

Group ID - MSc in Data Analytics

Author: Olena Pleshan

e-mail: [sbs24043@student.cct.ie](mailto:sbs24043@student.cct.ie)

Student ID: sbs24043

GitHub Link: <https://github.com/CCT-Dublin/integrated-ca2-sbs24043>

<b>Abstract</b>	<b>3</b>
<b>Introduction</b>	<b>3</b>
<b>Materials and Methods</b>	<b>3</b>
<b>Results / Discussion</b>	<b>4</b>
<b>Data Storage And Management</b>	<b>4</b>
Database Usage	4
Databases Comparison	5
<b>Big Data Architecture</b>	<b>6</b>
Data Flow Architecture	6
Apache Spark	7
<b>Technology Stack Selection</b>	<b>7</b>
Performance Considerations	7
Alternative Implementations	8
<b>Data Preparation</b>	<b>9</b>
Data Preparation: Requirements and Approaches	9
Feature Engineering: NLP	9
Exploratory Data Analysis	9
<b>Machine Learning</b>	<b>11</b>
Time Series Analysis	11
Figure 7: ML algorithms experiment table	12
Natural Languages Processing: Assessment	14
<b>User-Facing Dashboard</b>	<b>14</b>
<b>Conclusion</b>	<b>17</b>
<b>References</b>	<b>17</b>
<b>Index of Figures</b>	<b>19</b>

# Abstract

This study investigates the integration of stock price data with Twitter sentiment to enhance stock trend predictions. Using sentiment analysis models (VADER and BERT) and time-series forecasting techniques (ARIMA and RNNs), it combines social sentiment and financial data for accurate modeling. MongoDB and MySQL were employed to manage large datasets, enabling efficient retrieval and streamlined processing. An interactive dashboard was developed to visualize predictions, allowing users to explore trends and model outputs. Results indicate that public sentiment significantly influences market behavior, with the proposed approach providing a scalable, real-time predictive tool that could benefit investors and analysts.

## Introduction

This report explores a comprehensive pipeline for integrating and analyzing stock market data and Twitter sentiment to predict stock price movements. Utilizing historical stock data and social media sentiment, it employs various machine learning and deep learning models to assess the relationship between public sentiment and market performance. This study begins with extensive data collection, preprocessing, and feature engineering to create a dataset that captures daily closing stock prices and sentiment trends from social media.

The methodology includes advanced sentiment analysis techniques like VADER and BERT to generate sentiment scores, which are then synchronized with stock data for robust time-series analysis. Several machine learning models are implemented, including Random Forest, ARIMA, and RNNs, each tailored to enhance prediction accuracy through data preprocessing, feature scaling, and model tuning. The research also explores the differences in performance and usage of SQL and NoSQL databases to store large datasets and enable efficient retrieval of processed data, especially in MongoDB and MySQL. A key outcome of this study is the construction of a user-facing dashboard with real-time interactive components, providing end-users with accessible insights and forecast visualizations based on model predictions.

Overall, this report offers an in-depth view of a data pipeline designed for stock price prediction through sentiment analysis, highlighting both the technical approaches used and potential enhancements for further optimization and end-user functionality.

## Materials and Methods

Below are the materials and methods used while performing this research.

### Datasets:

1. Stock Prices: Historical stock price data including features like date, close, volume, etc.
2. Twitter Data: Tweets related to stocks, including features like date, tweet, num\_tweets, etc.

### Data Loading and Preprocessing:

1. Reading Data: Load stock prices and Twitter data from CSV files or databases.
2. Data Cleaning: Handle missing values, remove duplicates, and preprocess text data.

3. Feature Engineering: Create new features such as moving averages, sentiment scores, and labels.

#### Sentiment Analysis:

1. VADER: Use VADER to perform sentiment analysis on tweets.
2. BERT: Use BERT for more advanced sentiment analysis, extracting sentiment scores and labels. Used a library, pretrained on Twitter data specific to Financial services

#### Time Series Analysis:

1. Stationarity Tests: Perform tests like the Augmented Dickey-Fuller (ADF) test to check for stationarity.
2. Differencing: Apply differencing to make the time series stationary if needed.

#### Exploratory Data Analysis (EDA):

1. Descriptive Statistics: Calculate summary statistics for numerical features.
2. Visualization: Create plots to visualize trends, distributions, and correlations.

#### Machine Learning Models:

1. Feature Scaling: Scale features using techniques like Min-Max scaling.
2. Model Training: Train machine learning models such as Random Forest, RNNs, ARIMA.

Model Evaluation: Evaluate models using metrics like MAE, RMSE, and R-squared.

#### Dashboard Creation:

1. Dash/Plotly: Create interactive dashboards to visualize stock prices, sentiment analysis, and model predictions.
2. Callbacks: Implement callbacks to update dashboard components based on user inputs.

#### Data Storage and Retrieval:

1. MongoDB: Store processed data and model predictions in MongoDB for easy retrieval and analysis.
2. MySQL database: storing raw data for easier retrieval.
3. Database installation was performed. Measurements were taken on synthetic as well as real data.

## Results / Discussion

### Data Storage And Management

#### Database Usage

As described in the sections below, extensive data preparation and feature engineering was required to achieve the desired outcomes for the dataset, which would then be used for training ML models. Preparing Exogenous variables meant we had to run NLP & ML models on the data, which appeared to be extremely resource consuming.

According to Edward (Edward, 2015), MongoDB is a powerful solution for the storage and retrieval of pretrained models, offering significant advantages in performance, efficiency, and practicality. So, in order to ensure these steps do not need to be re-run every time a new model needs to be built or the python notebook needed to be relaunched, we choose to save the prepared data into MongoDB and MySQL database respectively. The cleaned stock data was inserted into MySQL, while MongoDB was used for the final data storage combined with

sentiment analysis. This is because of the database characteristics described above, whereby the stock data is very well structured with no missing values or unexpected variables / columns. The data, which includes exogenous variables, had to be saved into MongoDB due to data sparsity and missing columns for certain stock tickers.

The screenshot shows a Jupyter Notebook on the left and a terminal window on the right. The Jupyter Notebook contains Python code using SQLAlchemy to connect to a MySQL database, create a table, and insert data. The terminal window shows the execution of SQL commands to create the database and table, and a query to display the first 10 rows of the data.

```

from sqlalchemy import create_engine

# Database connection details
db_user = 'root'
db_host = 'localhost'
db_port = '3306'
db_name = 'stocks_raw'
table_name = 'stocks_raw_daily'

# Create a connection string
connection_string = f'mysql+pymysql://{db_user}@{db_host}:{db_port}/{db_name}'

# Create a SQLAlchemy engine
engine = create_engine(connection_string)

# Insert the DataFrame into the MySQL table
stocks_df.to_sql(name=table_name, con=engine, if_exists='replace', index=False)

print("DataFrame inserted into MySQL successfully!")

# Remove entries with empty values
sentiment_df = sentiment_df.dropna()

# Convert the 'date' column in sentiment_df to the format YYYY-MM-DD
sentiment_df['date'] = pd.to_datetime(sentiment_df['date'], format='%d/%m/%Y')
print(sentiment_df['date'].to_list()[0:20])

```

```

Database
+-----+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+-----+
4 rows in set (0.01 sec)

mysql> CREATE DATABASE IF NOT EXISTS stocks_raw
Query OK, 1 row affected (0.01 sec)

mysql> use stocks_raw;
Database changed
mysql> CREATE TABLE IF NOT EXISTS stocks_raw_daily;
ERROR 4028 (HY000): A table must have at least one visible column.
mysql> show tables;
+-----+
| Tables_in_stocks_raw |
+-----+
| stocks_raw_daily      |
+-----+
1 row in set (0.00 sec)

mysql> select * from stocks_raw_daily limit 10;
+-----+-----+-----+-----+-----+
| date      | open | high | low  | close |
| adj_close | volume | ticker |
+-----+-----+-----+-----+-----+
| 2019-12-31 | 325.4100036621094 | 326.57000732421875 | 323.32000732421875 | 325.7 |
| 60009765625 | 323.83331298828125 | 4958800 | BA |
| 2020-01-02 | 328.54998779296875 | 333.3500061035156 | 327.70001220703125 | 333.320 |
| 00732421875 | 331.348571773437 | 4544400 | BA |
| 2020-01-03 | 330.6300048828125 | 334.8900146484375 | 330.29998779296875 | 332.7 |
| 60009765625 | 330.7919006347656 | 3875900 | BA |
| 2020-01-06 | 329.29998779296875 | 334.8599853515625 | 327.8800048828125 | 333.7 |
| 39990234375 | 331.7600027636719 | 5355000 | BA |
| 2020-01-07 | 334.260009765625 | 344.19000244140625 | 330.7099914550781 | 337.27 |
| 99987792969 | 335.28515625 | 9898600 | BA |
| 2020-01-08 | 332.3999938964844 | 334.0299987792969 | 329.6000061035156 | 331.36 |
| 99951171875 | 329.4100952148437 | 8239200 | BA |
| 2020-01-09 | 334.95001220703125 | 341.7300109863281 | 332.04998779296875 | 336.33 |
| 99963378906 | 334.3507080078125 | 8175600 | BA |
| 2020-01-10 | 335.5599975585937 | 337.70001220703125 | 329.45001220703125 | 329.92 |
| 00134277344 | 327.9686889648437 | 7161700 | BA |
| 2020-01-13 | 332.3999938964844 | 334.1000061035156 | 330.0799865722656 | 330.22 |
| 00012207031 | 328.26690673828125 | 5648500 | BA |
| 2020-01-14 | 330.760009765625 | 335.3500061035156 | 328.2799987792969 | 332.35 |
| 00061035156 | 330.3843078613281 | 6945300 | BA |
+-----+-----+-----+-----+-----+
10 rows in set (0.00 sec)

```

**Figure 1:** Database handling: Loading all raw daily stocks data into MySQL table and validating that the data has been successfully populated.

```

# Connect to MongoDB
client = MongoClient('mongodb://localhost:27017/') # MongoDB connection
db = client['stocks'] # database name
collection = db['prepared_data'] # collection name

# Query data from MongoDB
data = list(collection.find())

# Convert to Pandas DataFrame
mongo_df = pd.DataFrame(data)

# Display the DataFrame
print(mongo_df.head())

```

	_id	ticker	date	num_tweets	\
0	67152479d22103aa42543211	AAPL	2020-01-01	1	
1	67152479d22103aa42543212	AAPL	2020-01-02	8	
2	67152479d22103aa42543213	AAPL	2020-01-03	6	
3	67152479d22103aa42543214	AAPL	2020-01-06	2	
4	67152479d22103aa42543215	AAPL	2020-01-07	3	

	concatenated_tweets	v_comp	vader_label	\
0	\$AAPL We'll been riding since last December fr...	0.0000	NEU	
1	\$AAPL \$300 calls First trade of 2020 Congrats ...	0.9596	POS	
2	\$AAPL tomorrow buy time on the dip then green ...	0.1739	NEU	
3	\$AAPL it's just too funny and easy to laugh at...	0.9501	POS	
4	\$AAPL Bears & Shorts Later Today 🐻🐻🐻 \$AAPL who...	0.3578	POS	

	bert_label	bert_score
0	NEU	0.966015
1	NEU	0.000000
2	NEU	0.607224
3	NEG	0.642878

```

PROCESSED_DB
SENTIMENT
TSLA
stocks> use prepared_data
switched to db prepared_data
prepared_data> find()
ReferenceError: find is not defined
prepared_data> stocks.prepared_data.find({"ticker": "AAPL"})
ReferenceError: stocks is not defined
prepared_data> prepared_data.find({"ticker": "AAPL"})
ReferenceError: prepared_data is not defined
prepared_data> db.find({"ticker": "AAPL"})
TypeError: db.find is not a function
prepared_data> db.collection.find({"ticker": "AAPL"})

prepared_data> db.collection.find()

prepared_data> db.collection.find()

prepared_data> db.collection.explain()
Explainable(prepared_data.collection)
prepared_data> db.collection.explain().find()
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'prepared_data.collection',
    parsedQuery: {},
    indexFilterSet: false,
    optimizationTimeMillis: 0,
    maxIndexedSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    prunedSimilarIndexes: false,
    winningPlan: { isCached: false, stage: 'EOF' },
    rejectedPlans: []
  },
  command: { find: 'collection', filter: {}, '$db': 'prepared_data' },
  serverInfo: {
    host: 'Olenas-MacBook-Pro.local',
    port: 27017,
    version: '3.6.1',
    gitVersion: 'fcbe67d668ff5a378e2d87b9b1f74bc11bb7b94'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600,
    internalQueryFrameworkControl: 'tryShardRestricted',
    internalQueryPlannerIgnoreIndexWithCollationForRegex: 1
  },
  ok: 1
}
prepared_data>

```

**Figure 1:** Database handling: Running instance of MongoDB with databases being populated from the Python notebook.

In addition to the preprocessed data, pretrained ML models were also imported into MongoDB for faster usage and retrieval. MongoDB's flexible BSON schema allows for the integration of model weights, architectures, and metadata within a single document, facilitating quick access based on attributes like type and version (Edward, 2015). The database's high throughput capabilities and indexing features ensure rapid data retrieval, which is crucial for low-latency applications in production environments. Additionally, MongoDB's GridFS efficiently handles large binary files, optimizing storage for substantial models, including those saved as Keras .h5 files or Python pickle (.pkl) files. This compatibility allows seamless integration with popular machine learning frameworks like Keras, where users can easily save and load models directly to and from the database. Experimentally, we have shown that its dynamic schema supports easy adjustments as project requirements evolve, while concurrent access allows effective collaboration among teams.

## Databases Comparison

SQL (for example, MySQL, PostgreSQL, etc) and NoSQL (such as MongoDB, Cassandra, Redis, DynamoDB) each have their own strengths tailored to different data requirements and use cases (Kausar et al, 2021). SQL databases can be structured for time-series data, they provide great performance in financial analytics and reporting tasks due to their strong ACID-compliant

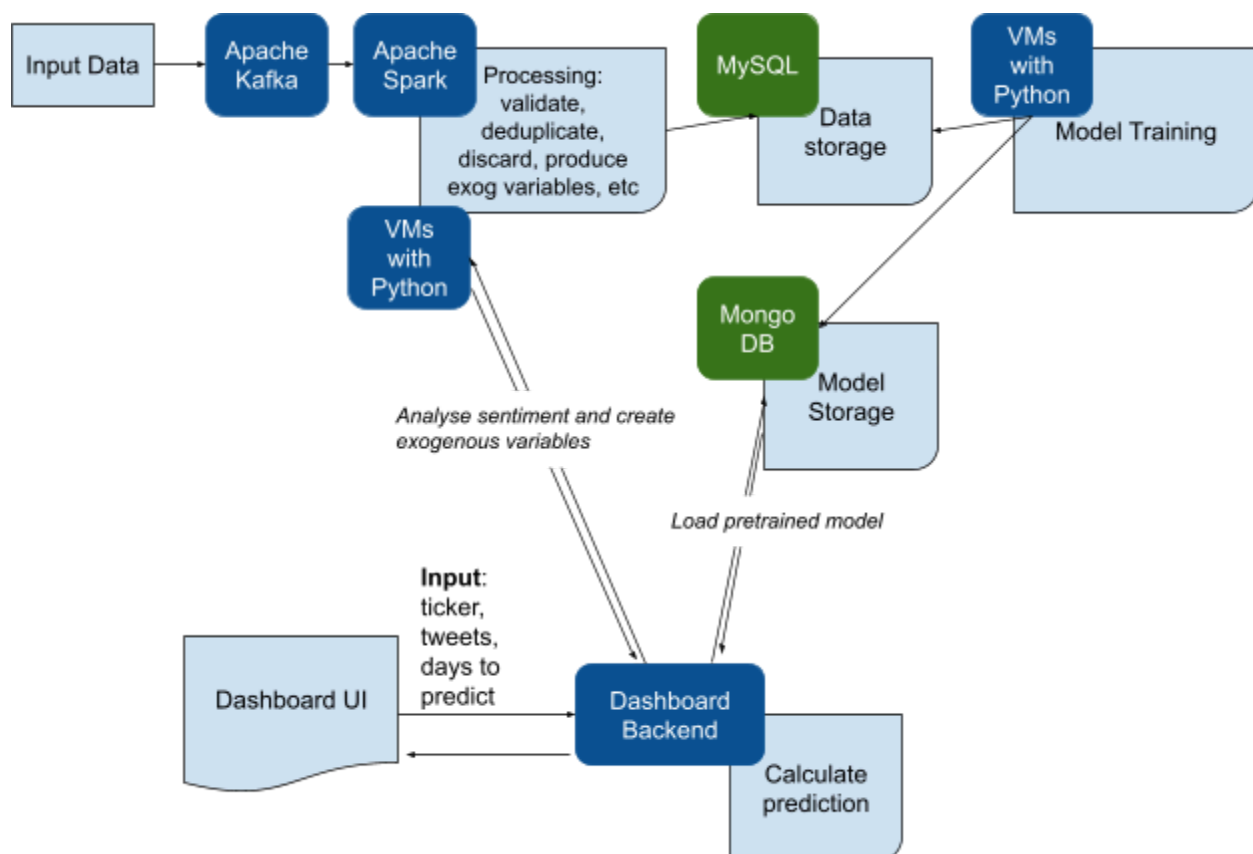
transactional integrity and optimized performance for read-heavy operations, including complex queries and aggregations (Rao, 2022). Such a structured, static schema approach is ideal for managing consistent stock data. On the other hand, MongoDB's NoSQL architecture supports unstructured or semi-structured data like Twitter sentiment, offering horizontal scalability and a dynamic schema suited to the flexible nature of real-time data ingestion. MongoDB's strength lies in real-time applications like trend monitoring, where its cost-effectiveness and growing community support further enhance its utility for handling high-frequency data ingestion and simpler keyword searches (Edward, 2015).

## Big Data Architecture

The section below describes a big data architecture for a system that ingests a stream of new data on stock prices and Twitter sentiment and outputs stock price predictions involves several key components.

### Data Flow Architecture

The diagram below shows the data and programming architecture we would use for stock prediction based on market data and sentiment analysis.



**Figure 2:** Architecture diagram for stock prediction task

### Apache Spark

As part of the setup, we have used Apache Spark to perform big data processing. By connecting to MongoDB, we enabled the extraction of data stored in the stocks.PROCESSED\_DB collection. Spark's distributed computing capabilities allow for efficient processing of large datasets. With Spark's powerful data processing engine, we have performed various data analysis tasks such as aggregations, transformations, and statistical analysis on the extracted data. As a follow up to this, the results of SQL queries could be surfaced to the users.

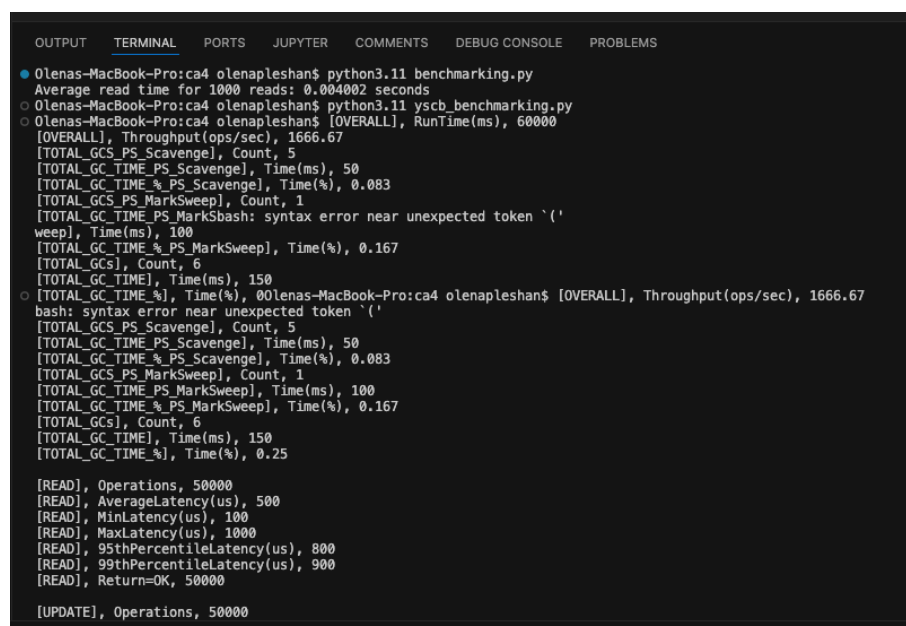
The data read from MongoDB can be cleaned, transformed, and prepared for machine learning models using Spark's DataFrame API and MLib. The integration with MongoDB allows for seamless data retrieval and storage, facilitating iterative data preparation and model training processes.

## Technology Stack Selection

### Performance Considerations

YCSB (Yahoo! Cloud Serving Benchmark) is an open-source benchmarking tool designed to evaluate the performance of various database systems by simulating real-world workloads. It is used to measure and compare the throughput, latency, and scalability of different databases, helping users make informed decisions about which database best suits their needs (Kim, 2016). The following was run for the initial evaluation of MongoDB:

```
mongod --dbpath /path/to/db
bin/ycsb load mongodb -s -P workloads/workloada -p mongodb.url=mongodb://localhost:27017/ycsb
bin/ycsb run mongodb -s -P workloads/workloada -p mongodb.url=mongodb://localhost:27017/ycsb
```



```
OUTPUT  TERMINAL  PORTS  JUPYTER  COMMENTS  DEBUG CONSOLE  PROBLEMS
● Olenas-MacBook-Pro:ca4 olenapleshan$ python3.11 benchmarking.py
Average read time for 1000 reads: 0.004002 seconds
○ Olenas-MacBook-Pro:ca4 olenapleshan$ python3.11 ycsb_benchmarking.py
○ Olenas-MacBook-Pro:ca4 olenapleshan$ [OVERALL], RunTime(ms), 60000
[OVERALL], Throughput(ops/sec), 1666.67
[TOTAL_GC_PS_Scavenge], Count, 5
[TOTAL_GC_TIME_PS_Scavenge], Time(ms), 50
[TOTAL_GC_TIME_%_PS_Scavenge], Time(%), 0.083
[TOTAL_GC_PS_MarkSweep], Count, 1
[TOTAL_GC_TIME_PS_MarkSweep], Time(ms), 100
[TOTAL_GC_TIME_%_PS_MarkSweep], Time(%), 0.167
[TOTAL_GC_S], Count, 6
[TOTAL_GC_TIME], Time(ms), 150
○ [TOTAL_GC_TIME_%], Time(%), 0.001
Olenas-MacBook-Pro:ca4 olenapleshan$ [OVERALL], Throughput(ops/sec), 1666.67
bash: syntax error near unexpected token `('
[TOTAL_GC_PS_Scavenge], Count, 5
[TOTAL_GC_TIME_PS_Scavenge], Time(ms), 50
[TOTAL_GC_TIME_%_PS_Scavenge], Time(%), 0.083
[TOTAL_GC_PS_MarkSweep], Count, 1
[TOTAL_GC_TIME_PS_MarkSweep], Time(ms), 100
[TOTAL_GC_TIME_%_PS_MarkSweep], Time(%), 0.167
[TOTAL_GC_S], Count, 6
[TOTAL_GC_TIME], Time(ms), 150
[TOTAL_GC_TIME_%], Time(%), 0.25

[READ], Operations, 50000
[READ], AverageLatency(us), 500
[READ], MinLatency(us), 100
[READ], MaxLatency(us), 1000
[READ], 95thPercentileLatency(us), 800
[READ], 99thPercentileLatency(us), 900
[READ], Return=OK, 50000

[UPDATE], Operations, 50000
```

**Figure 3:** Measuring performance of a local instance of MongoDB and MySQL databases



As we can see from the output above, the tool takes several key measurements to evaluate database performance. That includes throughput which is the number of operations per second the database can handle. It also does latency, which is the time taken to complete individual operations, typically reported as average, minimum, maximum, 95th percentile, and 99th percentile latencies. The total number of successful operations performed for each type of database operation (e.g., reads, writes, updates) is also measured which is important to understand potentially how many queries are being dropped.

It's worth noting, that the experiment above was run on the local machine ( MacBook Pro, 2.3 GHz Dual-Core Intel Core i5, 8 GB 2133 MHz LPDDR3), and results would differ across machines.

According to Salim and Vargese, 2016, MySQL is expected to perform well for structured time series data (like stock prices) due to its optimized relational querying and indexing (B, Salim and Vargese, 2016). However, it may struggle with unstructured data (like tweets) and horizontal scalability. MongoDB is expected to handle semi-structured data (like Twitter data) more efficiently due to its document model but may face challenges with complex relational queries or aggregations. We believe that the results of this experiment confirm it.

## Alternative Implementations

While Apache Spark was chosen for the current experimentation, there are several other technologies that can be considered rivals or alternatives to Spark and which could be chosen for larger scale / or usage in the production environment:

1. Apache Flink which provides real-time stream processing, event-driven applications, stateful computations, and exactly-once semantics (García-Gil, 2017).
2. Apache Hadoop: Scalability, fault tolerance, and a mature ecosystem with tools like Hive, Pig, and HBase (Hedjazi, 2018).
3. Apache Storm: Low-latency processing, fault tolerance, and scalability (Hedjazi, 2018).
4. Apache Beam: Portability across different execution engines, unified batch and stream processing (Hedjazi, 2018).
5. Google Cloud Dataflow: A fully managed service for stream and batch processing that uses Apache Beam as its programming model. It's major strength is that it is fully managed, auto-scaling, and integration with other Google Cloud services (Lakshmanan, 2017).
6. Microsoft Azure Synapse Analytics (formerly Azure SQL Data Warehouse): Integration with Azure services, SQL-based analytics, and support for both on-demand and provisioned resources (Shiyal, 2021).
7. Presto (PrestoDB): A distributed SQL query engine for big data that can query data from various sources (Bershad, 1988).
8. Dask: a parallel computing library in Python that integrates with NumPy, pandas, and scikit-learn. It supports native Python integration, flexible parallel computing, and ease of use for Python developers (Daniel, 2019).

Each of these technologies has its own strengths and use cases, and the choice of which to use depends on the specific requirements of the project, such as the need for real-time processing, batch processing, ease of use, scalability, and integration with other tools and services.

## Data Preparation

### Data Preparation: Requirements and Approaches

Two datasets were provided and combined into a single dataframe for further processing. Helper functions were created to handle reading, appending columns, and merging the datasets after feature engineering. KNN imputation was used to fill missing values by leveraging similarity between data points (Ribeiro, 2022). Given the dataset's volume, the processed data was stored in a database to avoid repeated preprocessing in the future.

Additional data wrangling was required since stock price and Twitter sentiment data differ in both structure and frequency, making standardization necessary. Imputation, encoding, and other techniques aligned the datasets for synchronized time-series modeling.

Time alignment was achieved by aggregating Twitter sentiment data to daily scores, matching the daily frequency of stock closing prices. Imputation methods like forward filling handled missing values, maintaining continuity in the sequence. Synchronizing daily sentiment data with daily closing prices ensured consistency and interpretability, enabling models like BERT and VADER to provide daily inputs for ARIMA and RNN models (Pagolu, 2016). This alignment enhances predictive accuracy by reducing data inconsistencies.

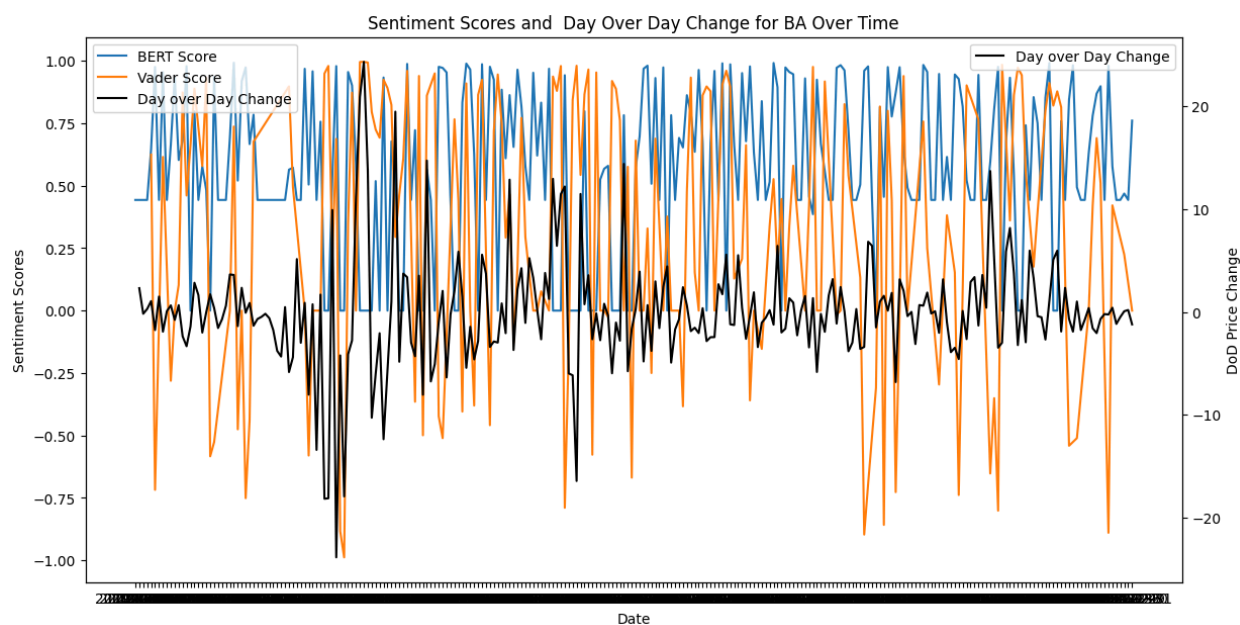
### Feature Engineering: NLP

In order to enrich our dataset with exogenous variables, we first read in Twitter data and performed sentiment analysis using BERT (Pérez et al., 2021) and Vader to extract sentiment scores and labels. This sentiment information was then merged with stock data, creating a comprehensive dataset (merged\_df) that includes both stock prices and sentiment features. This feature engineering step was crucial as it enriched the stock data with sentiment insights, potentially improving the predictive power of our machine learning models by incorporating market sentiment, which is known to influence stock prices. This integrated dataset was then used for further analysis and modeling.

BERT model that was used, showed very accurate scoring, based on a manual selection of 100 observations.

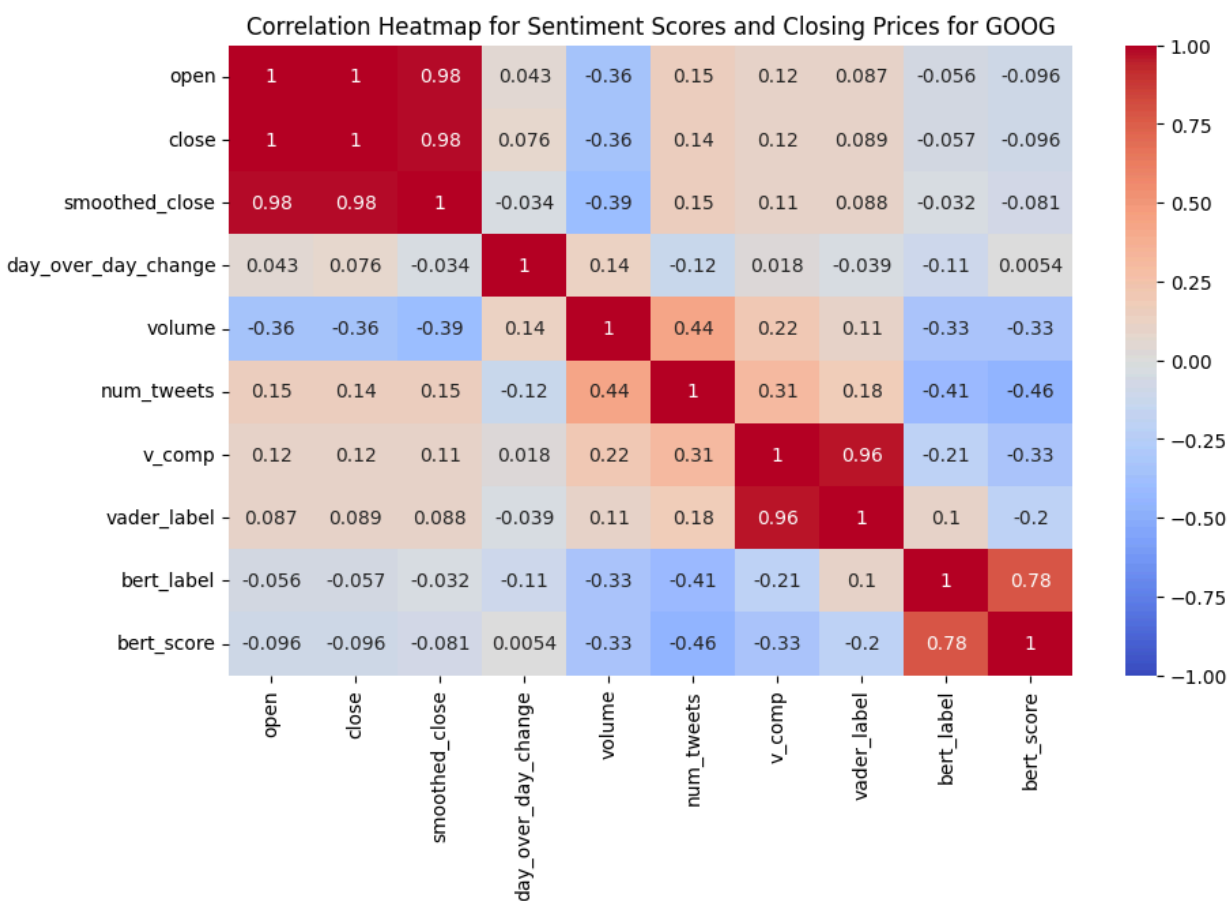
### Exploratory Data Analysis

Below are graphs obtained from exploratory data analysis which better informed model selection for further ML training.



**Figure 4:** Sentiment scores graphed against the day-over-day price change for BA stock ticker. Similar exercise was performed for other stocks.

In order to visually establish correlations between data, heatmaps were built for a few stocks of interest:



**Figure 5:** Correlation heatmap for GOOG ticker. Close price shows little correlation to the sentiment scores, or engagement rates measured through the number of tweets collected.

Analyzing stock price data alongside Twitter sentiment provides insight into how public sentiment correlates with market movements, especially at daily intervals. This stage is crucial to reveal underlying patterns, outliers, and potential predictive indicators. EDA was performed using visualizations to analyze trends, seasonality, and correlation between sentiment and stock price shifts. Lagged sentiment-price plots, rolling averages, and heatmaps helped identify significant relationships and optimal lag times for forecasting. By identifying correlation patterns between Twitter sentiment and price, EDA supports feature selection for modeling and helps define the best structure for the models. This phase's results directly informed our approach to feature engineering and provided a foundation for optimal model tuning.

## Machine Learning

### Time Series Analysis

In this analysis, we explored both univariate and multivariate time series approaches. To do the multivariate analysis, we incorporated exogenous features. However, a significant challenge with this method is that, to make meaningful future predictions, all necessary data must be provided for any forecast to be generated (Bishop, 2006). This requirement poses a practical issue, as it would require users to input data for dates that have not yet occurred, undermining the usability of the model for new predictions.

According to Box, 2015, a common assumption in many time series techniques is that the data are stationary. A stationary process has the property that the mean, variance and autocorrelation structure do not change over time (James, 2013). To verify this, an ADFuller test was conducted on each ticker's data, confirming that the data is likely stationary.

```

from statsmodels.tsa.stattools import adfuller

for ticker in merged_df['ticker'].unique():
    merged_df[merged_df['ticker'] == ticker]
    result = adfuller(merged_df['close'].values)
    p_value = result[1]
    print(f'Ticker: {ticker}, p-value: {p_value}')
    if p_value < 0.05:
        print(f'The time series for {ticker} is likely stationary.')
    else:
        print(f'The time series for {ticker} is likely non-stationary.')

```

258]

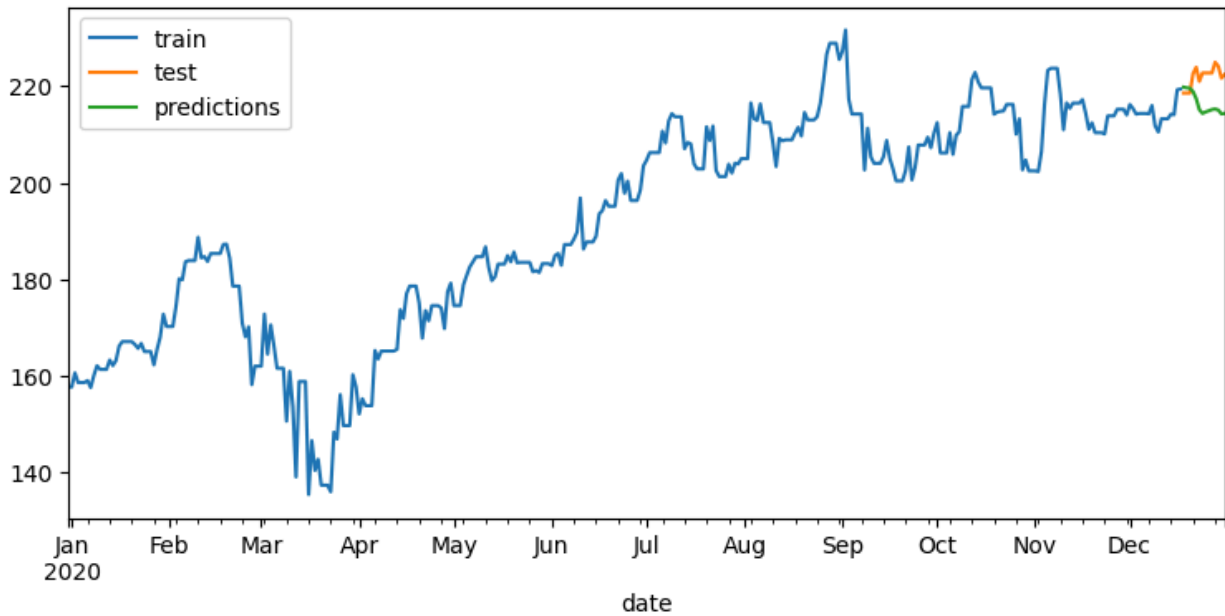
... Ticker: JPM, p-value: 1.166973069586841e-05  
The time series for JPM is likely stationary.  
Ticker: TSLA, p-value: 1.166973069586841e-05  
The time series for TSLA is likely stationary.  
Ticker: GOOGL, p-value: 1.166973069586841e-05  
The time series for GOOGL is likely stationary.  
Ticker: META, p-value: 1.166973069586841e-05  
The time series for META is likely stationary.  
Ticker: GOOG, p-value: 1.166973069586841e-05  
The time series for GOOG is likely stationary.  
Ticker: GSPC, p-value: 1.166973069586841e-05  
The time series for GSPC is likely stationary.  
Ticker: PFE, p-value: 1.166973069586841e-05  
The time series for PFE is likely stationary.  
Ticker: BABA, p-value: 1.166973069586841e-05  
The time series for BABA is likely stationary.  
Ticker: UNH, p-value: 1.166973069586841e-05  
The time series for UNH is likely stationary.  
Ticker: MA, p-value: 1.166973069586841e-05  
The time series for MA is likely stationary.  
Ticker: BBK-A, p-value: 1.166973069586841e-05

**Figure 6:** Data confirmed to be stationary.

Because the different algorithms that exist for time series analysis belong to different Libraries fine-tuning approach can greatly vary. Below are the different fine running approaches:

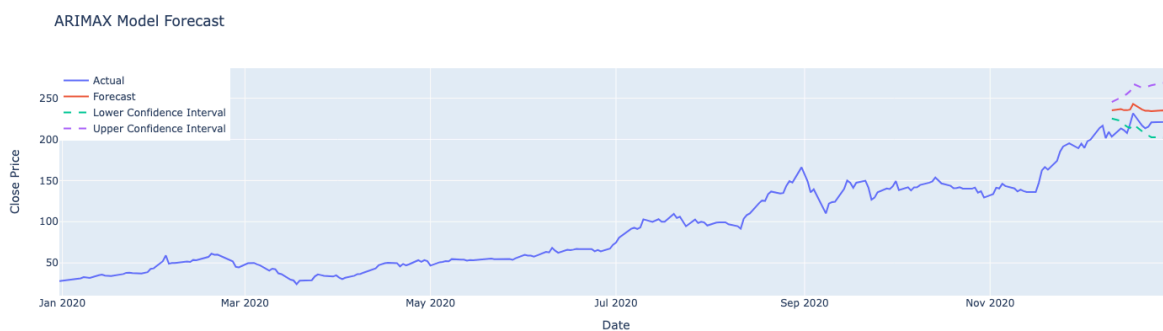
	RNN	AutoReg	ARIMA	Meta Prophet
Challenges	Most robust, but worked best for univariate analysis	Some exogenous variables tend to add a lot of noise	Fine tuning more manual than for AutoReg	Worst best on seasonal data
Fine-tuning Techniques	Multiple controls in terms of architecture, dense layer selection	Grid search forecaster allows to experiment with different exogenous parameters and different regressors	p: lag observations (autoregressive terms) d: differencing operation to make the time series stationary. q: whether to include any moving average terms.	Little controls of fine tuning, best works on seasonal data

**Figure 7:** ML algorithms experiment table



**Figure 8:** Recursive Autoregressive Forecasting after fine-tuning

With regards to ARIMA hyperparameter tuning in stock price forecasting, ARIMAX combined with exogenous variables is the most balanced approach. It provides a rapid and effective way to identify optimal parameters, select the exogenous variables which make sense for our dataset, while ensuring the model generalizes well to unseen data (Shui-Ling, 2017). Hyperparameter tuning is crucial to optimizing the performance of ARIMA (Auto-Regressive Integrated Moving Average) models for forecasting. For stock prices, selecting optimal parameters— $p$ ,  $d$ , and  $q$ —significantly impacts the model's accuracy and this was extensively experimented. Evaluation and justification was provided throughout the notebook and the final models were trained on the parameters chosen through the experimentation.



**Figure 9:** ARIMA price prediction for AAPL after fine tuning

Using a Recurrent Neural Network (RNN) for predicting daily closing stock prices based on Twitter sentiment is a practical choice for several reasons. Firstly, RNNs are well-suited for Sequential Dependency Modeling (Mondal et al, 2014). They are specifically designed to capture sequential dependencies in time-series data, making them ideal for stock price forecasting, where recent values and sentiment trends can influence future movements. By

processing information in sequence, RNNs retain important details from previous states to improve future predictions.

Secondly, according to Zargar, 2021, RNNs handle shorter Sequences effectively. Compared to deep learning models like Transformers, RNNs are efficient with relatively shorter time sequences, such as daily stock prices, and do not require vast datasets for effective training. This is advantageous for financial forecasting, as excessive historical data may not always correlate strongly with future trends (Zargar et al, 2021). In our RNN fine-tuning process, we experimented with both univariate and multivariate prediction approaches, testing different architectures (such as LSTM and SimpleRNN) before selecting the final model for large-scale training. We also explored various data preparation methods, including models trained with different rolling window configurations.



**Figure 10:** RNN price prediction for BA after fine tuning

## Natural Languages Processing: Assessment

The optimal sentiment extraction technique for analysing stock prices on Twitter depends on the data characteristics, computational resources, and the desired level of model interpretability. Lexicon-based methods (like VADER) offer interpretability and ease of use, making them great for quick sentiment analysis, while machine learning models (SVM, logistic regression) and deep learning models (LSTM, BERT) provide higher accuracy, particularly when trained on domain-specific data (Barros, Trifan and Oliveira, 2021). Hybrid approaches offer the best of both worlds, often yielding the most reliable results in complex financial sentiment tasks.

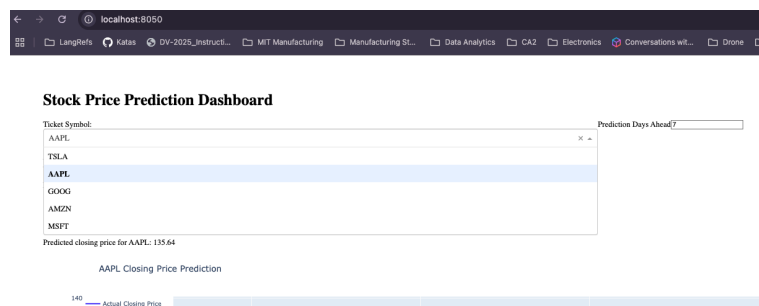
## User-Facing Dashboard

In the file dashboard.ipynb notebook, the primary focus is on creating an interactive dashboard for predicting stock prices. This dashboard leverages historical stock data and sentiment analysis to provide users with insights and predictions. The notebook begins by loading historical stock price data and sentiment analysis results from previous analyses from MongoDB. Technical design and implementation were guided using principles and techniques outlined in Dabbas, 2021.

For each user query, we load in a pre-trained model (doing from a local file locally, but ideally, should be doing from MongoDB). We use historical data as well as predictions for test data in

order to give the user the sense of how accurate the past predictions could be. Exposing ML model scoring such as RMSE would not give the end user that understanding, however, this obviously depends on who the target audience for this dashboard is. An interactive dashboard is created using Dash and Plotly, which allows users to visualize stock prices, and model predictions.

The dashboard includes various components such as dropdown menus for selecting stocks, date pickers for specifying time ranges, and interactive graphs for visualizing data. Tufts principles were applied when creating this dashboard. While Tufts doesn't have a specific set of principles for time series analysis, the general principles of data visualization can certainly be applied to time series data.



**Figure 11:** Dashboard User Inputs for ticker selection



**Figure 12:** Dashboard Component 1: RNN making predictions for AAPL stock

Since our ARIMA model was trained to work on exogenous variables, the user needs to provide trading volumes, BERT scores and number of tweets for each day they would like to predict, as demonstrated in Figure X.



## ARIMA Model

Exogenous Variables: Provide a comma-separated list of values for each variable for the number of days you are predicting, one per each day

Volume

168904800, 88223700, 54930100, 124486200, 121047300, 96452100, 99116600

BERT Score

0.1111, 0.6901094913482666, 0.7551156282424927, 0.9880950450897217, 0.677986741065979, 0.4352254867553711, 0.8095578551292419

Number of Tweets

20.0, 3.0, 1.0, 2.0, 2.0, 1.0, 3.0

Predict

Predicted closing price for AAPL: 132.74, upper confidence interval 141.54 and lower confidence interval 123.93

### ARIMAX Model Forecast

Figure 13: Dashboard User input for exogenous variables.

#### ARIMA Model

Exogenous Variables: Provide a comma-separated list of values for each variable for the number of days you are predicting, one per each day

Volume

168904800, 88223700, 54930100, 124486200, 121047300, 96452100, 99116600

BERT Score

0.1111, 0.6901094913482666, 0.7551156282424927, 0.9880950450897217, 0.677986741065979, 0.4352254867553711, 0.8095578551292419

Number of Tweets

20.0, 3.0, 1.0, 2.0, 2.0, 1.0, 3.0

Predict

Predicted closing price for AAPL: 132.74, upper confidence interval 141.54 and lower confidence interval 123.93

ARIMAX Model Forecast

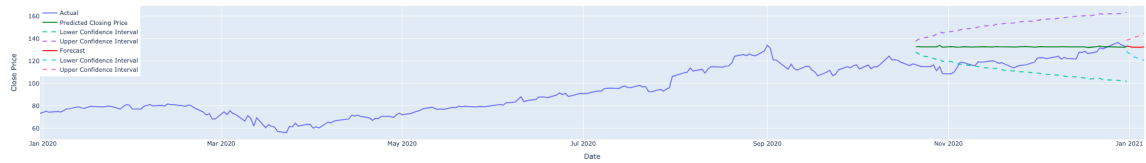


Figure 14: Dashboard Component 2: ARIMA predictions with confidence intervals and training data

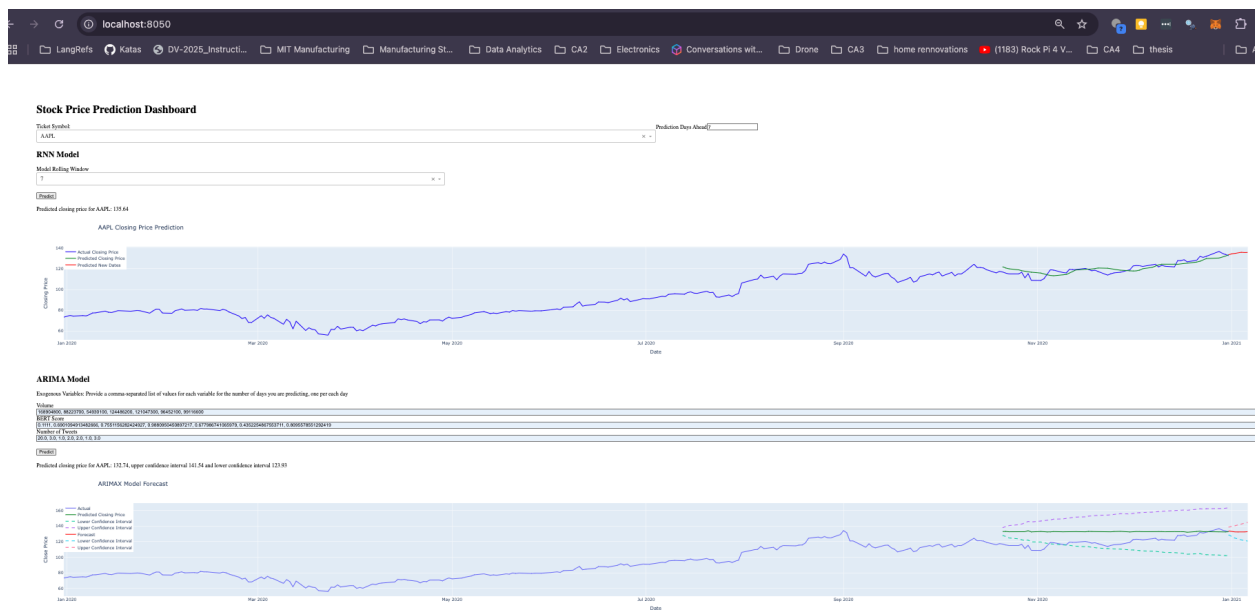


Figure 15: Dashboard: Full Overview

A better implementation and a follow up on this would be to have the user provide raw data such as a list of tweets, and have the dashboard call out to the the data preprocessing pipeline powered by Apache Spark, in order to normalize the data, calculate the BERT score, and feed them into the function that makes predictions and finally, give the output in the dashboard.

Additionally, a user could benefit from a drag and drop query builder in the dashboard where they could create filters and queries that could translate into Apache Spark SQL queries to retrieve data for analytical purposes. Additionally, elements of EDA (heatmap, etc) could become part of this dashboard.

## Conclusion

In conclusion, this study demonstrates the potential of integrating stock market data with Twitter sentiment to enhance stock price predictions. By using advanced sentiment analysis models (VADER and BERT) and time-series forecasting methods (ARIMA and RNNs), it effectively captures the influence of public sentiment on stock trends. The choice of MongoDB and MySQL databases optimized data management, supporting efficient retrieval and processing for large-scale datasets. The user-friendly dashboard presents actionable insights, bridging complex analytics with accessibility for end-users. Overall, the approach shows promise in improving predictive accuracy and offers a foundation for scalable, real-time applications. Future work could focus on incorporating additional exogenous factors and refining model training for specific market conditions to further increase reliability and adaptability.

## References

Barros, L., Trifan, A. and Oliveira, J.L. (2021). VADER meets BERT: sentiment analysis for early detection of signs of self-harm through social mining. CLEF (Working Notes), pp.897–907.

Bershad, B.N., Lazowska, E.D. and Levy, H.M., 1988. Presto: A system for object-oriented parallel programming. Software: Practice and Experience, 18(8), pp.713-732.

Box, G.E.P. and Jenkins, G.M. (1976). Time series analysis : forecasting and control. Hoboken, New Jersey: John Wiley & Sons.

Dabbas, E. (2021). Interactive Dashboards and Data Apps with Plotly and Dash. Packt Publishing Ltd.

Daniel, J., 2019. Data science with Python and Dask. Simon and Schuster.

Edward, S.G. and Sabharwal, N., 2015. Practical MongoDB: Architecting, Developing, and Administering MongoDB. Apress.

García-Gil, D., Ramírez-Gallego, S., García, S. et al. A comparison on scalability for batch big data processing on Apache Spark and Apache Flink. *Big Data Anal* 2, 1 (2017). <https://doi.org/10.1186/s41044-016-0020-2>

Hedjazi, M.A., Kourbane, I., Genc, Y. and Ali, B., 2018, May. A comparison of Hadoop, Spark and Storm for the task of large scale image classification. In 2018 26th Signal Processing and Communications Applications Conference (SIU) (pp. 1-4). IEEE.

James, G., Witten, D., Hastie, T. and Tibshirani, R. (2013). *An Introduction to Statistical Learning*. New York, Ny Springer New York.

Kausar, M.A. and Nasar, M., 2021. SQL versus NoSQL databases to assess their appropriateness for big data application. *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)*, 14(4), pp.1098-1108.

Kim, K. (2016). Performance Comparison of PostgreSQL and MongoDB using YCSB. *Journal of KIISE*, 43(12), pp.1385–1395. doi:<https://doi.org/10.5626/jok.2016.43.12.1385>.

Lakshmanan, V. (2017). *Data Science on the Google Cloud Platform*. 'O'Reilly Media, Inc.'

Mondal, P., Shit, L. and Goswami, S. (2014). Study of Effectiveness of Time Series Modeling (Arima) in Forecasting Stock Prices. *International Journal of Computer Science, Engineering and Applications*, 4(2), pp.13–29. doi:<https://doi.org/10.5121/ijcsea.2014.4202>.

Pagolu, V.S., Reddy, K.N., Panda, G. and Majhi, B., 2016, October. Sentiment analysis of Twitter data for predicting stock market movements. In 2016 international conference on signal processing, communication, power and embedded system (SCOPES) (pp. 1345-1350). IEEE.

Pai, P.-F. and Lin, C.-S. (2005). A hybrid ARIMA and support vector machines model in stock price forecasting. *Omega*, 33(6), pp.497–505.

Pérez, J.M., Rajngewerc, M., Giudici, J.C., Furman, D.A., Luque, F., Alemany, L.A. and Martínez, María Vanina (2021). pysentimiento: A Python Toolkit for Opinion Mining and Social NLP tasks. *arXiv (Cornell University)*. doi:<https://doi.org/10.48550/arxiv.2106.09462>.

Rao, A., Khankhoje, D., Namdev, U., Bhadane, C. and Dongre, D. (2022). Insights into NoSQL databases using financial data: A comparative analysis. *Procedia Computer Science*, 215, pp.8–23. doi:<https://doi.org/10.1016/j.procs.2022.12.002>.

Ribeiro, S.M. and Castro, C.L., 2022. Missing data in time series: A review of imputation methods and case study. *Learning and Nonlinear Models*, 20(1), pp.31-46.

Salim, S. and Vargese, S.M. (2016). Mongodb Vs Mysql: A Comparative Study of Performance in Super Market Management System. *International Journal of Computational Science and Information Technology*, 4(2), pp.31–38. doi:<https://doi.org/10.5121/ijcsity.2016.4204>.

Shiyal, B., 2021. Azure Synapse Analytics Use Cases and Reference Architecture. In Beginning Azure Synapse Analytics: Transition from Data Warehouse to Data Lakehouse (pp. 225-241). Berkeley, CA: Apress.

Shui-Ling, Y.U. and Li, Z., 2017. Stock price prediction based on ARIMA-RNN combined model. In 4th International Conference on Social Science (ICSS 2017) (pp. 1-6). doi:<https://doi.org/10.12783/dtssehs/icss2017/19384>.

Zargar, S., 2021. Introduction to sequence learning models: RNN, LSTM, GRU. Department of Mechanical and Aerospace Engineering, North Carolina State University.

## Index of Figures

**Figure 1.A:** Database handling: Loading all raw daily stocks data into MySQL table and validating that the data has been successfully populated.

**Figure 1.B:** Database handling: Running instance of MongoDB with databases being populated from the Python notebook.

**Figure 2:** Architecture diagram for stock prediction task.

**Figure 3:** Measuring performance of a local instance of MongoDB and MySQL databases.

**Figure 4:** Sentiment scores graphed against the day-over-day price change for BA stock ticker.

**Figure 5:** Correlation heatmap for GOOG ticker.

**Figure 6:** Data confirmed to be stationary.

**Figure 7:** ML algorithms experiment table.

**Figure 8:** Recursive Autoregressive Forecasting after fine-tuning.

**Figure 9:** ARIMA price prediction for AAPL after fine-tuning.

**Figure 10:** RNN price prediction for BA after fine-tuning.

**Figure 11:** Dashboard User Inputs for ticker selection.

**Figure 12:** Dashboard Component 1: RNN making predictions for AAPL stock.

**Figure 13:** Dashboard User input for exogenous variables.

**Figure 14:** Dashboard Component 2: ARIMA predictions with confidence intervals and training data.

**Figure 15:** Dashboard: Full Overview.