IntelliHack 5.0

Machine Learning hackathon

Q4 - Report 01

The Code Rushers

1 Exploratory Data Analysis (EDA)

1.1 Data Overview

The dataset provided contains daily stock market data, including features such as the stock's opening price, high price, low price, closing price, adjacent closing price, and volume traded. The data spans a significant period, offering valuable information to understand stock behavior over time.

```
In [5]: df.describe()
```

Out[5]:

	Unnamed: 0	Adj Close	Close	High	Low	Open	Volume
count	11291.000000	11198.000000	11174.000000	11196.000000	11164.000000	11188.000000	1.114600e+04
mean	5645.000000	63.609130	72.026945	72.503100	71.665079	67.999259	2.144157e+05
std	3259.575279	52.266247	51.259828	51.550735	51.011632	55.834401	3.883662e+05
min	0.000000	2.259452	3.237711	3.237711	3.237711	0.000000	0.000000e+00
25%	2822.500000	19.224636	27.500000	27.789255	27.536156	0.000000	1.350000e+04
50%	5645.000000	50.608900	66.035000	66.724998	65.418751	66.065002	9.032350e+04
75%	8467.500000	104.723621	114.297503	114.892500	113.639999	114.269997	2.915750e+05
max	11290.000000	254.770004	254.770004	255.229996	253.589996	255.000000	1.858270e+07

Figure 1: Dataset Description

1.2 Data Preprocessing

Null Values: Initially, there were missing values in the dataset, which were handled using forward fill for price-related columns and linear interpolation for volume data.

```
In [6]: df.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 11291 entries, 1980-03-17 to 2024-12-27
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  11291 non-null  int64
 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: float64(6), int64(1)
memory usage: 705.7 KB
```

Figure 2: Nullvalues Preview

- **Forward Fill for Price-Related Columns:** This method assumes that the missing price on a given day can be estimated by the previous day's price. Stock prices typically follow a continuous trend, so forward filling helps maintain that trend without introducing significant errors.

- **Linear Interpolation for Volume:** Volume data is usually less volatile and changes gradually. Linear interpolation estimates missing volume values by averaging the surrounding data points, preserving the trend and avoiding abrupt changes. This ensures smooth and consistent data for model training.

Feature Engineering:

- Price Differences: The differences between open-close, low-high, and high-low prices were calculated to capture fluctuations within the daily range.
- Exponential Moving Averages (EMA): 5-day and 10-day EMAs were added to capture the smoothing effect of prices over a short-term period.
- Rolling Features:A 5-day rolling mean for the open-close price difference and a 10-day rolling standard deviation for volatility were computed to add more temporal features.
- Target Variable: The target variable is the stock's closing price, shifted by 5 days, representing the future price to predict.

1.3 Visualizing the Data

- Time Series Plots: The closing price and adjusted closing price were plotted over time, demonstrating the overall trend in stock prices.

Figure 3: Comparison of Closed Price and Adjacent Closed Price

  - The **closing price** is universally used as the standard benchmark in stock market analysis. It reflects the final price at which a stock is traded during regular market hours, offering a consistent point of reference for performance comparison across different days.
  - **Adjacent closing price**, on the other hand, refers to the closing price of the previous day or a similar measure, which may not capture the full context of the current market condition.

- Histograms: In the distribution plot of OHLC data, we can see two peaks which means the data has varied significantly in two regions. And the Volume data is left-skewed.

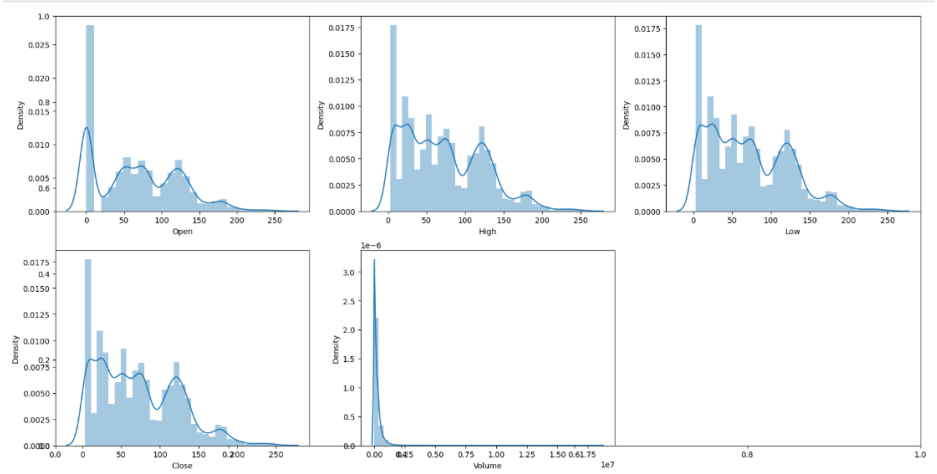

Figure 4: Histograms

- Boxplots: Outliers were identified primarily in volume data.

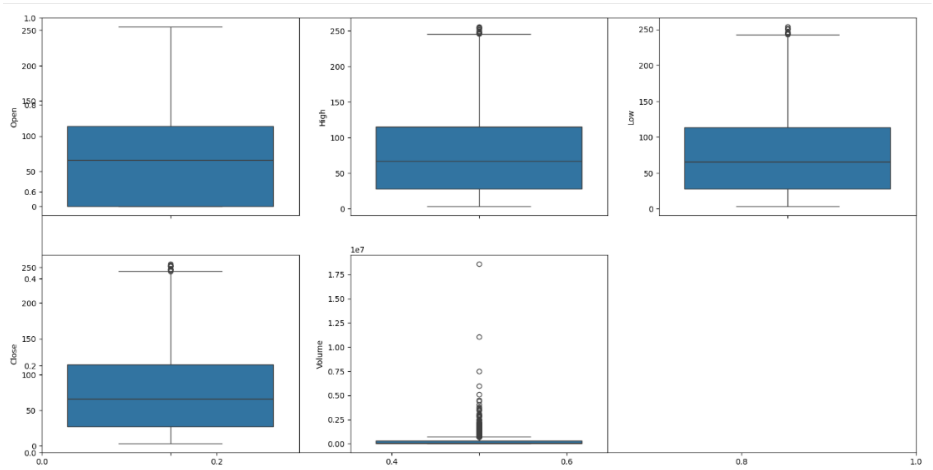

Figure 5: Boxplots

- Correlation Heatmap: A heatmap was used to examine the relationships between the features, revealing high correlation between the added features, where the OHLC features are not highly correlated with each other. Which means the newly added features contribute much to the model accuracy.

2 Feature Selection

The selected features for modeling include:

- Open-Close Difference, Low-High Difference, High-Low Difference, Volume
- Low, High, Open prices

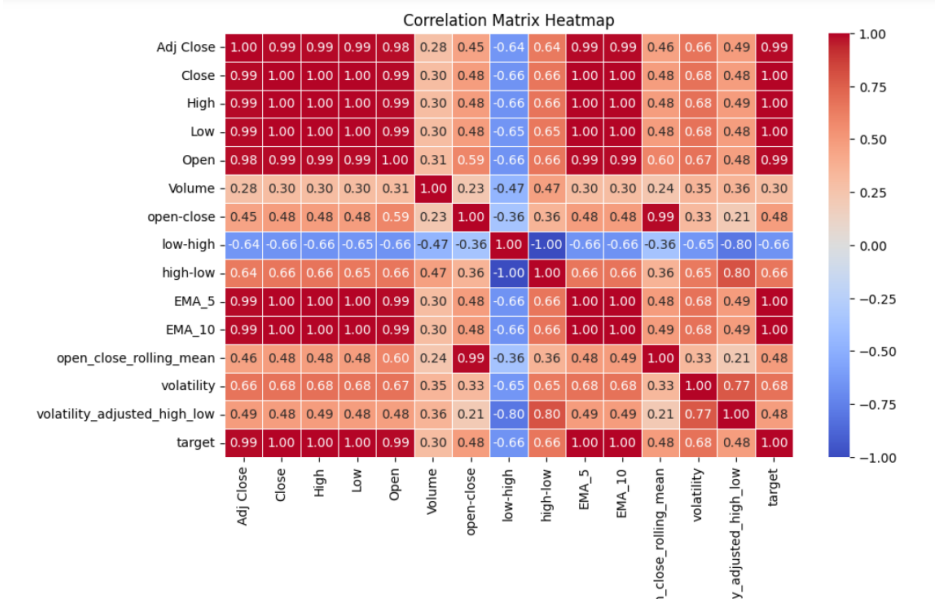

Figure 6: Correlation HeatMap

- 5-day and 10-day EMA, Rolling Mean, and Volatility Features

These features were chosen based on their relevance in predicting future stock prices and their ability to capture patterns in stock movements over time. The above features showed less correlation with each other. And also a few those showed a highly distributed nature in the plots. So we could conclude that the above features contribute to the precise model training.

3 Data Handling

Feature scaling was performed using `StandardScaler()` to standardize the dataset, by removing the mean and scaling it to unit variance. This ensures that all features have the same scale, preventing models from giving higher importance to features with larger numerical ranges. Standardization also helps optimization algorithms converge faster and improves the performance of models that rely on distance-based calculations.
The dataset was then split into training (90%) and validation (10%) sets using `train_test_split()` with `random_state=2022` for reproducibility.

4 Model Development

4.1 Model Selection

Several models were tested to predict the closing stock price:

- **Linear Regression:** A simple model, useful for linear relationships between features.

- **Support Vector Regression (SVR):** A more complex model using kernel tricks to capture non-linear relationships.
- **XGBoost Regressor:** A gradient boosting method that is known for its high predictive accuracy and ability to handle complex datasets.

4.2 Model Evaluation Metrics

- **RMSE (Root Mean Squared Error):** This metric was used to measure the model's predictive accuracy.
- **Directional Accuracy:** The ability of the model to predict the direction (up or down) of stock prices was evaluated. This is critical in trading applications where the exact price isn't as important as correctly predicting price direction.

4.3 Model Comparison

```
LinearRegression() :
Training R2 Score:  0.9965112658472597
Validation R2 Score:  0.9962712815312719

SVR(kernel='poly') :
Training R2 Score:  0.7485105182191014
Validation R2 Score:  0.712476446708141

XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...) :
Training R2 Score:  0.9992580731948617
Validation R2 Score:  0.9968387629019054
```

Figure 7: Results

- **Linear Regression:** Performs well with a high R^2 score, but it assumes a linear relationship between features and target, which may not always hold in complex stock market data.
- **SVR (Polynomial Kernel):** Shows significantly lower performance, likely because polynomial SVR struggles with high-dimensional, non-linear relationships in financial data.
- **XGBoost Regressor:** Best performance with high validation scores.

  - Provides the highest performance with both training and validation scores close to 1.
  - Captures non-linearity and feature interactions effectively.
  - Uses boosting, reducing bias and variance, making it highly suitable for stock price prediction.

- **LSTM Model:** Higher RMSE suggests that LSTM had larger prediction errors compared to XGBoost. Also LSTM is **computationally expensive**, as to **makes the cost high,** LSTM model is not selected.

4.4 Why XGBRegressor is selected?

- XGBRegressor achieved the highest validation R^2 (0.9968), meaning it explains more variance in the data than other models.
- XGBRegressor can model complex relationships in data, unlike linear regression, which assumes a linear relationship.
- It uses gradient boosting, which corrects errors in sequential iterations.
- XGBRegressor is selected because it provides the best balance of accuracy, efficiency, and predictive power, outperforming other models in both training and validation.

4.5 Model Tuning with Grid Search

To improve performance, hyperparameter tuning was performed using Grid Search on the XGBoost model. The following hyperparameters were tuned:

- `n_estimators`, `learning_rate`, `max_depth`, `subsample`, `colsample_bytree`

Best model selection was based on minimizing `neg_mean_squared_error`.

5 Model Performance and Evaluation

Results:

- RMSE (Root Mean Squared Error): The final XGBoost model achieved a satisfactory RMSE, indicating reasonable accuracy in predicting the stock's future closing price.
- Directional Accuracy: The directional accuracy was computed by comparing the predicted price movement direction to the actual movement direction, with a strong match indicating good trading value. The value ¿50% indicates that the model performed well.

Feature Importance: Feature importance analysis was performed using the trained XGBoost model. The most important features influencing predictions were:

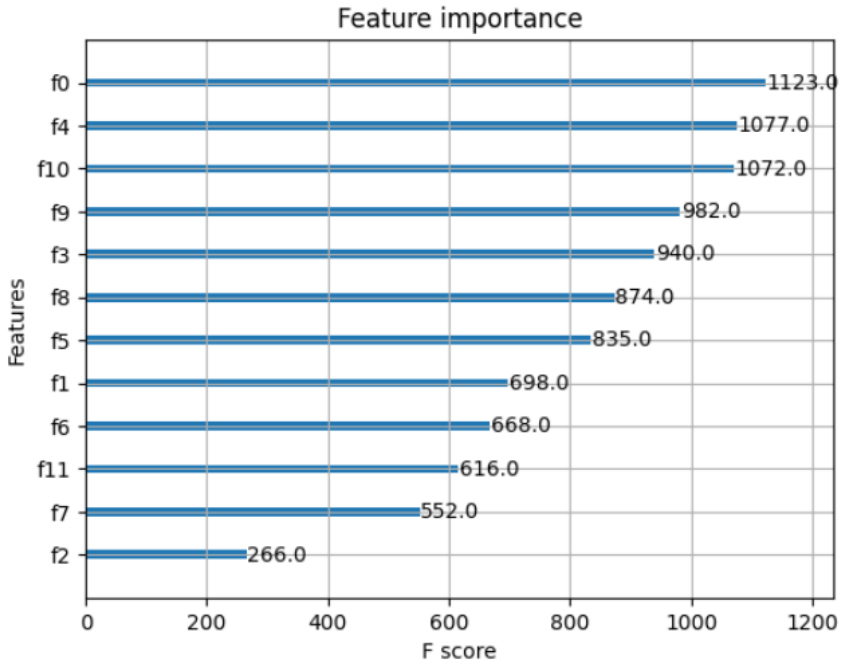

Figure 8: Feature Importance using XGBoost model

- EMA_5, EMA_10 were the most influential features.
- Open-Close difference significantly contributed to predictions.

6 Model Limitations and Future Improvements

6.1 Limitations

- Overfitting: Despite hyperparameter tuning, the model may still be prone to overfitting due to the complexity of stock price movements hyperparameter tuning.
- Exogenous Factors: The model does not incorporate external factors like market news, geopolitical events, or macroeconomic indicators, which can heavily influence stock prices.
- Feature Set: Additional features, such as sentiment analysis from news articles or social media, could further enhance the model.

6.2 Future Improvements

- Incorporating sentiment analysis from news articles.
- Exploring deep learning models like LSTMs.
- Expanding training data with alternative stock markets.

7 Conclusion

The XGBoost model demonstrated reasonable performance in predicting the stock's closing price 5 trading days into the future. With the current dataset and feature set, the model achieved good accuracy and directional forecasting, which is critical in trading scenarios. Future work could focus on incorporating external factors and exploring more advanced models like deep learning approaches to enhance predictive accuracy.

IntelliHack 5.0

Machine Learning hackathon

Q4 - Report 02

The Code Rushers

1 System Architecture Design

Stock markets are one of the most important financial pillars in the country's economy. They play a crucial role in a country's success. There's a huge risk in inventing these, and there is also high security involved. Therefore, the existence of a decision support system for evaluating markets' conditions and providing suggestions to investors can be very beneficial.
Below is a proposed system architecture for a Stock Price Prediction system.

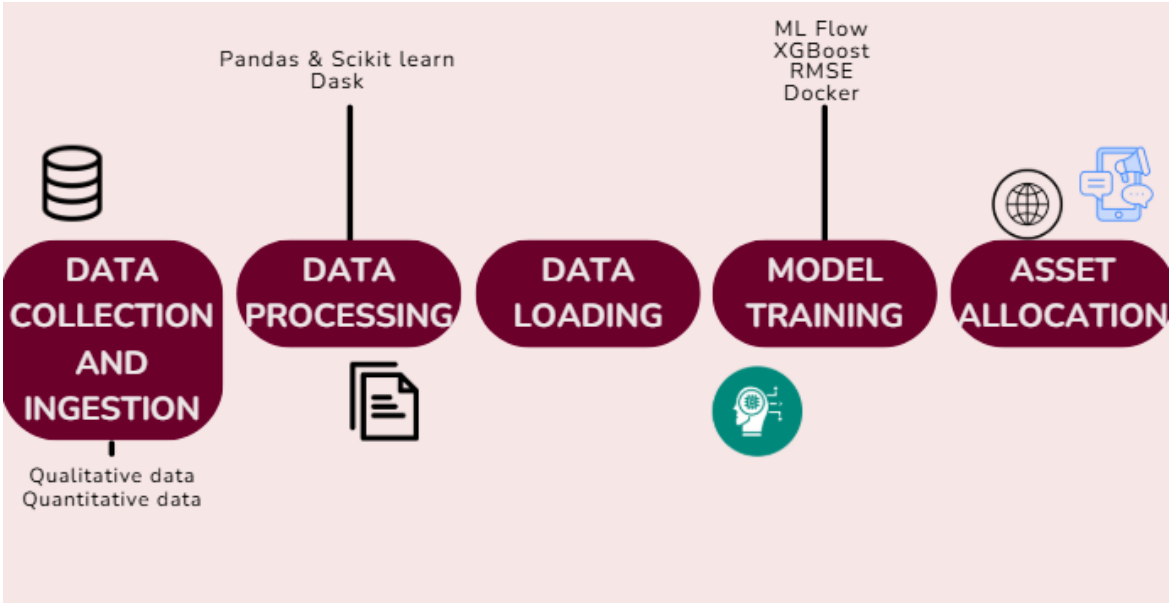

Figure 1: System Architecture

1.1 Data Collection & Ingestion

The system is based on qualitative and quantitative data for deciding between buy, keep, or sell options. The quantitative data includes past stock trades data of the company, and the qualitative data is gathered from expert traders of the stock market.
Example: Alpha Vantage API, Yahoo Finance API, Quandl API
This works as Batch Data ingestion. Data is queried daily and put into a SQL database. Data can be fetched and stored at regular intervals for subsequent processing.
Technologies: **Airflow** for scheduling tasks, **Google Cloud Functions**.

1.2 Data Processing

Cleaning: Handling missing values using Forward fill and interpolation, outliers are removed.
Feature Engineering: Stock price prediction is done using OHLC price data along with the volume of trades.

Scaling: Standardize or normalize numerical features.
Tools and Technologies:

- Pandas and Scikit-learn: for smaller data and feature engineering.
- Dask: For scaling Pandas-like workflows for larger datasets.

Data Storage Architecture

Raw Data Storage: SQL database for storing raw stock data.
Processed Data Storage: Stored data is then converted to JSON files.

1.3 Data Loading

The third step is for loading data from stored JSONs and merging sequential reports published quarterly. This stage speeds up the data processing and acts as a caching system.

1.4 Model Training

Model Training & Evaluation:

- Automated pipeline to retrain the model after every ingestion.
- MLflow is used for experiment tracking and versioning of models.
- XGBoost is trained on the processed data.

Model Deployment: Deploy the model in a Docker container to isolate the environment and ensure portability. Google AI Platform can be used for model deployment, monitoring, and scalability.

Model Monitoring:

- Implement continuous monitoring using Prometheus to track metrics like prediction accuracy, model performance over time, and latency.
- Use Grafana to visualize model performance and detect any potential data drift or performance degradation.
- Data Drift Detection: Use statistical tests to monitor whether the distribution of incoming data changes over time, indicating the need to retrain the model.

Model Evaluation & Retraining:

- Regularly evaluate the model on a holdout validation dataset.
- Track performance metrics like RMSE, Directional Accuracy, and Simulated Trading Performance.
- Automate retraining if performance metrics fall below a predefined threshold, triggered by the monitoring system.

1.5 Asset Allocation (Insight Delivery)

A website and an SMS system can be incorporated to give the traders real-time updates.

API Gateway Expose the model's predictions and insights via a REST API using Flask. The API handles requests from analysts, brokers, or trading systems and returns predictions or trading recommendations (e.g., "buy" or "sell").
Real-Time Dashboard A Streamlit dashboard can be used to present stock price predictions, insights, and live data updates. Key metrics such as predicted stock price, volatility, and trading signals are displayed. Provided historical performance comparisons.
Notification System An alert system to notify analysts or brokers when significant changes are predicted (e.g., price spike, trend reversal). Use Twilio API to send SMS notifications, or integrate email systems for alerts.

1.6 System Considerations

Scalability:

- **Data Scalability:** Data filtering and feature selection improves scalability. Batch-wise ingestion and processing makes it more scalable. Usage of distributed storage systems like Apache Hadoop or cloud storage (AWS S3, Google BigQuery).
- **Computational Scalability:** Optimizing code using parallel processing (NumPy, TensorFlow, PyTorch). Deploy serverless computing (AWS Lambda, Google Cloud Functions) for real-time predictions.
- **Model Scalability:** Usage of XGBRegressor for long-term sequential data.

Reliability:

- **Data Reliability:** Usage of multiple data sources to cross-check information. Data cleaning techniques (handling missing values, outlier detection). Real-time anomaly detection to filter unreliable inputs.
- **Risk Management & Reliability:** Continuously retrain models with updated market data. Integrate stock market movement prediction.

Latency Requirements: Parallel Processing & Asynchronous Execution (e.g., Dask, Ray for Python). Optimizing real-time trading models.

Cost Considerations:

- Usage of free or open-source datasets (Yahoo Finance, Quandl - limited options).
- Usage of serverless computing (AWS Lambda, Google Cloud Functions) for on-demand execution.

- Training models in batches instead of real-time retraining.
- Implementing batch inference for non-real-time scenarios.

2 Component Justification

2.1 Data Collection & Ingestion

Technology/Approach:

- API-based Integration (e.g., Alpha Vantage, Yahoo Finance)
- Historical data from the company.
- Batch-wise Ingestion

Why this solution?

- APIs provide an easy-to-integrate, reliable, and structured way to fetch financial data. These services offer historical, real-time, and sometimes alternative data with well-documented endpoints.
- Historical data provide trends related to the particular company.

Trade-offs:

- API-based solutions often come with usage limitations (e.g., rate limits) and costs (especially for real-time data). You may need to pay for premium plans for continuous data.

2.2 Data Processing Pipeline

Technology/Approach:

- Pandas and Scikit-learn
- Dask for in-memory data processing for smaller datasets.

Why this solution?

- Pandas/Dask is well-suited for smaller datasets and provides intuitive APIs for data manipulation and transformation.

Trade-offs:

- Dask is a great option for parallel processing on smaller machines but might struggle with very large datasets.

2.3 Feature Engineering

Technology/Approach:

- Time-Series Feature: Moving averages, Volatility, Exponential Moving Averages
- Lag features

Why this solution?

- Time-series indicators such as moving averages are widely accepted in financial analysis and provide actionable insights.

Trade-offs:

- Time-series feature engineering can be computationally expensive, especially if you are incorporating multiple technical indicators or features.

2.4 Model Training & Evaluation

Technology/Approach:

- LSTM (Long Short-Term Memory) for sequence modeling.
- Random Forest/Gradient Boosting for non-linear patterns.
- XGBoost for boosting performance and feature importance analysis.

Why this solution?

- LSTMs are a popular choice for time-series forecasting as they can capture long-term dependencies in data.
- XGBoost/Random Forest are powerful ensemble models that can handle tabular data well and are particularly useful for features generated through traditional technical indicators.

Trade-offs:

- LSTM requires more computational resources for training and is more time-consuming than simpler models.
- Ensemble models like XGBoost may perform well in terms of accuracy, but they are prone to overfitting if not tuned properly.

2.5 Model Deployment & Monitoring

Technology/Approach:

- Docker for containerization of models and ensuring reproducibility.
- Model Monitoring tools like MLflow for tracking model performance, drift, and retraining.

Why this solution?

- Docker makes it easier to deploy the model in different environments (e.g., cloud, on-premises) without worrying about dependencies.
- MLflow can help monitor model drift and performance in real-time, ensuring the model stays relevant and continues to perform as expected.

Trade-offs:

- Docker adds complexity to the deployment process, especially in smaller teams or organizations.
- MLflow requires proper configuration and integration with your model pipeline, which can be time-consuming. Over-monitoring can also add unnecessary overhead.

2.6 Insight Delivery to End-Users

Technology/Approach:

- Web-based Dashboards using Dash, Streamlit for visualizations and prediction insights.
- SMS notification system.
- APIs for delivering model predictions to analysts/brokers.

Why this solution?

- Dash/Streamlit allows for the creation of interactive web dashboards with Python, making it easy to visualize stock price trends and predictions in real-time.
- APIs provide a clean, accessible interface for programmatic access to predictions and insights.

Trade-offs:

- Web dashboards may require continuous development and maintenance as new features are added or user requirements change.
- API delivery might require additional security measures, such as rate limiting and authentication to avoid abuse.

3 Data flow

3.1 Data Flow explanation

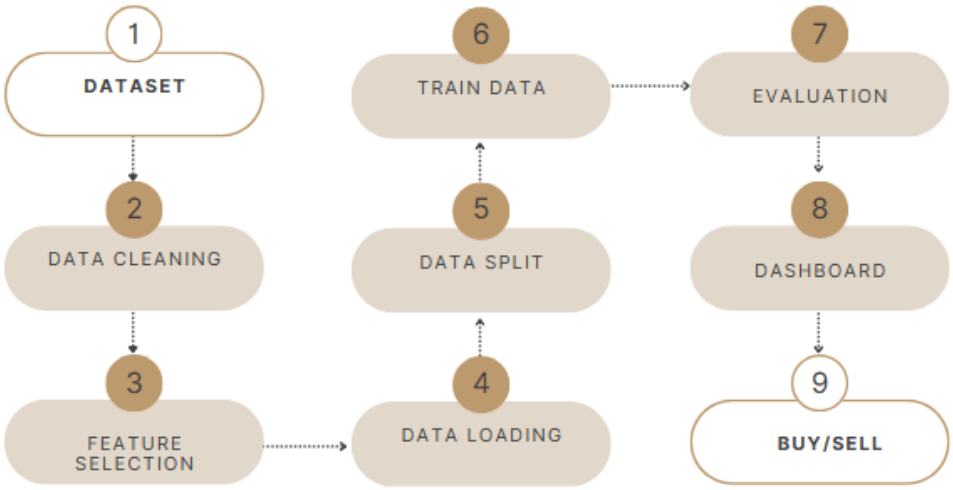

Figure 2: Data flow diagram

1.Data Gathering: The data gathering step includes crawling and saving reports' announcements (online and historical data) in a SQL database. After downloading announcements, the next step is saving the financial reports' data in their original format (HTML files).

2.Data Cleaning: In this step, the main activity is cleaning of the semi-structured reports, including normalizing texts and numbers, mapping reports to a unified format, and etc. For later access to the cleaned data, we save it in JSON format in the SQL database.

3.Data Loading: The third step is for loading data from stored JSONs and merging sequential reports published daily. This stage speeds up the data processing and acts as a caching system.

4.Stock Modeling: The main part of the system is training a model that can predict stock return in various periods. For this purpose, different categories of features are prepared for predicting stock return and a machine learning model is trained.

5.Asset Allocation: The last step is for allocating assets in different securities. The trained model in the previous step is used for predicting the stock return and then determining the percentage of fund for each asset.

The main point of system design is the pipeline architecture, which allows the output of each stage to be stored independently, and each stage can be run separately.

3.2 Stock Market Movement Prediction

The stock market's health is a key indicator of overall economic well-being, and the good performance of stocks attracts investors' attention to the stock market among different financial markets. Asset allocation, in general, is the decision of distributing funds in different assets of various financial markets. Asset allocation in various markets is a key decision for every investor. With asset diversification, the investment risk is minimized in terms of the earned return.
The system's core functionality is based on the proposed stock market return predictor.
Based on the computed weighted probabilities, we choose between two asset allocation strategies:

1. Market growth probability of more than 50%
2. Market growth probability of less than 50%

4 Challenge Analysis

1.Gathering, cleaning, integrating, and exploring published fundamental data. Solution:

- Use data cleaning techniques (e.g., handling missing values, outliers).
- Explore alternative data sources like sentiment analysis or economic indicators.
- Continuously monitor data quality and clean it during data preprocessing.

2.Some companies in the stock market publish their financial reports regularly in each quarter. But there are more challenges in accessing those data. A crawler is needed for continuous download of announcements. Data is published in semi-structured data, populated with texts and numbers. Data format is changed in different quarters, thus requiring a data mapper for unifying formats. Stocks in different industries have different report formats.

3.Latency & Real-Time Prediction - High latency can result in missed opportunities or incorrect predictions, especially in the highly volatile and fast-paced stock market. **Solution:**

- Optimize the data ingestion pipeline for faster real-time data processing.
- Use model quantization or simpler models for quicker inference.

4.Market Volatility and External Factors - Stock prices are affected by historical data and external factors such as geopolitical events, natural disasters, or economic crises. Predictive models based only on historical prices may fail to adapt to these sudden changes.
Solution:

- Integrate external data sources (e.g., news sentiment, macroeconomic indicators) into your model to improve prediction robustness.
- Regularly retrain the model with fresh data to capture recent market changes.

5.Computational Resource Requirements - Training sophisticated models like LSTMs or transformers requires significant computational power (GPUs/TPUs), which can lead to high infrastructure costs.
Solution: Usage of simple models like XGBRegressor, which need less computational power.

References

1. https://arxiv.org/html/2412.05297v1
2. https://www.researchgate.net/publication/356093237_Stock_Market_Prediction_Using_Machine_Learning_Techniques