Stock Market Analysis

Task 4 - Team Insighta

The Dataset

Features of the original dataset :

```
<class 'pandas.core.frame.DataFrame'>
Index: 11291 entries, 0 to 11290
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       11181 non-null  datetime64[ns]
 1   Adj Close  11198 non-null  float64
 2   Close      11174 non-null  float64
 3   High       11196 non-null  float64
 4   Low        11164 non-null  float64
 5   Open       11188 non-null  float64
 6   Volume     11146 non-null  float64
dtypes: datetime64[ns](1), float64(6)
memory usage: 705.7 KB
```

The Dataset:

	Date	Adj Close	Close	High	Low	Open	Volume
0	1980-03-17	2.296798	3.291227	3.344743	3.291227	0.000000	41109.0
1	1980-03-18	2.306134	3.304606	3.358122	3.304606	0.000000	9343.0
2	1980-03-19	2.306134	3.304606	3.304606	3.304606	3.304606	0.0
3	1980-03-20	2.306134	3.304606	3.358122	3.304606	0.000000	10277.0
4	1980-03-21	2.362154	3.384880	3.438396	3.384880	0.000000	8409.0
...	...	...	...	...	...	...	...
11286	2024-12-20	178.169998	178.169998	179.919998	175.839996	175.839996	425700.0
11287	2024-12-23	180.449997	180.449997	180.619995	177.970001	179.119995	422700.0
11288	2024-12-24	181.429993	181.429993	181.720001	180.830002	181.000000	168600.0
11289	2024-12-26	197.360001	197.360001	198.000000	193.130005	195.970001	1281200.0
11290	2024-12-27	199.520004	199.520004	201.000000	198.179993	200.360001	779500.0

11291 rows × 7 columns

Data Pre-Processing

Converted the ‘Date’ to a pandas DateTime object.

Contained null values in certain columns.

```
Date          110
Adj Close      93
Close         117
High           95
Low           127
Open          103
Volume        145
dtype: int64
```

Therefore dropped all rows containing null values in ‘Date’ and ‘Close’ attributes.

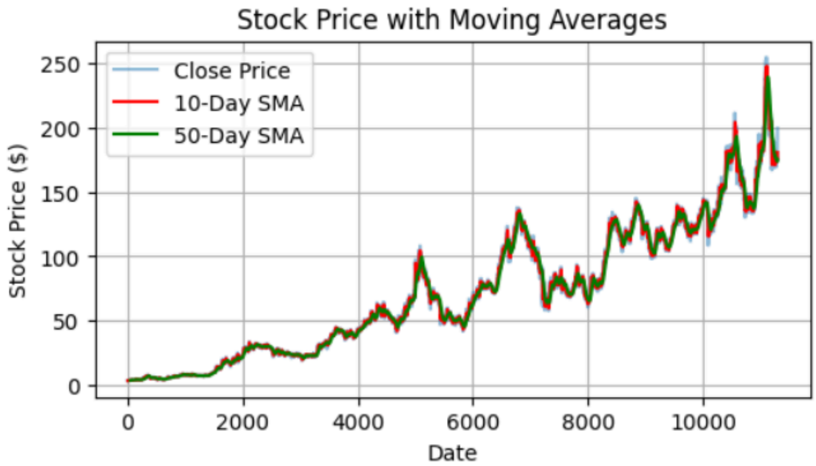
```
Before dropping (11291, 7)
After dropping: (11065, 7)
```

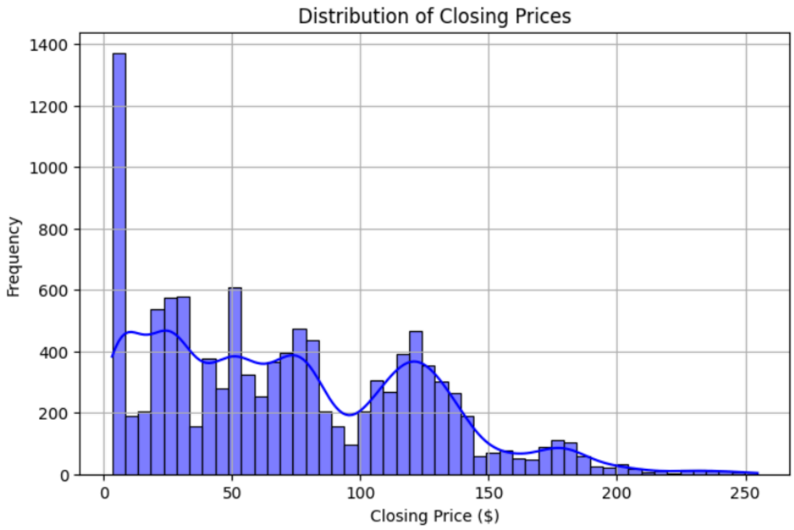
Visual Guide - Title and Explanation

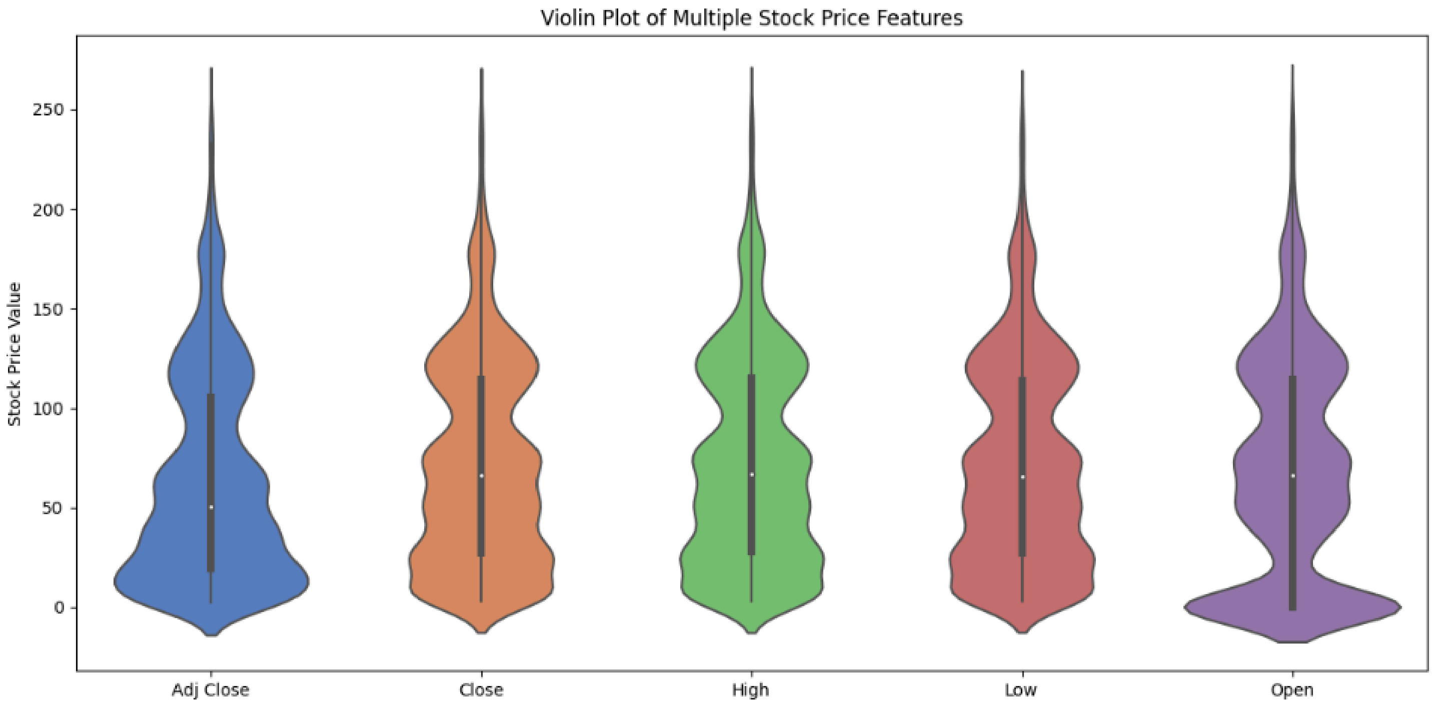
Original Features :

First sorted the data by date and set the index to Date.


Stock Closing Price Over Time
Close Price
Stock Price ($)
Date
0
50
100
150
200
250
0
2000
4000
6000
8000
10000


Stock Price with Moving Averages
Close Price
10-Day SMA
50-Day SMA
Stock Price ($)
Date
0
50
100
150
200
250
0
2000
4000
6000
8000
10000


Distribution of Closing Prices
Frequency
Closing Price ($)
0
200
400
600
800
1000
1200
1400
0
50
100
150
200
250



Handling Missing Values:

After the initial null value handling :

For my training I have chosen the Random Forest Model and XG Boost Model, therefore I handled the null values accordingly.

For Random Forest model I imputed the null values with forward fill method, since the the following day's price to maintain the temporal pattern of the data.

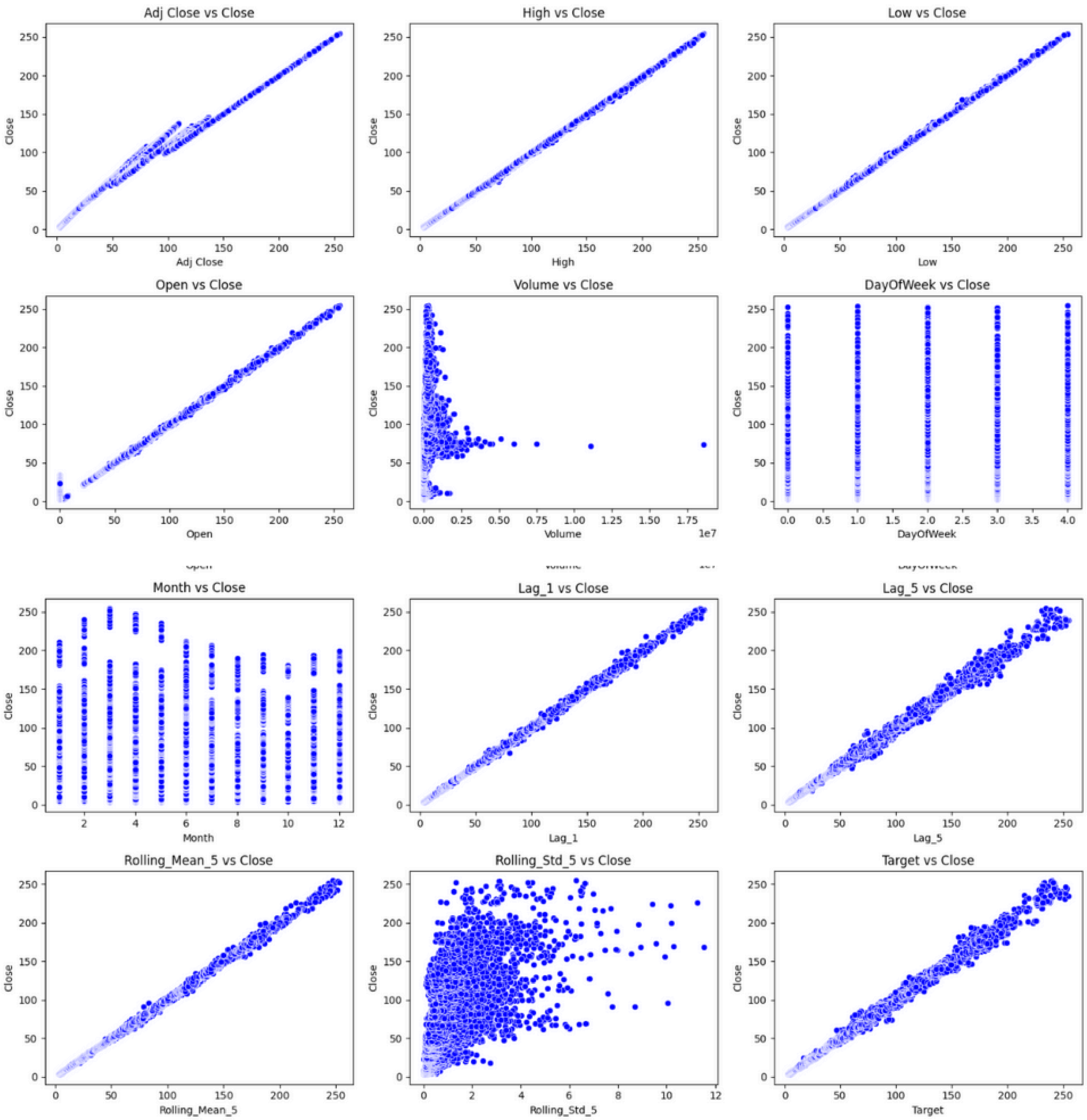
For XG Boost model I did not handle the null values explicitly, since the model handles null values by itself.

Feature Engineering:

Added the following features:

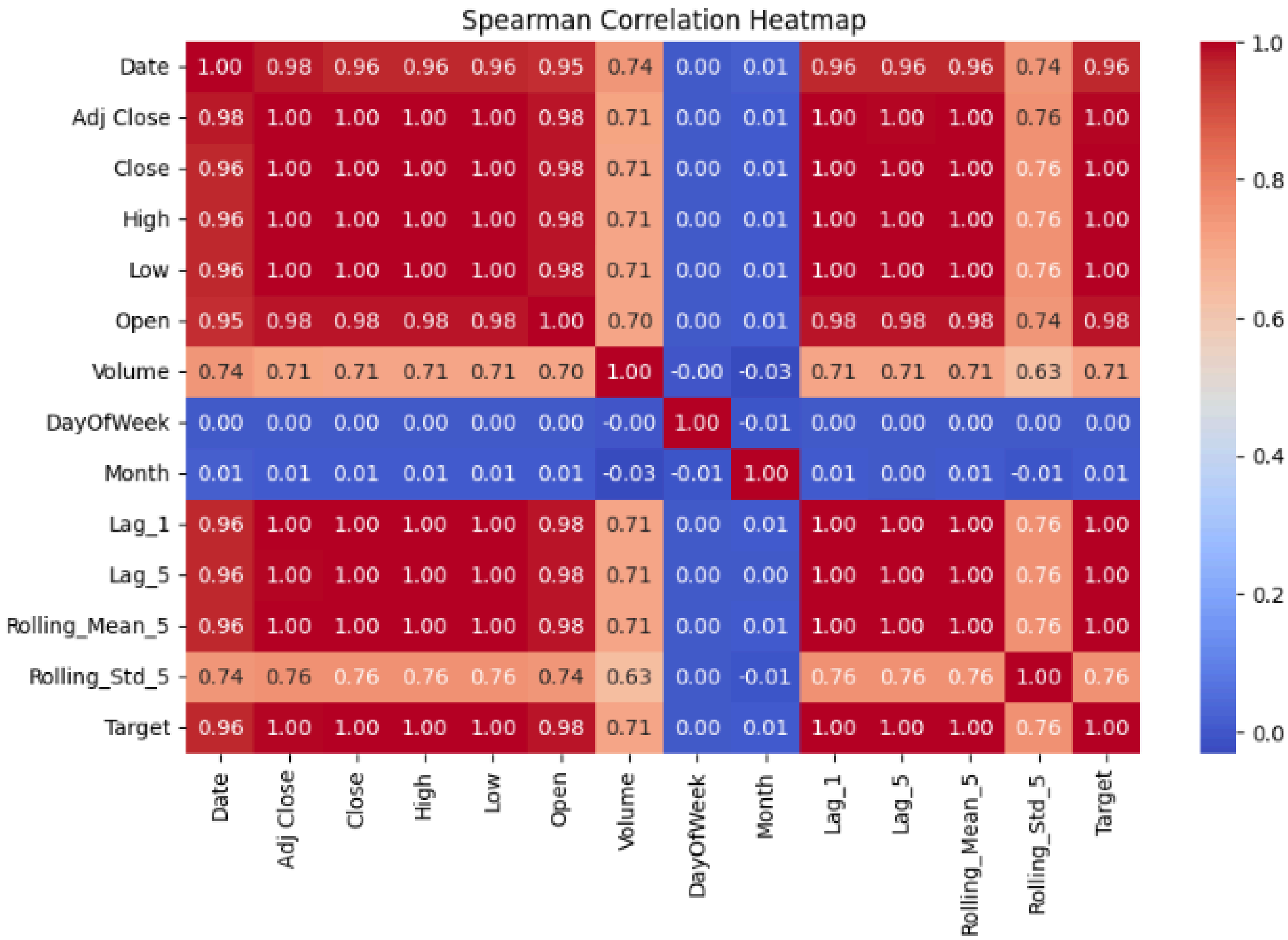
```python
# Feature Engineering
data_random['DayOfWeek'] = data_random['Date'].dt.dayofweek
data_random['Month'] = data_random['Date'].dt.month
data_random['Lag_1'] = data_random['Close'].shift(1)
data_random['Lag_5'] = data_random['Close'].shift(5)
data_random['Rolling_Mean_5'] = data_random['Close'].rolling(window=5).mean()
data_random['Rolling_Std_5'] = data_random['Close'].rolling(window=5).std()
```

Following is the visualizaton of the new features with 'Close' price.

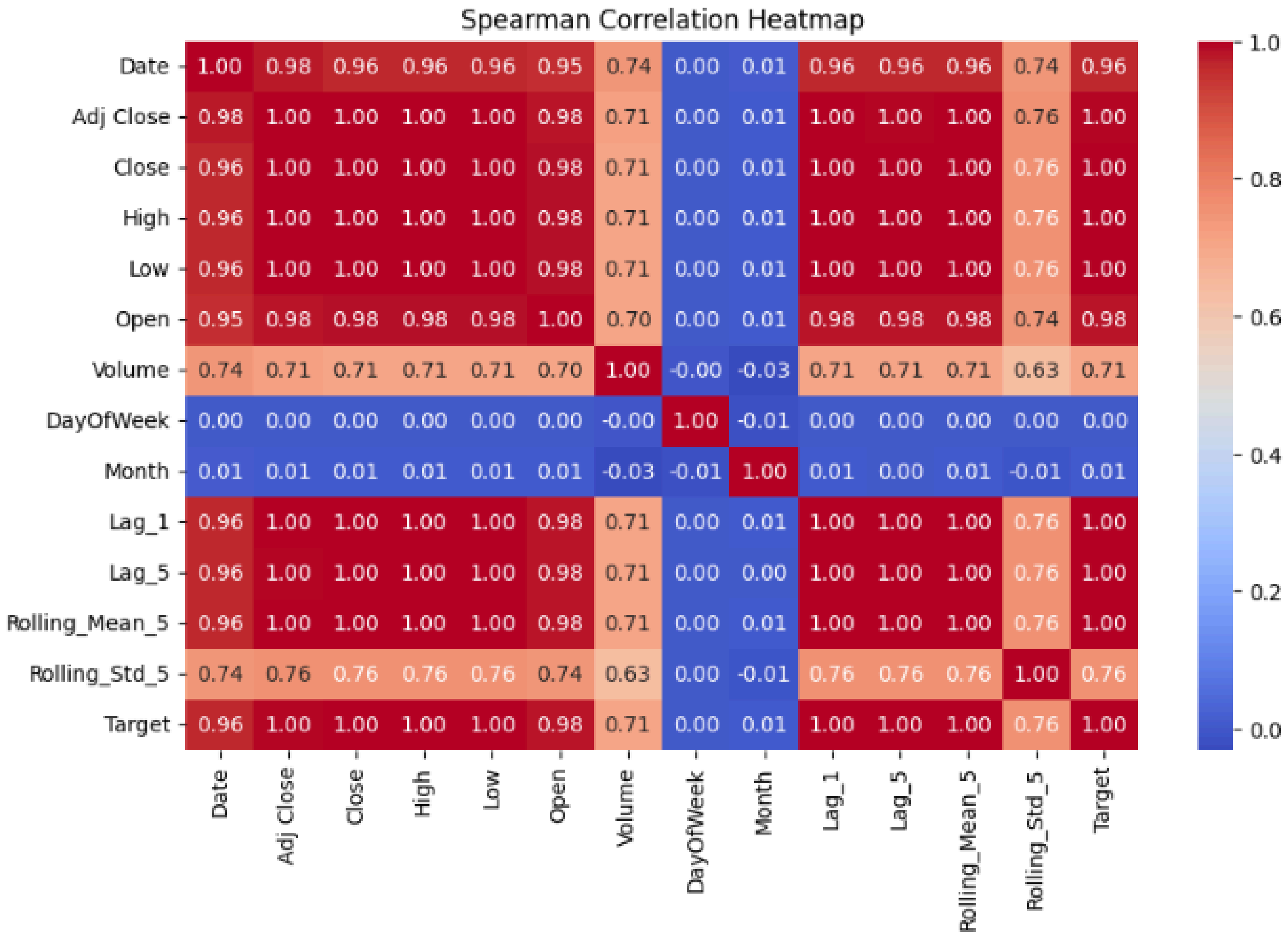


Following is the 'Spearman' correlation heatmap for the datasets.

(With imputed values)



(Without imputed values)



Chose Spearman's correlation coefficient to compare the correlation since the relationships in prices are linear.

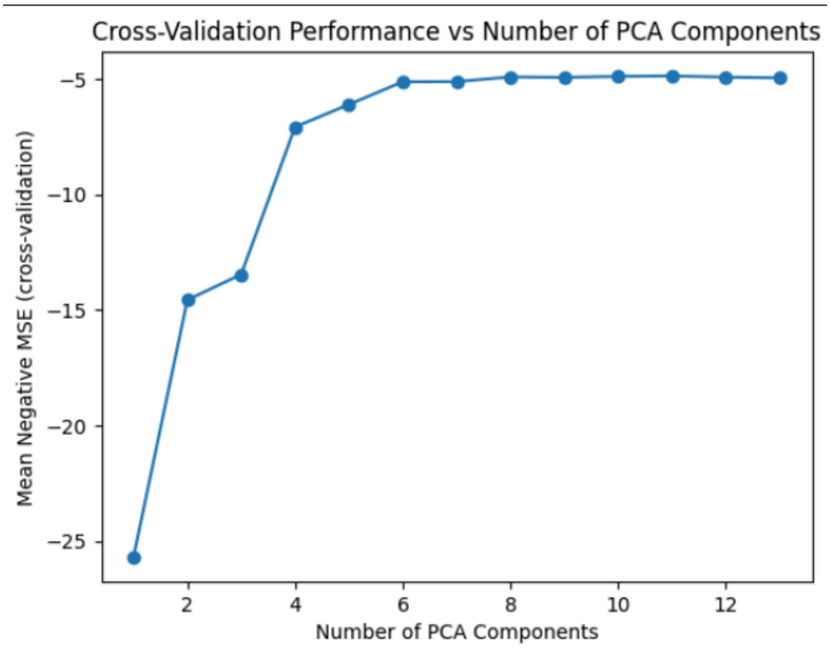
Model Training and Predicting.

For both the models I have used time-series split to split the data into train and test.

Random Forest Model

Applied PCA to separate components since the model has features which are much correlated to each other.

The optimal number of components were found to be 11 .

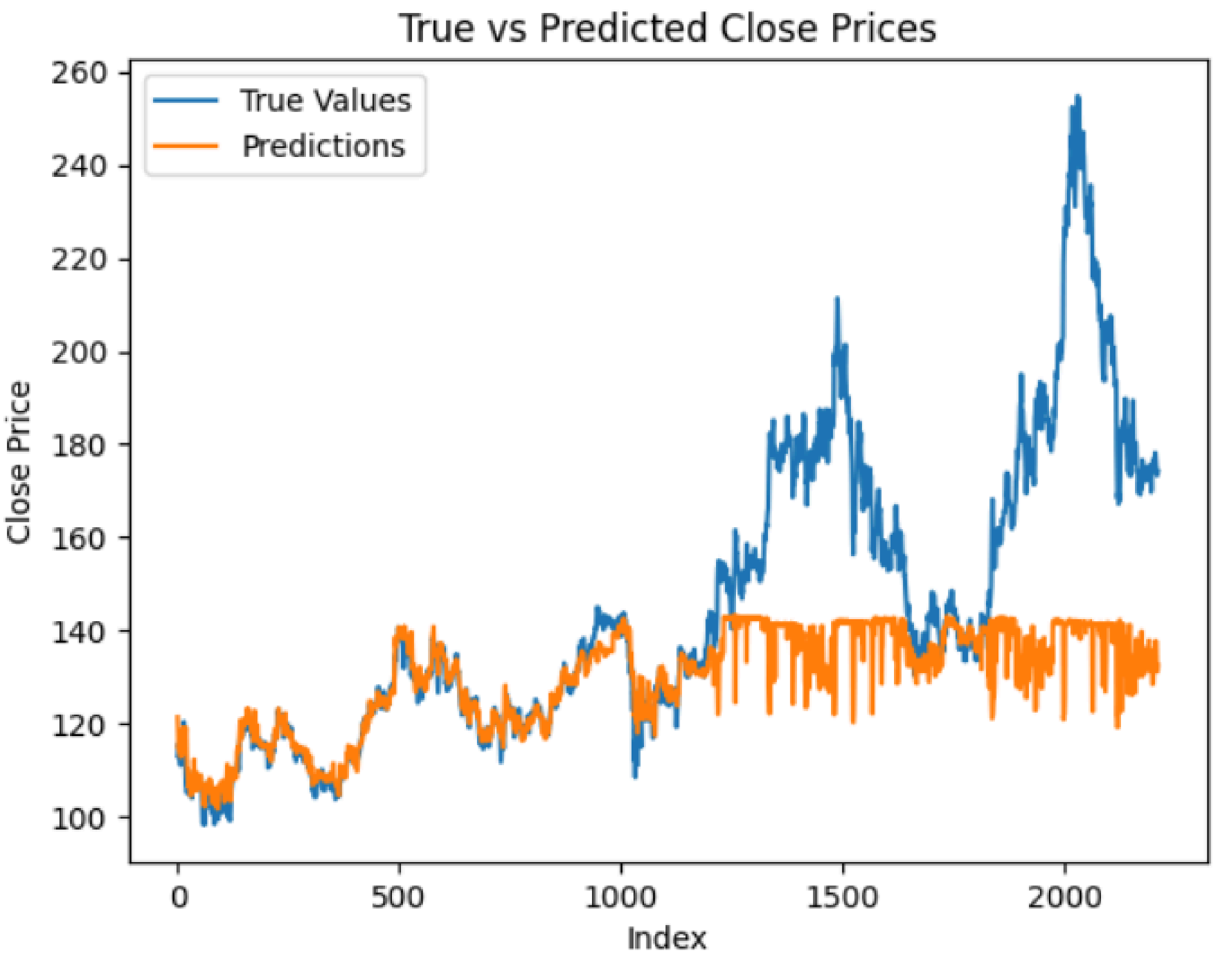


Following are the predictions and accuracy obtained from Random Forest model.

Mean Absolute Error (MAE): 16.606538021066306

Root Mean Squared Error (RMSE): 29.12687265603245

Predicted 'Close' prices for the next 5 days: [132.53246672494095, 132.53246672494095, 132.53246672494095, 132.53246672494095, 132.53246672494095]



XG Boost Model.

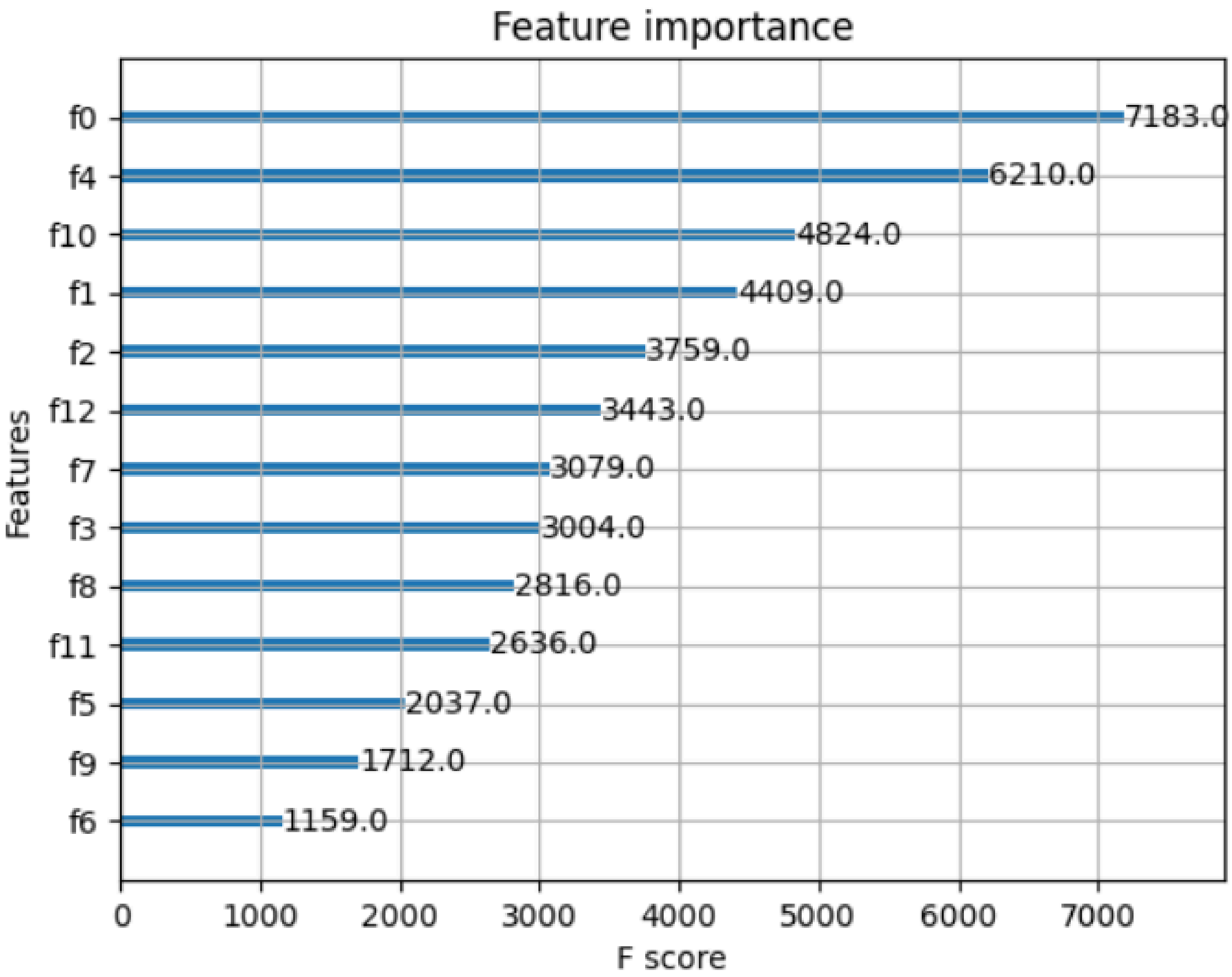
First tried without applying PCA. Following are the statistics obtained.

Mean Absolute Error (MAE): 14.022909338072912

Root Mean Squared Error (RMSE): 26.244445537087085

Predicted 'Close' prices for the next 5 days: [125.38102, 125.38102, 125.38102, 125.38102, 125.38102]


True vs Predicted Close Prices
True Values
Predictions
Close Price
Index
260
240
220
200
180
160
140
120
100
0
500
1000
1500
2000

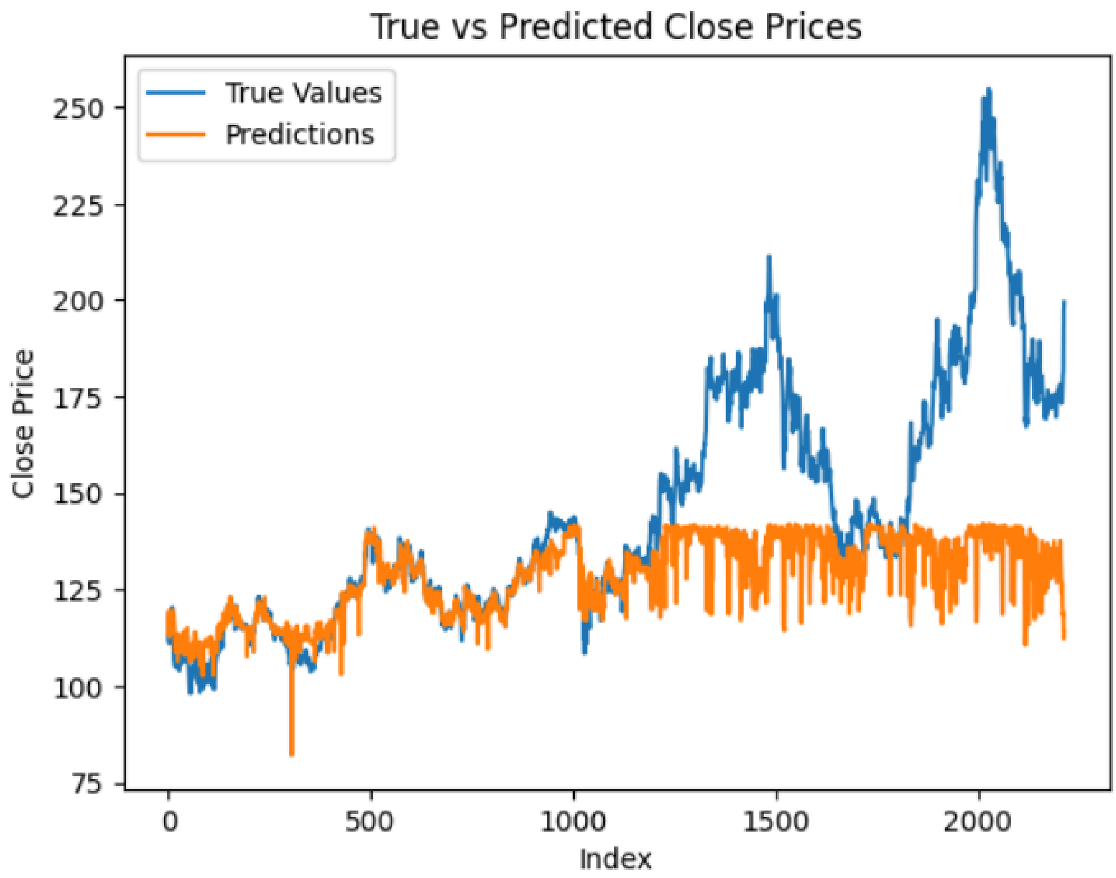


Then made a XGBoost model with PCA applied with 11 components.

Mean Absolute Error (MAE): 18.42134967603185

Root Mean Squared Error (RMSE): 30.7913956760775

Predicted 'Close' prices for the next 5 days: [114.12838981035267, 114.12838981035267, 114.12838981035267, 114.12838981035267, 114.12838981035267]



Based on the RMSE and MAE scores, it is clear that applying PCA here has decreased the accuracy of the XGBoost model.

1. Component Justification

1. Data Collection & Ingestion

Approach:

- Use financial market APIs such as Alpha Vantage, Yahoo Finance, or Bloomberg Terminal to fetch real-time and historical stock price data.
- Ingest additional features like macroeconomic indicators, news sentiment, and social media trends via third-party APIs (e.g., NewsAPI, Twitter API).
- Implement a scheduled batch job (daily updates) or a streaming pipeline (Kafka, AWS Kinesis) for real-time data ingestion.

Technology Choices:

- **Batch Processing:** Python scripts with Pandas for historical data fetching.
- **Streaming Data:** Apache Kafka or AWS Kinesis for real-time market data.
- **Storage:** AWS S3 for raw data storage, PostgreSQL or BigQuery for structured data.

Trade-offs:

- **Batch processing** ensures reliability but lacks real-time updates.
- **Streaming ingestion** provides up-to-date information but requires more resources.

2. Data Processing Pipeline

Approach:

- Perform preprocessing tasks like handling missing values, scaling, and feature engineering.
- Implement feature extraction techniques such as technical indicators (SMA, EMA, RSI) and sentiment analysis.
- Store cleaned and processed data in a cloud-based database (e.g., Snowflake, PostgreSQL, or MongoDB).

Technology Choices:

- **Preprocessing:** Apache Spark for large-scale data processing, Pandas for smaller datasets.
- **Feature Engineering:** TA-Lib for technical indicators, NLP libraries for sentiment analysis.
- **Storage:** Snowflake or PostgreSQL for structured data.

Trade-offs:

- **Spark enables scalability** but is more complex to manage.
- **Pandas is simpler** but struggles with large datasets.

3. Model Operations

Approach:

- Implement a scheduled model retraining pipeline to incorporate new data.
- Evaluate model performance using RMSE and directional accuracy.
- Deploy the trained model via a REST API (Flask, FastAPI) or as a microservice on AWS Lambda.

- Monitor model drift and performance with MLOps tools (e.g., MLflow, EvidentlyAI).

Technology Choices:

- **Model Training:** XGBoost, LSTMs (TensorFlow/PyTorch), or ensemble models.
- **Deployment:** Dockerized REST API using FastAPI.
- **Monitoring:** MLflow for model versioning, Prometheus & Grafana for real-time monitoring.

Trade-offs:

- **Batch retraining ensures stability** but might miss short-term trends.
- **Continuous learning adapts quickly** but can introduce noise.

4. Insight Delivery

Approach:

- Present predictions through a dashboard for analysts and brokers.
- Provide automated trading signals based on model predictions.
- Implement alerting systems for significant stock movements.

Technology Choices:

- **Dashboard:** Streamlit, Plotly Dash, or React.js with D3.js.
- **Alert System:** AWS SNS or Twilio for SMS/email notifications.
- **API for Data Access:** GraphQL or REST API.

Trade-offs:

- **Interactive dashboards provide deep insights** but require user interaction.
- **Automated alerts provide quick insights** but lack in-depth visualization.

5. System Considerations

- **Scalability:** Use AWS Lambda for API scaling, Kubernetes for model deployment.
- **Reliability:** Implement failover strategies, redundant data sources.
- **Latency:** Optimize preprocessing, use GPU acceleration for deep learning models.
- **Cost Considerations:** Minimize cloud costs by choosing efficient data storage (e.g., AWS S3 lifecycle policies).

2. Data Flow Explanation

Batch vs. Streaming Decisions

- **Batch Processing:** Used for model retraining, historical data analysis.
- **Streaming Data:** Used for real-time market updates and instant insights.

Data Transformation Stages

1. **Raw Data Ingestion:** Fetching stock market data and additional sources.
2. **Preprocessing:** Cleaning, normalizing, and transforming data.
3. **Feature Engineering:** Adding technical indicators, sentiment scores.
4. **Model Training & Prediction:** Using trained models for forecasting.
5. **Storage & Serving:** Storing results in databases and providing API access.

System Interaction Points

- APIs interact with the database and prediction service.
- Web dashboard queries prediction results.
- Alerts notify users of important trading signals.

3. Challenge Analysis & Mitigation Approaches

Challenge 1: Data Quality & Noise

- **Problem:** Stock price data contains outliers and anomalies due to market events.
- **Mitigation:** Use anomaly detection algorithms and robust scaling methods.

Challenge 2: Model Drift & Performance Decay

- **Problem:** Models degrade over time as market conditions change.
- **Mitigation:** Implement continuous monitoring and retraining mechanisms.

Challenge 3: High Latency in Real-Time Predictions

- **Problem:** Processing large amounts of data quickly is computationally expensive.
- **Mitigation:** Optimize feature extraction, use GPU acceleration, and deploy models on low-latency servers.

Challenge 4: Scalability Constraints

- **Problem:** Handling large volumes of streaming data efficiently.
- **Mitigation:** Use distributed computing frameworks like Apache Spark and Kafka.

Challenge 5: Security & Compliance

- **Problem:** Stock market predictions require secure handling of sensitive data.
- **Mitigation:** Implement data encryption, access controls, and comply with financial regulations.