score(y_xgb_all_sent_test, y_xgb_pred_textblob_test)

# Output Results
print("   XGBoost (Lag + TextBlob) - Train Set:")
print(f"  RMSE: {rmse_train_tb:.4f}")
print(f"  MAE : {mae_train_tb:.4f}")
print(f"  R²  : {r2_train_tb:.4f}")

print("\n   XGBoost (Lag + TextBlob) - Test Set:")
print(f"  RMSE: {rmse_test_tb:.4f}")
print(f"  MAE : {mae_test_tb:.4f}")
print(f"  R²  : {r2_test_tb:.4f}")

# Plot
plt.figure(figsize=(14, 6))
plt.plot(y_xgb_all_sent_test.index, y_xgb_all_sent_test, label='Actual TSLA Close', color='black')
plt.plot(y_xgb_all_sent_test.index, y_xgb_pred_textblob_test, label='XGBoost Predicted (TextBlob Only)',
    color='orange', linestyle='--')
plt.axvline(y_xgb_all_sent_test.index[0], color='gray', linestyle='--', label='Train/Test Split')
plt.title('XGBoost Forecast vs Actual (Lag + TextBlob)')
plt.xlabel('Date')
plt.ylabel('$TSLA Close Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
#%%
# Use only lag1, lag2, and fintwitbert_weighted
features_fintwitbert = ['lag1', 'fintwitbert_weighted']

# Train
xgb_model_fintwitbert_only = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_model_fintwitbert_only.fit(X_xgb_all_sent_train[features_fintwitbert], y_xgb_all_sent_train)

# Predict on Train and Test Sets
y_xgb_pred_fintwitbert_train = xgb_model_fintwitbert_only.predict(X_xgb_all_sent_train[features_fintwitbert
    ])
y_xgb_pred_fintwitbert_test = xgb_model_fintwitbert_only.predict(X_xgb_all_sent_test[features_fintwitbert])

# Evaluate Training Set
rmse_train_ftb = mean_squared_error(y_xgb_all_sent_train, y_xgb_pred_fintwitbert_train, squared=False)
mae_train_ftb = mean_absolute_error(y_xgb_all_sent_train, y_xgb_pred_fintwitbert_train)
r2_train_ftb = r2_score(y_xgb_all_sent_train, y_xgb_pred_fintwitbert_train)

# Evaluate Test Set
rmse_test_ftb = mean_squared_error(y_xgb_all_sent_test, y_xgb_pred_fintwitbert_test, squared=False)
mae_test_ftb = mean_absolute_error(y_xgb_all_sent_test, y_xgb_pred_fintwitbert_test)
r2_test_ftb = r2_score(y_xgb_all_sent_test, y_xgb_pred_fintwitbert_test)

# Output Results
print("   XGBoost (Lag + FinTwitBERT) - Train Set:")
print(f"  RMSE: {rmse_train_ftb:.4f}")
print(f"  MAE : {mae_train_ftb:.4f}")
print(f"  R²  : {r2_train_ftb:.4f}")

print("\n   XGBoost (Lag + FinTwitBERT) - Test Set:")
print(f"  RMSE: {rmse_test_ftb:.4f}")
print(f"  MAE : {mae_test_ftb:.4f}")
print(f"  R²  : {r2_test_ftb:.4f}")

# Plot
plt.figure(figsize=(14, 6))
plt.plot(y_xgb_all_sent_test.index, y_xgb_all_sent_test, label='Actual TSLA Close', color='black')
plt.plot(y_xgb_all_sent_test.index, y_xgb_pred_fintwitbert_test, label='XGBoost Predicted (FinTwitBERT Only
    )', color='purple', linestyle='--')
```

```
plt.axvline(y_xgb_all_sent_test.index[0], color='gray', linestyle='--', label='Train/Test Split')
plt.title('XGBoost Forecast vs Actual (Lag + FinTwitBERT)')
plt.xlabel('Date')
plt.ylabel('TSLA Close Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
#%%
# Use all features: lagged prices + all sentiment scores
features_all_sent = ['lag1', 'vader_score', 'textblob_polarity', 'fintwitbert_weighted']

# Train
xgb_model_all_sent = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_model_all_sent.fit(X_xgb_all_sent_train[features_all_sent], y_xgb_all_sent_train)

# Predict on Train and Test Sets
y_xgb_pred_all_sent_train = xgb_model_all_sent.predict(X_xgb_all_sent_train[features_all_sent])
y_xgb_pred_all_sent_test = xgb_model_all_sent.predict(X_xgb_all_sent_test[features_all_sent])

# Evaluate Training Set
rmse_train_all = mean_squared_error(y_xgb_all_sent_train, y_xgb_pred_all_sent_train, squared=False)
mae_train_all = mean_absolute_error(y_xgb_all_sent_train, y_xgb_pred_all_sent_train)
r2_train_all = r2_score(y_xgb_all_sent_train, y_xgb_pred_all_sent_train)

# Evaluate Test Set
rmse_test_all = mean_squared_error(y_xgb_all_sent_test, y_xgb_pred_all_sent_test, squared=False)
mae_test_all = mean_absolute_error(y_xgb_all_sent_test, y_xgb_pred_all_sent_test)
r2_test_all = r2_score(y_xgb_all_sent_test, y_xgb_pred_all_sent_test)

# Output Results
print("   XGBoost (Lag + All Sentiment) - Train Set:")
print(f"  RMSE: {rmse_train_all:.4f}")
print(f"  MAE : {mae_train_all:.4f}")
print(f"  R²  : {r2_train_all:.4f}")

print("\n   XGBoost (Lag + All Sentiment) - Test Set:")
print(f"  RMSE: {rmse_test_all:.4f}")
print(f"  MAE : {mae_test_all:.4f}")
print(f"  R²  : {r2_test_all:.4f}")

# Plot
plt.figure(figsize=(14, 6))
plt.plot(y_xgb_all_sent_test.index, y_xgb_all_sent_test, label='Actual TSLA Close', color='black')
plt.plot(y_xgb_all_sent_test.index, y_xgb_pred_all_sent_test, label='XGBoost Predicted (All Sentiment)',
    color='purple', linestyle='--')
plt.axvline(y_xgb_all_sent_test.index[0], color='gray', linestyle='--', label='Train/Test Split')
plt.title('XGBoost Forecast vs Actual (Lag + All Sentiment)')
plt.xlabel('Date')
plt.ylabel('TSLA Close Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
#%%
# Extract feature importances and feature names
importances = xgb_model_all_sent.feature_importances_
feature_names = ['lag1', 'vader_score', 'textblob_polarity', 'fintwitbert_weighted']

# Create a DataFrame for plotting
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
importance_df.sort_values(by='Importance', ascending=True, inplace=True)

# Plot
plt.figure(figsize=(8, 5))
sns.barplot(x='Importance', y='Feature', data=importance_df, palette='viridis')
plt.title('XGBoost Feature Importances (All Sentiment)')
plt.tight_layout()
plt.savefig("xgboost_tsla_feature_importance.png", dpi=300)
plt.show()
#%%
# Create time series splitter
tscv = TimeSeriesSplit(n_splits=5)

# Define the model again
xgb_all_sent = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)

# Use RMSE as scoring metric (scikit-learn minimizes, so we use neg_root_mean_squared_error)
rmse_scorer = make_scorer(mean_squared_error, squared=False)

# Perform CV on full data
cv_rmse_scores = cross_val_score(
    xgb_all_sent,
    X_xgb_all_sent[features_all_sent],
    y_xgb_all_sent,
    cv=tscv,
    scoring=rmse_scorer
)

```

```
# Print mean and std
print("   TimeSeries Cross-Validation (RMSE):")
print(f"  Mean RMSE: {np.mean(cv_rmse_scores):.4f}")
print(f"  Std  RMSE: {np.std(cv_rmse_scores):.4f}")
#%%
# attempt at Hyperparameter Tuning

param_grid = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [2, 3, 4],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

tscv = TimeSeriesSplit(n_splits=5)
xgb = XGBRegressor(random_state=42)

grid = GridSearchCV(
    estimator=xgb,
    param_grid=param_grid,
    cv=tscv,
    scoring='neg_root_mean_squared_error',
    verbose=1,
    n_jobs=-1
)

grid.fit(X_xgb_all_sent, y_xgb_all_sent)
print("Best Params:", grid.best_params_)
print("Best RMSE:", -grid.best_score_)
#%%
# Define best hyperparameters from GridSearchCV
best_params = grid.best_params_

# Features used in best model
features_all_sentiment_only = [
    'lag1',
    'vader_score', 'textblob_polarity', 'fintwitbert_weighted'
]

# Train the best model on full training set
xgb_model_best = XGBRegressor(**best_params, random_state=42)
xgb_model_best.fit(X_xgb_all_sent_train[features_all_sentiment_only], y_xgb_all_sent_train)

# Predict on train and test sets
y_pred_train_best = xgb_model_best.predict(X_xgb_all_sent_train[features_all_sentiment_only])
y_pred_test_best = xgb_model_best.predict(X_xgb_all_sent_test[features_all_sentiment_only])

# Evaluate on train set
rmse_train = mean_squared_error(y_xgb_all_sent_train, y_pred_train_best, squared=False)
mae_train = mean_absolute_error(y_xgb_all_sent_train, y_pred_train_best)
r2_train = r2_score(y_xgb_all_sent_train, y_pred_train_best)

# Evaluate on test set
rmse_test = mean_squared_error(y_xgb_all_sent_test, y_pred_test_best, squared=False)
mae_test = mean_absolute_error(y_xgb_all_sent_test, y_pred_test_best)
r2_test = r2_score(y_xgb_all_sent_test, y_pred_test_best)

# Print results
print("   XGBoost (Best Tuned Model: Lag + All Sentiment) — Train Set:")
print(f"  RMSE: {rmse_train:.4f}")
print(f"  MAE : {mae_train:.4f}")
print(f"  R²  : {r2_train:.4f}")

print("\n   XGBoost (Best Tuned Model: Lag + All Sentiment) — Test Set:")
print(f"  RMSE: {rmse_test:.4f}")
print(f"  MAE : {mae_test:.4f}")
print(f"  R²  : {r2_test:.4f}")

# Plot forecast vs actual (Test Set)
plt.figure(figsize=(14, 6))
plt.plot(y_xgb_all_sent_test.index, y_xgb_all_sent_test, label='Actual $TSLA Close', color='black')
plt.plot(y_xgb_all_sent_test.index, y_pred_test_best, label='XGBoost Forecast (Lag + Sentiment)', color='
    blue', linestyle='--')
plt.axvline(y_xgb_all_sent_test.index[0], color='gray', linestyle='--', label='Train/Test Split')
plt.title("XGBoost Forecast vs Actual $TSLA Close (Best Tuned Model)")
plt.xlabel("Date")
plt.ylabel("$TSLA Close Price")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig("xgboost_tsla_best_forecast.png", dpi=300)
plt.show()
#%%
# 1. Residuals over time
residuals = y_xgb_all_sent_test - y_xgb_pred_all_sent_test

plt.figure(figsize=(12, 4))
plt.plot(y_xgb_all_sent_test.index, residuals, marker='o', linestyle='-', color='red')
```

```
plt.axhline(0, color='gray', linestyle='--')
plt.title("Residuals Over Time")
plt.xlabel("Date")
plt.ylabel("Residual (Actual - Predicted)")
plt.grid(True)
plt.tight_layout()
plt.show()

# 2. Histogram of residuals
plt.figure(figsize=(8, 4))
sns.histplot(residuals, bins=20, kde=True, color='purple')
plt.title("Histogram of Residuals")
plt.xlabel("Residual Value")
plt.ylabel("Frequency")
plt.grid(True)
plt.tight_layout()
plt.show()

# 3. Q-Q plot for normality

sm.qqplot(residuals, line='s')
plt.title("Q-Q Plot of Residuals")
plt.tight_layout()
plt.show()

# 4. Autocorrelation plot (ACF)
plot_acf(residuals, lags=20)
plt.title("Autocorrelation of Residuals")
plt.tight_layout()
plt.show()
#%%
# Add financial features to best XGBoost Model

# Use all features: lagged prices + financial + all sentiment scores
features_all_sent_financial = [
    'lag1',
    'vader_score', 'textblob_polarity', 'fintwitbert_weighted',
    'daily_return', 'hl_spread', 'lagged_return'
]

# Redefine full X and y using updated df
X_xgb_all_sent = df_xgb_all_sent[features_all_sent_financial]
y_xgb_all_sent = df_xgb_all_sent['target']

# Redo the train-test split
split_idx = int(0.8 * len(df_xgb_all_sent))
X_xgb_all_sent_train = X_xgb_all_sent[:split_idx]
X_xgb_all_sent_test  = X_xgb_all_sent[split_idx:]

y_xgb_all_sent_train = y_xgb_all_sent[:split_idx]
y_xgb_all_sent_test  = y_xgb_all_sent[split_idx:]
# Train
xgb_model_all_sent_fin = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_model_all_sent_fin.fit(X_xgb_all_sent_train[features_all_sent_financial], y_xgb_all_sent_train)

# Predict on Train and Test Sets
y_pred_train_fin = xgb_model_all_sent_fin.predict(X_xgb_all_sent_train[features_all_sent_financial])
y_pred_test_fin = xgb_model_all_sent_fin.predict(X_xgb_all_sent_test[features_all_sent_financial])

# Evaluate Training Set
rmse_train = mean_squared_error(y_xgb_all_sent_train, y_pred_train_fin, squared=False)
mae_train = mean_absolute_error(y_xgb_all_sent_train, y_pred_train_fin)
r2_train = r2_score(y_xgb_all_sent_train, y_pred_train_fin)

# Evaluate Test Set
rmse_test = mean_squared_error(y_xgb_all_sent_test, y_pred_test_fin, squared=False)
mae_test = mean_absolute_error(y_xgb_all_sent_test, y_pred_test_fin)
r2_test = r2_score(y_xgb_all_sent_test, y_pred_test_fin)

# Output Results
print("   XGBoost (All Sentiment + Financial) - Train Set:")
print(f"  RMSE: {rmse_train:.4f}")
print(f"  MAE : {mae_train:.4f}")
print(f"  R²  : {r2_train:.4f}")

print("\n   XGBoost (All Sentiment + Financial) - Test Set:")
print(f"  RMSE: {rmse_test:.4f}")
print(f"  MAE : {mae_test:.4f}")
print(f"  R²  : {r2_test:.4f}")

# Plot
plt.figure(figsize=(14, 6))
plt.plot(y_xgb_all_sent_test.index, y_xgb_all_sent_test, label='Actual $TSLA Close', color='black')
plt.plot(y_xgb_all_sent_test.index, y_pred_test_fin, label='XGBoost Predicted (All Sentiment + Financial)'
    , color='green', linestyle='--')
plt.axvline(y_xgb_all_sent_test.index[0], color='gray', linestyle='--', label='Train/Test Split')
plt.title('XGBoost Forecast vs Actual (Lag + All Sentiment + Financial Features)')
plt.xlabel('Date')
plt.ylabel('TSLA Close Price')
```

```
873 plt.legend()
874 plt.grid(True)
875 plt.tight_layout()
876 plt.show()
877 #%%
878 # Unified DataFrame for LSTM
879
880 # Start from VADER base
881 df_lstm_all_sent = tsla_vader[['TSLA_Open', 'TSLA_High', 'TSLA_Low', 'TSLA_Close', 'vader_score']].copy()
882
883 # Add TextBlob polarity
884 df_lstm_all_sent = df_lstm_all_sent.join(tsla_textblob[['textblob_polarity']])
885
886 # Add FinTwitBERT weighted score
887 df_lstm_all_sent = df_lstm_all_sent.join(tsla_fintwitbert[['fintwitbert_weighted']])
888
889 # Lagged price features
890 df_lstm_all_sent['lag1'] = df_lstm_all_sent['TSLA_Close'].shift(1)
891
892 # Target: next day's closing price
893 df_lstm_all_sent['target'] = df_lstm_all_sent['TSLA_Close'].shift(-1)
894
895 # Drop any rows with NaNs
896 df_lstm_all_sent.dropna(inplace=True)
897
898 # Preview
899 df_lstm_all_sent
900
901 #%%
902 #  Full LSTM Pipeline (Modular + Train/Test Evaluation)
903
904 # Baseline Model
905
906 # Choose input features
907 selected_features = ['lag1']
908
909 # Time-aware train/test split
910 split_idx = int(0.8 * len(df_lstm_all_sent))
911 df_train = df_lstm_all_sent[:split_idx]
912 df_test = df_lstm_all_sent[split_idx:]
913
914 # Scale features and target separately
915 scaler_X = MinMaxScaler()
916 scaler_y = MinMaxScaler()
917
918 X_train = scaler_X.fit_transform(df_train[selected_features])
919 y_train = scaler_y.fit_transform(df_train[['target']])
920
921 X_test = scaler_X.transform(df_test[selected_features])
922 y_test = scaler_y.transform(df_test[['target']])
923
924 # Reshape for LSTM: (samples, timesteps, features)
925 X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
926 X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))
927
928 # Define and compile model
929 model = Sequential()
930 model.add(LSTM(units=50, input_shape=(X_train.shape[1], X_train.shape[2])))
931 model.add(Dense(1))
932 model.compile(optimizer='adam', loss='mean_squared_error')
933
934 # Train model
935 history = model.fit(
936     X_train, y_train,
937     epochs=50,
938     batch_size=16,
939     validation_data=(X_test, y_test),
940     verbose=1,
941     shuffle=False
942 )
943
944 # Predict Train + Test
945 y_train_pred_scaled = model.predict(X_train)
946 y_test_pred_scaled = model.predict(X_test)
947
948 y_train_pred = scaler_y.inverse_transform(y_train_pred_scaled)
949 y_test_pred = scaler_y.inverse_transform(y_test_pred_scaled)
950
951 y_train_true = scaler_y.inverse_transform(y_train)
952 y_test_true = scaler_y.inverse_transform(y_test)
953
954 # Evaluate Train
955 rmse_train = mean_squared_error(y_train_true, y_train_pred, squared=False)
956 mae_train = mean_absolute_error(y_train_true, y_train_pred)
957 r2_train = r2_score(y_train_true, y_train_pred)
958
959 # Evaluate Test
960 rmse_test = mean_squared_error(y_test_true, y_test_pred, squared=False)
961 mae_test = mean_absolute_error(y_test_true, y_test_pred)
```

```
r2_test = r2_score(y_test_true, y_test_pred)

# Print Results
print(f"\n   LSTM Performance ({'+'.join(selected_features)}) — Train:")
print(f"  RMSE: {rmse_train:.4f}")
print(f"  MAE : {mae_train:.4f}")
print(f"  R²  : {r2_train:.4f}")

print(f"\n   LSTM Performance ({'+'.join(selected_features)}) — Test:")
print(f"  RMSE: {rmse_test:.4f}")
print(f"  MAE : {mae_test:.4f}")
print(f"  R²  : {r2_test:.4f}")

# Plot Predictions vs. Actual (Test Set)
date_index = df_test.index
plt.figure(figsize=(14, 6))
plt.plot(date_index, y_test_true, label='Actual TSLA Close', color='black')
plt.plot(date_index, y_test_pred, label='LSTM Predicted', linestyle='--', color='blue')
plt.title(f"LSTM Forecast vs Actual - Features: {selected_features}")
plt.xlabel("Date")
plt.ylabel("$TSLA Close Price")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

#%%
# create loop to quickly run all combinations we used before

feature_sets = [
    ['lag1', 'vader_score'],
    ['lag1', 'textblob_polarity'],
    ['lag1', 'fintwitbert_weighted'],
    ['lag1', 'vader_score', 'textblob_polarity', 'fintwitbert_weighted']
]


for selected_features in feature_sets:
    #  Split
    split_idx = int(0.8 * len(df_lstm_all_sent))
    df_train = df_lstm_all_sent[:split_idx]
    df_test = df_lstm_all_sent[split_idx:]

    #  Scale
    scaler_X = MinMaxScaler()
    scaler_y = MinMaxScaler()

    X_train = scaler_X.fit_transform(df_train[selected_features])
    y_train = scaler_y.fit_transform(df_train[['target']])

    X_test = scaler_X.transform(df_test[selected_features])
    y_test = scaler_y.transform(df_test[['target']])

    #  Reshape
    X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
    X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

    #  Define Model
    model = Sequential()
    model.add(LSTM(units=50, input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mean_squared_error')

    #  Fit
    model.fit(
        X_train, y_train,
        epochs=50,
        batch_size=16,
        validation_data=(X_test, y_test),
        verbose=0,
        shuffle=False
    )

    #  Predict
    y_train_pred = scaler_y.inverse_transform(model.predict(X_train))
    y_test_pred  = scaler_y.inverse_transform(model.predict(X_test))
    y_train_true = scaler_y.inverse_transform(y_train)
    y_test_true  = scaler_y.inverse_transform(y_test)

    #  Metrics
    rmse_train = mean_squared_error(y_train_true, y_train_pred, squared=False)
    mae_train  = mean_absolute_error(y_train_true, y_train_pred)
    r2_train   = r2_score(y_train_true, y_train_pred)

    rmse_test = mean_squared_error(y_test_true, y_test_pred, squared=False)
    mae_test  = mean_absolute_error(y_test_true, y_test_pred)
    r2_test   = r2_score(y_test_true, y_test_pred)

    #  Print Results
```

```
    feature_str = '+'.join(selected_features)
    print(f"\n   LSTM Performance ({feature_str}) — Train:")
    print(f"  RMSE: {rmse_train:.4f}")
    print(f"  MAE : {mae_train:.4f}")
    print(f"  R²  : {r2_train:.4f}")

    print(f"\n   LSTM Performance ({feature_str}) — Test:")
    print(f"  RMSE: {rmse_test:.4f}")
    print(f"  MAE : {mae_test:.4f}")
    print(f"  R²  : {r2_test:.4f}")

    #  Plot Predictions (Test)
    date_index = df_test.index
    plt.figure(figsize=(14, 6))
    plt.plot(date_index, y_test_true, label='Actual TSLA Close', color='black')
    plt.plot(date_index, y_test_pred, label='LSTM Predicted', linestyle='--', color='blue')
    plt.title(f"LSTM Forecast vs Actual — Features: {selected_features}")
    plt.xlabel("Date")
    plt.ylabel("TSLA Close Price")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
#%%
# Try to improve best performing model further

# Add financial features to df_lstm_all_sent
df_lstm_all_sent['daily_return'] = df_lstm_all_sent['TSLA_Close'].pct_change()
df_lstm_all_sent['hl_spread'] = df_lstm_all_sent['TSLA_High'] - df_lstm_all_sent['TSLA_Low']
df_lstm_all_sent['lagged_return'] = df_lstm_all_sent['daily_return'].shift(1)

# Drop resulting NaNs (e.g. from lag2 or division)
df_lstm_all_sent.dropna(inplace=True)
#%%
# new features plus lags
selected_features = ['lag1', 'daily_return', 'hl_spread', 'lagged_return']

#  Time-aware train/test split
split_idx = int(0.8 * len(df_lstm_all_sent))
df_train = df_lstm_all_sent[:split_idx]
df_test = df_lstm_all_sent[split_idx:]

#  Scale features and target separately
scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

X_train = scaler_X.fit_transform(df_train[selected_features])
y_train = scaler_y.fit_transform(df_train[['target']])

X_test = scaler_X.transform(df_test[selected_features])
y_test = scaler_y.transform(df_test[['target']])

#  Reshape for LSTM: (samples, timesteps, features)
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

#  Define and compile model
model = Sequential()
model.add(LSTM(units=50, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

#  Train model
history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=16,
    validation_data=(X_test, y_test),
    verbose=1,
    shuffle=False
)

#  Predict Train + Test
y_train_pred_scaled = model.predict(X_train)
y_test_pred_scaled = model.predict(X_test)

y_train_pred = scaler_y.inverse_transform(y_train_pred_scaled)
y_test_pred = scaler_y.inverse_transform(y_test_pred_scaled)

y_train_true = scaler_y.inverse_transform(y_train)
y_test_true = scaler_y.inverse_transform(y_test)

#  Evaluate Train
rmse_train = mean_squared_error(y_train_true, y_train_pred, squared=False)
mae_train = mean_absolute_error(y_train_true, y_train_pred)
r2_train = r2_score(y_train_true, y_train_pred)

#  Evaluate Test
rmse_test = mean_squared_error(y_test_true, y_test_pred, squared=False)
```

```
mae_test = mean_absolute_error(y_test_true, y_test_pred)
r2_test = r2_score(y_test_true, y_test_pred)

#  Print Results
print(f"\n   LSTM Performance ({'+'.join(selected_features)}) — Train:")
print(f"  RMSE: {rmse_train:.4f}")
print(f"  MAE : {mae_train:.4f}")
print(f"  R²  : {r2_train:.4f}")

print(f"\n   LSTM Performance ({'+'.join(selected_features)}) — Test:")
print(f"  RMSE: {rmse_test:.4f}")
print(f"  MAE : {mae_test:.4f}")
print(f"  R²  : {r2_test:.4f}")

#  Plot Predictions vs Actual (Test Set)
date_index = df_test.index
plt.figure(figsize=(14, 6))
plt.plot(date_index, y_test_true, label='Actual $TSLA Close', color='black')
plt.plot(date_index, y_test_pred, label='LSTM Predicted', linestyle='--', color='blue')
plt.title(f"LSTM Forecast vs Actual - Features: {selected_features}")
plt.xlabel("Date")
plt.ylabel("$TSLA Close Price")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
#%%
# new features plus lag plus all sentiment scores

selected_features = [
    'lag1',
    'vader_score', 'textblob_polarity', 'fintwitbert_weighted',
    'daily_return', 'hl_spread', 'lagged_return'
]

# Time-aware train/test split
split_idx = int(0.8 * len(df_lstm_all_sent))
df_train = df_lstm_all_sent[:split_idx]
df_test = df_lstm_all_sent[split_idx:]

#  Scale features and target separately
scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

X_train = scaler_X.fit_transform(df_train[selected_features])
y_train = scaler_y.fit_transform(df_train[['target']])

X_test = scaler_X.transform(df_test[selected_features])
y_test = scaler_y.transform(df_test[['target']])

#  Reshape for LSTM: (samples, timesteps, features)
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

#  Define and compile model
model = Sequential()
model.add(LSTM(units=50, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

#  Train model
history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=16,
    validation_data=(X_test, y_test),
    verbose=1,
    shuffle=False
)

#  Predict Train + Test
y_train_pred_scaled = model.predict(X_train)
y_test_pred_scaled = model.predict(X_test)

y_train_pred = scaler_y.inverse_transform(y_train_pred_scaled)
y_test_pred = scaler_y.inverse_transform(y_test_pred_scaled)

y_train_true = scaler_y.inverse_transform(y_train)
y_test_true = scaler_y.inverse_transform(y_test)

#  Evaluate Train
rmse_train = mean_squared_error(y_train_true, y_train_pred, squared=False)
mae_train = mean_absolute_error(y_train_true, y_train_pred)
r2_train = r2_score(y_train_true, y_train_pred)

#  Evaluate Test
rmse_test = mean_squared_error(y_test_true, y_test_pred, squared=False)
mae_test = mean_absolute_error(y_test_true, y_test_pred)
r2_test = r2_score(y_test_true, y_test_pred)
```

```
1229
1230 #  Print Results
1231 print(f"\n   LSTM Performance ({'+'.join(selected_features)}) — Train:")
1232 print(f"  RMSE: {rmse_train:.4f}")
1233 print(f"  MAE : {mae_train:.4f}")
1234 print(f"  R²  : {r2_train:.4f}")
1235
1236 print(f"\n   LSTM Performance ({'+'.join(selected_features)}) — Test:")
1237 print(f"  RMSE: {rmse_test:.4f}")
1238 print(f"  MAE : {mae_test:.4f}")
1239 print(f"  R²  : {r2_test:.4f}")
1240
1241 #  Plot Predictions vs Actual (Test Set)
1242
1243 date_index = df_test.index
1244 plt.figure(figsize=(14, 6))
1245 plt.plot(date_index, y_test_true, label='Actual $TSLA Close', color='black')
1246 plt.plot(date_index, y_test_pred, label='LSTM Predicted', linestyle='--', color='blue')
1247 plt.title(f"LSTM Forecast vs Actual - Features: {selected_features}")
1248 plt.xlabel("Date")
1249 plt.ylabel("$TSLA Close Price")
1250 plt.legend()
1251 plt.grid(True)
1252 plt.tight_layout()
1253 plt.show()
1254 #%%
1255 # Play with hyperparams including L2
1256 def build_model(hp):
1257     model = Sequential()
1258     model.add(
1259         LSTM(
1260             units=hp.Int('units', min_value=32, max_value=128, step=32),
1261             input_shape=(X_train.shape[1], X_train.shape[2]),
1262             return_sequences=False,
1263             kernel_regularizer=l2(hp.Float('l2_reg', min_value=1e-6, max_value=1e-2, sampling='LOG'))
1264         )
1265     )
1266     model.add(Dropout(hp.Float('dropout', 0.1, 0.5, step=0.1)))
1267     model.add(Dense(1))
1268     model.compile(
1269         optimizer=Adam(hp.Float('lr', 1e-4, 1e-2, sampling='LOG')),
1270         loss='mean_squared_error'
1271     )
1272     return model
1273
1274 # Initialize tuner
1275 tuner = RandomSearch(
1276     build_model,
1277     objective='val_loss',
1278     max_trials=10,
1279     executions_per_trial=1,
1280     directory='tuner_results',
1281     project_name='tsla_lstm_tuning_l2'
1282 )
1283
1284 # Run search
1285 tuner.search(X_train, y_train, epochs=30, batch_size=16, validation_data=(X_test, y_test), verbose=1)
1286
1287 # Get best model
1288 best_model = tuner.get_best_models(num_models=1)[0]
1289 best_model.summary()
1290 #%%
1291 # Predict on both train and test sets
1292 y_pred_train_l2 = scaler_y.inverse_transform(best_model.predict(X_train))
1293 y_true_train = scaler_y.inverse_transform(y_train)
1294
1295 y_pred_test_l2 = scaler_y.inverse_transform(best_model.predict(X_test))
1296 y_true_test = scaler_y.inverse_transform(y_test)
1297
1298 # Training metrics
1299 rmse_train = mean_squared_error(y_true_train, y_pred_train_l2, squared=False)
1300 mae_train = mean_absolute_error(y_true_train, y_pred_train_l2)
1301 r2_train = r2_score(y_true_train, y_pred_train_l2)
1302
1303 # Test metrics
1304 rmse_test = mean_squared_error(y_true_test, y_pred_test_l2, squared=False)
1305 mae_test = mean_absolute_error(y_true_test, y_pred_test_l2)
1306 r2_test = r2_score(y_true_test, y_pred_test_l2)
1307
1308 # Print results
1309 print("   Tuned LSTM (Dropout + L2) — Train Set:")
1310 print(f"  RMSE: {rmse_train:.4f}  | MAE: {mae_train:.4f}  | R²: {r2_train:.4f}")
1311
1312 print("\n   Tuned LSTM (Dropout + L2) — Test Set:")
1313 print(f"  RMSE: {rmse_test:.4f}  | MAE: {mae_test:.4f}  | R²: {r2_test:.4f}")
1314
1315 # Plot
1316 plt.figure(figsize=(14, 6))
1317 plt.plot(df_test.index, y_true_test, label='Actual $TSLA Close', color='black')
```

```
plt.plot(df_test.index, y_pred_test_l2, label='LSTM Predicted (Dropout + L2)', linestyle='--', color='blue')
plt.title("Final LSTM Forecast vs Actual — Tuned Model (Dropout + L2)")
plt.xlabel("Date")
plt.ylabel("$TSLA Close Price")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('lstm_tsla_best_forecast_tuned.png')
plt.show()
#%%

# Setup
tscv = TimeSeriesSplit(n_splits=3)
rmse_scores = []

for train_idx, val_idx in tscv.split(X_train):
    X_tr, X_val = X_train[train_idx], X_train[val_idx]
    y_tr, y_val = y_train[train_idx], y_train[val_idx]

    # Rebuild the same model structure (not tuning again)
    model = Sequential()
    model.add(LSTM(units=64, input_shape=(X_tr.shape[1], X_tr.shape[2]), kernel_regularizer=l2(0.001)))
    model.add(Dropout(0.2))
    model.add(Dense(1))
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')

    model.fit(X_tr, y_tr, epochs=30, batch_size=16, verbose=0, validation_data=(X_val, y_val), shuffle=False)

    # Predict and inverse transform
    y_val_pred = scaler_y.inverse_transform(model.predict(X_val))
    y_val_true = scaler_y.inverse_transform(y_val)

    # Compute RMSE
    rmse = mean_squared_error(y_val_true, y_val_pred, squared=False)
    rmse_scores.append(rmse)

# Output CV RMSE
print(f"   LSTM Time Series Cross-Validation RMSE: {np.mean(rmse_scores):.4f} ± {np.std(rmse_scores):.4f}")
```

```
#%%
#  Data Handling
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

#  Visualization
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use('seaborn-v0_8-whitegrid')

# Sentiment Analysis
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from textblob import TextBlob
from transformers import pipeline  # For FinTwitBERT

# Time Series Analysis
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from pmdarima import auto_arima
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.graphics.tsaplots import plot_acf

import statsmodels.api as sm


# for XGBoost
from xgboost import XGBRegressor
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer
from sklearn.model_selection import GridSearchCV
import scipy.stats as stats

# for LSTM
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from kerastuner.tuners import RandomSearch
from tensorflow.keras.callbacks import EarlyStopping
from kerastuner.engine.hyperparameters import HyperParameters
from tensorflow.keras.regularizers import l2
import itertools
import keras_tuner
#%%
# Load merged sentiment files for BTC
df_vader = pd.read_csv("/Users/malvin/DataspellProjects/MasterThesis/merged_btc_sentiment_vader.csv")
df_textblob = pd.read_csv("/Users/malvin/DataspellProjects/MasterThesis/merged_btc_sentiment_textblob.csv")
df_fintwitbert = pd.read_csv("/Users/malvin/DataspellProjects/MasterThesis/merged_btc_sentiment_fintwitbert.
csv")

# Process VADER sentiment
df_vader['date'] = pd.to_datetime(df_vader['date'])
df_vader.set_index('date', inplace=True)
btc_vader = df_vader[['BTC_Open', 'BTC_High', 'BTC_Low', 'BTC_Close', 'vader_score']].asfreq('D')
btc_vader['vader_score'] = btc_vader['vader_score'].fillna(0)

# Process TextBlob sentiment
df_textblob['date'] = pd.to_datetime(df_textblob['date'])
df_textblob.set_index('date', inplace=True)
btc_textblob = df_textblob[['BTC_Open', 'BTC_High', 'BTC_Low', 'BTC_Close', 'textblob_polarity']].asfreq('D'
)
btc_textblob['textblob_polarity'] = btc_textblob['textblob_polarity'].fillna(0)

# Process FinTwitBERT sentiment
df_fintwitbert['date'] = pd.to_datetime(df_fintwitbert['date'])
df_fintwitbert.set_index('date', inplace=True)
btc_fintwitbert = df_fintwitbert[['BTC_Open', 'BTC_High', 'BTC_Low', 'BTC_Close', 'fintwitbert_weighted']].
asfreq('D')
btc_fintwitbert['fintwitbert_weighted'] = btc_fintwitbert['fintwitbert_weighted'].fillna(0)

#%%
# Drop rows with missing price data
price_cols = ['BTC_Open', 'BTC_High', 'BTC_Low', 'BTC_Close']

btc_vader = btc_vader.dropna(subset=price_cols)
btc_textblob = btc_textblob.dropna(subset=price_cols)
btc_fintwitbert = btc_fintwitbert.dropna(subset=price_cols)
#%%
# ADF test function
def run_adf_test(series, label):
    result = adfuller(series.dropna())
    print(f"\n ADF Test for {label} ")
    print(f"ADF Statistic: {result[0]:.4f}")
    print(f"p-value: {result[1]:.4f}")
    if result[1] < 0.05:
```

```
        print("   Series is likely stationary (no differencing needed).")
    else:
        print("⚠️ Series is likely non-stationary (consider differencing).")
# Run ADF test on BTC_Close
run_adf_test(btc_vader['BTC_Close'], 'BTC_Close')
#%%
# ARIMA using BTC closing price only

y_btc = btc_vader['BTC_Close']
y_btc.index = pd.to_datetime(y_btc.index)

# 80/20 split
split_idx = int(0.8 * len(y_btc))
y_btc_train = y_btc[:split_idx]
y_btc_test = y_btc[split_idx:]

# Train ARIMA
auto_model = auto_arima(
    y_btc_train,
    start_p=1, start_q=1,
    max_p=5, max_q=5,
    d=1, seasonal=False,
    stepwise=True, trace=True
)

print(f"\nBest ARIMA order: {auto_model.order}")

# Forecast test period
forecast_auto = auto_model.predict(n_periods=len(y_btc_test))

# Evaluate Train
forecast_train = auto_model.predict_in_sample(start=1, end=split_idx - 1)
y_btc_train_trimmed = y_btc_train[1:]

rmse_train = mean_squared_error(y_btc_train_trimmed, forecast_train, squared=False)
mae_train = mean_absolute_error(y_btc_train_trimmed, forecast_train)
r2_train = r2_score(y_btc_train_trimmed, forecast_train)

print("\n   ARIMA Performance (Train Set):")
print(f"  RMSE: {rmse_train:.4f}")
print(f"  MAE : {mae_train:.4f}")
print(f"  R²  : {r2_train:.4f}")

# Evaluate Test
rmse_test = mean_squared_error(y_btc_test, forecast_auto, squared=False)
mae_test = mean_absolute_error(y_btc_test, forecast_auto)
r2_test = r2_score(y_btc_test, forecast_auto)

print("\n   ARIMA Performance (Test Set):")
print(f"  RMSE: {rmse_test:.4f}")
print(f"  MAE : {mae_test:.4f}")
print(f"  R²  : {r2_test:.4f}")

print(f"\nPrice Range (Test Set): Min = {y_btc_test.min():.2f}, Max = {y_btc_test.max():.2f}")

# Plot
plt.figure(figsize=(14, 6))
plt.plot(y_btc, label='Actual BTC Close', color='black')
plt.plot(y_btc_test.index, forecast_auto, label='ARIMA Forecast (Auto)', color='blue', linestyle='--')
plt.axvline(y_btc.index[split_idx], color='gray', linestyle='--', label='Train/Test Split')
plt.title('ARIMA Forecast vs Actual BTC Price (80/20 Split)')
plt.xlabel('Date')
plt.ylabel('BTC Close Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig("arima_btc_forecast.png", dpi=300)
plt.show()
#%%
print(auto_model.summary())
#%%
# ARIMA model summary and residual diagnostics
print(auto_model.summary())

# Residuals
residuals = auto_model.resid()

plt.figure(figsize=(12, 4))
plt.plot(residuals)
plt.title("Residuals from ARIMA Model")
plt.xlabel("Time")
plt.ylabel("Residual")
plt.grid(True)
plt.tight_layout()
plt.show()

plt.figure(figsize=(8, 4))
sns.histplot(residuals, bins=30, kde=True)
plt.title("Distribution of Residuals")
```

```
plt.xlabel("Residual")
plt.tight_layout()
plt.show()

sm.graphics.tsa.plot_acf(residuals.dropna(), lags=40)
plt.title("Autocorrelation of Residuals")
plt.tight_layout()
plt.show()
#%%
#  ARIMAX with VADER Sentiment as Exogenous Variable
X_vader = btc_vader[['vader_score']]
X_vader_train = X_vader[:split_idx]
X_vader_test = X_vader[split_idx:]

model_arimax_vader = ARIMA(endog=y_btc_train, exog=X_vader_train, order=(0, 1, 0))
model_arimax_vader_fit = model_arimax_vader.fit()

forecast_arimax_vader = model_arimax_vader_fit.forecast(steps=len(y_btc_test), exog=X_vader_test)

plt.figure(figsize=(14, 6))
plt.plot(y_btc, label='Actual BTC Close', color='black')
plt.plot(y_btc_test.index, forecast_arimax_vader, label='ARIMAX Forecast (VADER)', color='green', linestyle
='--')
plt.axvline(y_btc.index[split_idx], color='gray', linestyle='--', label='Train/Test Split')
plt.title('ARIMAX Forecast vs Actual (w/ VADER Sentiment)')
plt.xlabel('Date')
plt.ylabel('BTC Close Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

#  Forecast on Training Set (ARIMAX VADER)
forecast_train_vader = model_arimax_vader_fit.predict(start=1, end=split_idx - 1, exog=X_vader_train)
y_btc_train_trimmed = y_btc_train[1:]

rmse_train_vader = mean_squared_error(y_btc_train_trimmed, forecast_train_vader, squared=False)
mae_train_vader = mean_absolute_error(y_btc_train_trimmed, forecast_train_vader)
r2_train_vader = r2_score(y_btc_train_trimmed, forecast_train_vader)

print("\n   ARIMAX (VADER) Performance (Train Set):")
print(f"  RMSE: {rmse_train_vader:.4f}")
print(f"  MAE : {mae_train_vader:.4f}")
print(f"  R²  : {r2_train_vader:.4f}")

rmse_test_vader = mean_squared_error(y_btc_test, forecast_arimax_vader, squared=False)
mae_test_vader = mean_absolute_error(y_btc_test, forecast_arimax_vader)
r2_test_vader = r2_score(y_btc_test, forecast_arimax_vader)

print("\n   ARIMAX (VADER) Performance (Test Set):")
print(f"  RMSE: {rmse_test_vader:.4f}")
print(f"  MAE : {mae_test_vader:.4f}")
print(f"  R²  : {r2_test_vader:.4f}")

print(f"\nPrice Range (Test Set): Min = {y_btc_test.min():.2f}, Max = {y_btc_test.max():.2f}")

#%%
#  ARIMAX with TextBlob Sentiment
X_textblob = btc_textblob[['textblob_polarity']]
X_textblob_train = X_textblob[:split_idx]
X_textblob_test = X_textblob[split_idx:]

model_arimax_textblob = ARIMA(endog=y_btc_train, exog=X_textblob_train, order=(0, 1, 0))
model_arimax_textblob_fit = model_arimax_textblob.fit()

forecast_arimax_textblob = model_arimax_textblob_fit.forecast(steps=len(y_btc_test), exog=X_textblob_test)

plt.figure(figsize=(14, 6))
plt.plot(y_btc, label='Actual BTC Close', color='black')
plt.plot(y_btc_test.index, forecast_arimax_textblob, label='ARIMAX Forecast (TextBlob)', color='orange',
linestyle='--')
plt.axvline(y_btc.index[split_idx], color='gray', linestyle='--', label='Train/Test Split')
plt.title('ARIMAX Forecast vs Actual (w/ TextBlob Sentiment)')
plt.xlabel('Date')
plt.ylabel('BTC Close Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

forecast_train_textblob = model_arimax_textblob_fit.predict(start=1, end=split_idx - 1, exog=
X_textblob_train)

rmse_train_textblob = mean_squared_error(y_btc_train_trimmed, forecast_train_textblob, squared=False)
mae_train_textblob = mean_absolute_error(y_btc_train_trimmed, forecast_train_textblob)
r2_train_textblob = r2_score(y_btc_train_trimmed, forecast_train_textblob)

print("\n   ARIMAX (TextBlob) Performance (Train Set):")
print(f"  RMSE: {rmse_train_textblob:.4f}")
```

```
print(f"  MAE : {mae_train_textblob:.4f}")
print(f"  R²  : {r2_train_textblob:.4f}")

rmse_test_textblob = mean_squared_error(y_btc_test, forecast_arimax_textblob, squared=False)
mae_test_textblob = mean_absolute_error(y_btc_test, forecast_arimax_textblob)
r2_test_textblob = r2_score(y_btc_test, forecast_arimax_textblob)

print("\n   ARIMAX (TextBlob) Performance (Test Set):")
print(f"  RMSE: {rmse_test_textblob:.4f}")
print(f"  MAE : {mae_test_textblob:.4f}")
print(f"  R²  : {r2_test_textblob:.4f}")
#%%

#  ARIMAX with FinTwitBERT Sentiment
X_bert = btc_fintwitbert[['fintwitbert_weighted']]
X_bert_train = X_bert[:split_idx]
X_bert_test = X_bert[split_idx:]

model_arimax_bert = ARIMA(endog=y_btc_train, exog=X_bert_train, order=(0, 1, 0))
model_arimax_bert_fit = model_arimax_bert.fit()

forecast_arimax_bert = model_arimax_bert_fit.forecast(steps=len(y_btc_test), exog=X_bert_test)

plt.figure(figsize=(14, 6))
plt.plot(y_btc, label='Actual BTC Close', color='black')
plt.plot(y_btc_test.index, forecast_arimax_bert, label='ARIMAX Forecast (FinTwitBERT)', color='purple',
    linestyle='--')
plt.axvline(y_btc.index[split_idx], color='gray', linestyle='--', label='Train/Test Split')
plt.title('ARIMAX Forecast vs Actual (w/ FinTwitBERT Sentiment)')
plt.xlabel('Date')
plt.ylabel('BTC Close Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

forecast_train_bert = model_arimax_bert_fit.predict(start=1, end=split_idx - 1, exog=X_bert_train)

rmse_train_bert = mean_squared_error(y_btc_train_trimmed, forecast_train_bert, squared=False)
mae_train_bert = mean_absolute_error(y_btc_train_trimmed, forecast_train_bert)
r2_train_bert = r2_score(y_btc_train_trimmed, forecast_train_bert)

print("\n   ARIMAX (FinTwitBERT) Performance (Train Set):")
print(f"  RMSE: {rmse_train_bert:.4f}")
print(f"  MAE : {mae_train_bert:.4f}")
print(f"  R²  : {r2_train_bert:.4f}")

rmse_test_bert = mean_squared_error(y_btc_test, forecast_arimax_bert, squared=False)
mae_test_bert = mean_absolute_error(y_btc_test, forecast_arimax_bert)
r2_test_bert = r2_score(y_btc_test, forecast_arimax_bert)

print("\n   ARIMAX (FinTwitBERT) Performance (Test Set):")
print(f"  RMSE: {rmse_test_bert:.4f}")
print(f"  MAE : {mae_test_bert:.4f}")
print(f"  R²  : {r2_test_bert:.4f}")
#%%
# PACF Plot to decide lags for XGBoost
plot_pacf(btc_vader['BTC_Close'], lags=20, method='ywm')
plt.title('Partial Autocorrelation (PACF) of $BTC Closing Price')
plt.xlabel('Lag')
plt.ylabel('Partial Autocorrelation')
plt.grid(True)
plt.tight_layout()
plt.savefig("pacf_btc_closing_price.png", dpi=300, bbox_inches='tight')
plt.show()
#%%

# Create lagged features and target for XGBoost
df_xgb_base = btc_vader.copy()
df_xgb_base['lag1'] = df_xgb_base['BTC_Close'].shift(1)
df_xgb_base['target'] = df_xgb_base['BTC_Close'].shift(-1)
df_xgb_base.dropna(inplace=True)

X_xgb_base = df_xgb_base[['lag1']]
y_xgb_base = df_xgb_base['target']

# Train/test split
split_idx_xgb = int(0.8 * len(df_xgb_base))
X_xgb_base_train = X_xgb_base[:split_idx_xgb]
X_xgb_base_test = X_xgb_base[split_idx_xgb:]
y_xgb_base_train = y_xgb_base[:split_idx_xgb]
y_xgb_base_test = y_xgb_base[split_idx_xgb:]

# Fit base XGBoost model
xgb_model_base = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_model_base.fit(X_xgb_base_train, y_xgb_base_train)

preds_train = xgb_model_base.predict(X_xgb_base_train)
preds_test = xgb_model_base.predict(X_xgb_base_test)
```

```
350
351 rmse_train = mean_squared_error(y_xgb_base_train, preds_train, squared=False)
352 mae_train = mean_absolute_error(y_xgb_base_train, preds_train)
353 r2_train = r2_score(y_xgb_base_train, preds_train)
354
355 rmse_test = mean_squared_error(y_xgb_base_test, preds_test, squared=False)
356 mae_test = mean_absolute_error(y_xgb_base_test, preds_test)
357 r2_test = r2_score(y_xgb_base_test, preds_test)
358
359 print("   XGBoost Performance Price only (Train Set):")
360 print(f"  RMSE: {rmse_train:.4f}")
361 print(f"  MAE : {mae_train:.4f}")
362 print(f"  R²  : {r2_train:.4f}")
363
364 print("\n   XGBoost Performance Price only (Test Set):")
365 print(f"  RMSE: {rmse_test:.4f}")
366 print(f"  MAE : {mae_test:.4f}")
367 print(f"  R²  : {r2_test:.4f}")
368
369 plt.figure(figsize=(12, 6))
370 plt.plot(y_xgb_base_test.values, label='Actual')
371 plt.plot(preds_test, label='Predicted', linestyle='--')
372 plt.title("XGBoost Prediction vs Actual (Base Model - $BTC)")
373 plt.legend()
374 plt.grid(True)
375 plt.tight_layout()
376 plt.show()
377
378 #%%
379 #  Build merged df with sentiment and lagged features (BTC)
380 df_xgb_all_sent = btc_vader[['BTC_Open', 'BTC_High', 'BTC_Low', 'BTC_Close', 'vader_score']].copy()
381 df_xgb_all_sent = df_xgb_all_sent.join(btc_textblob[['textblob_polarity']])
382 df_xgb_all_sent = df_xgb_all_sent.join(btc_fintwitbert[['fintwitbert_weighted']])
383
384 df_xgb_all_sent['lag1'] = df_xgb_all_sent['BTC_Close'].shift(1)
385
386 df_xgb_all_sent['daily_return'] = df_xgb_all_sent['BTC_Close'].pct_change()
387 df_xgb_all_sent['hl_spread'] = df_xgb_all_sent['BTC_High'] - df_xgb_all_sent['BTC_Low']
388 df_xgb_all_sent['lagged_return'] = df_xgb_all_sent['daily_return'].shift(1)
389
390 df_xgb_all_sent['target'] = df_xgb_all_sent['BTC_Close'].shift(-1)
391
392 df_xgb_all_sent.dropna(inplace=True)
393 #%%
394 #  Define features and target
395 X_xgb_all_sent = df_xgb_all_sent[['lag1', 'vader_score', 'textblob_polarity', 'fintwitbert_weighted']]
396 y_xgb_all_sent = df_xgb_all_sent['target']
397
398 # Time-aware train/test split
399 split_idx_xgb_all_sent = int(0.8 * len(df_xgb_all_sent))
400 X_xgb_all_sent_train = X_xgb_all_sent[:split_idx_xgb_all_sent]
401 X_xgb_all_sent_test = X_xgb_all_sent[split_idx_xgb_all_sent:]
402 y_xgb_all_sent_train = y_xgb_all_sent[:split_idx_xgb_all_sent]
403 y_xgb_all_sent_test = y_xgb_all_sent[split_idx_xgb_all_sent:]
404
405 #  Train XGBoost with VADER sentiment only
406 features_vader = ['lag1', 'vader_score']
407
408 xgb_model_vader_only = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
409 xgb_model_vader_only.fit(X_xgb_all_sent_train[features_vader], y_xgb_all_sent_train)
410
411 y_pred_train_vader = xgb_model_vader_only.predict(X_xgb_all_sent_train[features_vader])
412 y_pred_test_vader = xgb_model_vader_only.predict(X_xgb_all_sent_test[features_vader])
413
414 # Evaluate performance
415 rmse_train = mean_squared_error(y_xgb_all_sent_train, y_pred_train_vader, squared=False)
416 mae_train = mean_absolute_error(y_xgb_all_sent_train, y_pred_train_vader)
417 r2_train = r2_score(y_xgb_all_sent_train, y_pred_train_vader)
418
419 rmse_test = mean_squared_error(y_xgb_all_sent_test, y_pred_test_vader, squared=False)
420 mae_test = mean_absolute_error(y_xgb_all_sent_test, y_pred_test_vader)
421 r2_test = r2_score(y_xgb_all_sent_test, y_pred_test_vader)
422
423 print("   XGBoost (Lag + VADER) - Train Set:")
424 print(f"  RMSE: {rmse_train:.4f}")
425 print(f"  MAE : {mae_train:.4f}")
426 print(f"  R²  : {r2_train:.4f}")
427
428 print("\n   XGBoost (Lag + VADER) - Test Set:")
429 print(f"  RMSE: {rmse_test:.4f}")
430 print(f"  MAE : {mae_test:.4f}")
431 print(f"  R²  : {r2_test:.4f}")
432
433 plt.figure(figsize=(14, 6))
434 plt.plot(y_xgb_all_sent_test.index, y_xgb_all_sent_test, label='Actual $BTC Close', color='black')
435 plt.plot(y_xgb_all_sent_test.index, y_pred_test_vader, label='XGBoost Predicted (VADER Only)', color='blue'
    ,
436          linestyle='--')
437 plt.axvline(y_xgb_all_sent_test.index[0], color='gray', linestyle='--', label='Train/Test Split')
```

```
plt.title('XGBoost Forecast vs Actual (Lag + VADER)')
plt.xlabel('Date')
plt.ylabel('$BTC Close Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
#%%
#  XGBoost using lag1, and TextBlob polarity
features_textblob = ['lag1', 'textblob_polarity']

xgb_model_textblob_only = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_model_textblob_only.fit(X_xgb_all_sent_train[features_textblob], y_xgb_all_sent_train)

y_xgb_pred_textblob_train = xgb_model_textblob_only.predict(X_xgb_all_sent_train[features_textblob])
y_xgb_pred_textblob_test = xgb_model_textblob_only.predict(X_xgb_all_sent_test[features_textblob])

rmse_train_tb = mean_squared_error(y_xgb_all_sent_train, y_xgb_pred_textblob_train, squared=False)
mae_train_tb = mean_absolute_error(y_xgb_all_sent_train, y_xgb_pred_textblob_train)
r2_train_tb = r2_score(y_xgb_all_sent_train, y_xgb_pred_textblob_train)

rmse_test_tb = mean_squared_error(y_xgb_all_sent_test, y_xgb_pred_textblob_test, squared=False)
mae_test_tb = mean_absolute_error(y_xgb_all_sent_test, y_xgb_pred_textblob_test)
r2_test_tb = r2_score(y_xgb_all_sent_test, y_xgb_pred_textblob_test)

print("   XGBoost (Lag + TextBlob) - Train Set:")
print(f"  RMSE: {rmse_train_tb:.4f}")
print(f"  MAE : {mae_train_tb:.4f}")
print(f"  R²  : {r2_train_tb:.4f}")

print("\n   XGBoost (Lag + TextBlob) - Test Set:")
print(f"  RMSE: {rmse_test_tb:.4f}")
print(f"  MAE : {mae_test_tb:.4f}")
print(f"  R²  : {r2_test_tb:.4f}")

plt.figure(figsize=(14, 6))
plt.plot(y_xgb_all_sent_test.index, y_xgb_all_sent_test, label='Actual $BTC Close', color='black')
plt.plot(y_xgb_all_sent_test.index, y_xgb_pred_textblob_test, label='XGBoost Predicted (TextBlob Only)',
    color='orange', linestyle='--')
plt.axvline(y_xgb_all_sent_test.index[0], color='gray', linestyle='--', label='Train/Test Split')
plt.title('XGBoost Forecast vs Actual (Lag + TextBlob)')
plt.xlabel('Date')
plt.ylabel('$BTC Close Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
#%%
#  XGBoost using lag1, and FinTwitBERT
features_fintwitbert = ['lag1', 'fintwitbert_weighted']

xgb_model_fintwitbert_only = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_model_fintwitbert_only.fit(X_xgb_all_sent_train[features_fintwitbert], y_xgb_all_sent_train)

y_xgb_pred_fintwitbert_train = xgb_model_fintwitbert_only.predict(X_xgb_all_sent_train[features_fintwitbert
    ])
y_xgb_pred_fintwitbert_test = xgb_model_fintwitbert_only.predict(X_xgb_all_sent_test[features_fintwitbert])

rmse_train_ftb = mean_squared_error(y_xgb_all_sent_train, y_xgb_pred_fintwitbert_train, squared=False)
mae_train_ftb = mean_absolute_error(y_xgb_all_sent_train, y_xgb_pred_fintwitbert_train)
r2_train_ftb = r2_score(y_xgb_all_sent_train, y_xgb_pred_fintwitbert_train)

rmse_test_ftb = mean_squared_error(y_xgb_all_sent_test, y_xgb_pred_fintwitbert_test, squared=False)
mae_test_ftb = mean_absolute_error(y_xgb_all_sent_test, y_xgb_pred_fintwitbert_test)
r2_test_ftb = r2_score(y_xgb_all_sent_test, y_xgb_pred_fintwitbert_test)

print("   XGBoost (Lag + FinTwitBERT) - Train Set:")
print(f"  RMSE: {rmse_train_ftb:.4f}")
print(f"  MAE : {mae_train_ftb:.4f}")
print(f"  R²  : {r2_train_ftb:.4f}")

print("\n   XGBoost (Lag + FinTwitBERT) - Test Set:")
print(f"  RMSE: {rmse_test_ftb:.4f}")
print(f"  MAE : {mae_test_ftb:.4f}")
print(f"  R²  : {r2_test_ftb:.4f}")

plt.figure(figsize=(14, 6))
plt.plot(y_xgb_all_sent_test.index, y_xgb_all_sent_test, label='Actual $BTC Close', color='black')
plt.plot(y_xgb_all_sent_test.index, y_xgb_pred_fintwitbert_test, label='XGBoost Predicted (FinTwitBERT Only
    )',
         color='purple', linestyle='--')
plt.axvline(y_xgb_all_sent_test.index[0], color='gray', linestyle='--', label='Train/Test Split')
plt.title('XGBoost Forecast vs Actual (Lag + FinTwitBERT)')
plt.xlabel('Date')
plt.ylabel('$BTC Close Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('xgboost_btc_best_forecast.png')
```

```
plt.show()

#%%
#  XGBoost using all lagged + sentiment features
features_all_sent = ['lag1', 'vader_score', 'textblob_polarity', 'fintwitbert_weighted']

xgb_model_all_sent = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_model_all_sent.fit(X_xgb_all_sent_train[features_all_sent], y_xgb_all_sent_train)

y_xgb_pred_all_sent_train = xgb_model_all_sent.predict(X_xgb_all_sent_train[features_all_sent])
y_xgb_pred_all_sent_test = xgb_model_all_sent.predict(X_xgb_all_sent_test[features_all_sent])

rmse_train_all = mean_squared_error(y_xgb_all_sent_train, y_xgb_pred_all_sent_train, squared=False)
mae_train_all = mean_absolute_error(y_xgb_all_sent_train, y_xgb_pred_all_sent_train)
r2_train_all = r2_score(y_xgb_all_sent_train, y_xgb_pred_all_sent_train)

rmse_test_all = mean_squared_error(y_xgb_all_sent_test, y_xgb_pred_all_sent_test, squared=False)
mae_test_all = mean_absolute_error(y_xgb_all_sent_test, y_xgb_pred_all_sent_test)
r2_test_all = r2_score(y_xgb_all_sent_test, y_xgb_pred_all_sent_test)

print("   XGBoost (Lag + All Sentiment) - Train Set:")
print(f"  RMSE: {rmse_train_all:.4f}")
print(f"  MAE : {mae_train_all:.4f}")
print(f"  R²  : {r2_train_all:.4f}")

print("\n   XGBoost (Lag + All Sentiment) - Test Set:")
print(f"  RMSE: {rmse_test_all:.4f}")
print(f"  MAE : {mae_test_all:.4f}")
print(f"  R²  : {r2_test_all:.4f}")

plt.figure(figsize=(14, 6))
plt.plot(y_xgb_all_sent_test.index, y_xgb_all_sent_test, label='Actual $BTC Close', color='black')
plt.plot(y_xgb_all_sent_test.index, y_xgb_pred_all_sent_test, label='XGBoost Predicted (All Sentiment)',
    color='purple',
         linestyle='--')
plt.axvline(y_xgb_all_sent_test.index[0], color='gray', linestyle='--', label='Train/Test Split')
plt.title('XGBoost Forecast vs Actual (Lag + All Sentiment)')
plt.xlabel('Date')
plt.ylabel('$BTC Close Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

#%%
#  XGBoost with lagged + sentiment + financial features
df_xgb_all_sent['daily_return'] = df_xgb_all_sent['BTC_Close'].pct_change()
df_xgb_all_sent['hl_spread'] = df_xgb_all_sent['BTC_High'] - df_xgb_all_sent['BTC_Low']
df_xgb_all_sent['lagged_return'] = df_xgb_all_sent['daily_return'].shift(1)

# Drop missing values after adding new features
df_xgb_all_sent.dropna(inplace=True)

# Features and target
total_features = ['lag1', 'vader_score', 'daily_return',
                  'hl_spread', 'lagged_return']
X_xgb_full = df_xgb_all_sent[total_features]
y_xgb_full = df_xgb_all_sent['target']

# Train/test split
split_idx_xgb_full = int(0.8 * len(df_xgb_all_sent))
X_xgb_full_train = X_xgb_full[:split_idx_xgb_full]
X_xgb_full_test = X_xgb_full[split_idx_xgb_full:]
y_xgb_full_train = y_xgb_full[:split_idx_xgb_full]
y_xgb_full_test = y_xgb_full[split_idx_xgb_full:]

# Train model
xgb_model_full = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_model_full.fit(X_xgb_full_train, y_xgb_full_train)

# Predictions
y_pred_train_full = xgb_model_full.predict(X_xgb_full_train)
y_pred_test_full = xgb_model_full.predict(X_xgb_full_test)

# Evaluation
rmse_train_full = mean_squared_error(y_xgb_full_train, y_pred_train_full, squared=False)
mae_train_full = mean_absolute_error(y_xgb_full_train, y_pred_train_full)
r2_train_full = r2_score(y_xgb_full_train, y_pred_train_full)

rmse_test_full = mean_squared_error(y_xgb_full_test, y_pred_test_full, squared=False)
mae_test_full = mean_absolute_error(y_xgb_full_test, y_pred_test_full)
r2_test_full = r2_score(y_xgb_full_test, y_pred_test_full)

# Results
print("   XGBoost (Lag + All Sentiment + Financial) - Train Set:")
print(f"  RMSE: {rmse_train_full:.4f}")
print(f"  MAE : {mae_train_full:.4f}")
print(f"  R²  : {r2_train_full:.4f}")

```

```
print("\n   XGBoost (Lag + All Sentiment + Financial) - Test Set:")
print(f"  RMSE: {rmse_test_full:.4f}")
print(f"  MAE : {mae_test_full:.4f}")
print(f"  R²  : {r2_test_full:.4f}")

# Plot
plt.figure(figsize=(14, 6))
plt.plot(y_xgb_full_test.index, y_xgb_full_test, label='Actual $BTC Close', color='black')
plt.plot(y_xgb_full_test.index, y_pred_test_full, label='Predicted (All Features)', color='teal', linestyle='--')
plt.axvline(y_xgb_full_test.index[0], color='gray', linestyle='--', label='Train/Test Split')
plt.title('XGBoost Forecast vs Actual (Lag + Sentiment + Financial)')
plt.xlabel('Date')
plt.ylabel('$BTC Close Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
#%%
# LSTM start from VADER base for BTC
df_lstm_all_sent = btc_vader[['BTC_Open', 'BTC_High', 'BTC_Low', 'BTC_Close', 'vader_score']].copy()

# Add TextBlob polarity
df_lstm_all_sent = df_lstm_all_sent.join(btc_textblob[['textblob_polarity']])

# Add FinTwitBERT weighted score
df_lstm_all_sent = df_lstm_all_sent.join(btc_fintwitbert[['fintwitbert_weighted']])

# Lagged price features
df_lstm_all_sent['lag1'] = df_lstm_all_sent['BTC_Close'].shift(1)

# Target: next day's closing price
df_lstm_all_sent['target'] = df_lstm_all_sent['BTC_Close'].shift(-1)

# Drop rows with NaNs
df_lstm_all_sent.dropna(inplace=True)
```

```
#%%
#  Full LSTM Pipeline for BTC (Baseline)

# Choose features for baseline model
selected_features = ['lag1']

# Train/test split
split_idx = int(0.8 * len(df_lstm_all_sent))
df_train = df_lstm_all_sent[:split_idx]
df_test = df_lstm_all_sent[split_idx:]

# Scale input features and target
target_scaler = MinMaxScaler()
feature_scaler = MinMaxScaler()

X_train = feature_scaler.fit_transform(df_train[selected_features])
y_train = target_scaler.fit_transform(df_train[['target']])

X_test = feature_scaler.transform(df_test[selected_features])
y_test = target_scaler.transform(df_test[['target']])

# Reshape for LSTM
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

# Build model
model = Sequential()
model.add(LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

# Train model
history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=16,
    validation_data=(X_test, y_test),
    verbose=1,
    shuffle=False
)

# Predict and inverse transform
y_train_pred = target_scaler.inverse_transform(model.predict(X_train))
y_test_pred = target_scaler.inverse_transform(model.predict(X_test))
y_train_true = target_scaler.inverse_transform(y_train)
y_test_true = target_scaler.inverse_transform(y_test)

# Evaluate performance
rmse_train = mean_squared_error(y_train_true, y_train_pred, squared=False)
mae_train = mean_absolute_error(y_train_true, y_train_pred)
r2_train = r2_score(y_train_true, y_train_pred)
```

```
rmse_test = mean_squared_error(y_test_true, y_test_pred, squared=False)
mae_test = mean_absolute_error(y_test_true, y_test_pred)
r2_test = r2_score(y_test_true, y_test_pred)

print(f"\n   LSTM Performance ({'+'.join(selected_features)}) — Train:")
print(f"  RMSE: {rmse_train:.4f}")
print(f"  MAE : {mae_train:.4f}")
print(f"  R²  : {r2_train:.4f}")

print(f"\n   LSTM Performance ({'+'.join(selected_features)}) — Test:")
print(f"  RMSE: {rmse_test:.4f}")
print(f"  MAE : {mae_test:.4f}")
print(f"  R²  : {r2_test:.4f}")

# Plot test predictions
plt.figure(figsize=(14, 6))
plt.plot(df_test.index, y_test_true, label='Actual BTC Close', color='black')
plt.plot(df_test.index, y_test_pred, label='LSTM Predicted', linestyle='--', color='blue')
plt.title(f"LSTM Forecast vs Actual - Features: {selected_features}")
plt.xlabel("Date")
plt.ylabel("BTC Close Price")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

#%%
#  Loop over multiple feature sets for LSTM evaluation (BTC)
feature_sets = [
    ['lag1', 'vader_score'],
    ['lag1', 'textblob_polarity'],
    ['lag1', 'fintwitbert_weighted'],
    ['lag1', 'vader_score', 'textblob_polarity', 'fintwitbert_weighted']
]

for selected_features in feature_sets:
    split_idx = int(0.8 * len(df_lstm_all_sent))
    df_train = df_lstm_all_sent[:split_idx]
    df_test = df_lstm_all_sent[split_idx:]

    scaler_X = MinMaxScaler()
    scaler_y = MinMaxScaler()

    X_train = scaler_X.fit_transform(df_train[selected_features])
    y_train = scaler_y.fit_transform(df_train[['target']])

    X_test = scaler_X.transform(df_test[selected_features])
    y_test = scaler_y.transform(df_test[['target']])

    X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
    X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

    model = Sequential()
    model.add(LSTM(units=50, input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mean_squared_error')

    model.fit(
        X_train, y_train,
        epochs=50,
        batch_size=16,
        validation_data=(X_test, y_test),
        verbose=0,
        shuffle=False
    )

    y_train_pred = scaler_y.inverse_transform(model.predict(X_train))
    y_test_pred = scaler_y.inverse_transform(model.predict(X_test))
    y_train_true = scaler_y.inverse_transform(y_train)
    y_test_true = scaler_y.inverse_transform(y_test)

    rmse_train = mean_squared_error(y_train_true, y_train_pred, squared=False)
    mae_train = mean_absolute_error(y_train_true, y_train_pred)
    r2_train = r2_score(y_train_true, y_train_pred)

    rmse_test = mean_squared_error(y_test_true, y_test_pred, squared=False)
    mae_test = mean_absolute_error(y_test_true, y_test_pred)
    r2_test = r2_score(y_test_true, y_test_pred)

    feature_str = '+'.join(selected_features)
    print(f"\n   LSTM Performance ({feature_str}) — Train:")
    print(f"  RMSE: {rmse_train:.4f}")
    print(f"  MAE : {mae_train:.4f}")
    print(f"  R²  : {r2_train:.4f}")

    print(f"\n   LSTM Performance ({feature_str}) — Test:")
    print(f"  RMSE: {rmse_test:.4f}")
    print(f"  MAE : {mae_test:.4f}")
    print(f"  R²  : {r2_test:.4f}")
```