Big Data Final Report

1 Overview

Our project investigates daily return prediction, specifically targeting the one-day-ahead return (`Target_Return_1d`), using time series data from the `sliding_time_splits_forbes2000` and `sliding_time_splits_nasdaq` datasets, available via Kaggle[1]. The target is computed as the percentage change between the current day's closing price and the next day's closing price:

`Target_Return_1d` = (`Target_FutureClose_1d` - `Close`) / `Close`

These datasets comprise thousands of CSV and JSON files containing historical market data, including daily prices, volume, and company metadata. We leveraged PySpark for all major stages of data loading, transformation, analysis, and modeling pipeline construction.

2 Exploratory Data Analysis (EDA)

2.1 Data Sources and Structure

We first examined both CSV and JSON file formats:

- **CSV files:** Contain daily trading metrics such as `Date`, `Open`, `High`, `Low`, `Close`, `Adjusted Close`, and `Volume`.
- **JSON files:** Provide additional metadata including exchange information, currency, instrument type, and richer time series through the `chart.result` object.

File Coverage: We identified over 1,000 CSV and JSON files, spanning a wide set of global companies.

2.2 Initial Statistics and Sampling

Using PySpark, we read a sample of 20 CSV files using a unified schema and combined them into a master DataFrame:

[1] `https://www.kaggle.com/datasets/paultimothymooney/stock-market-data`

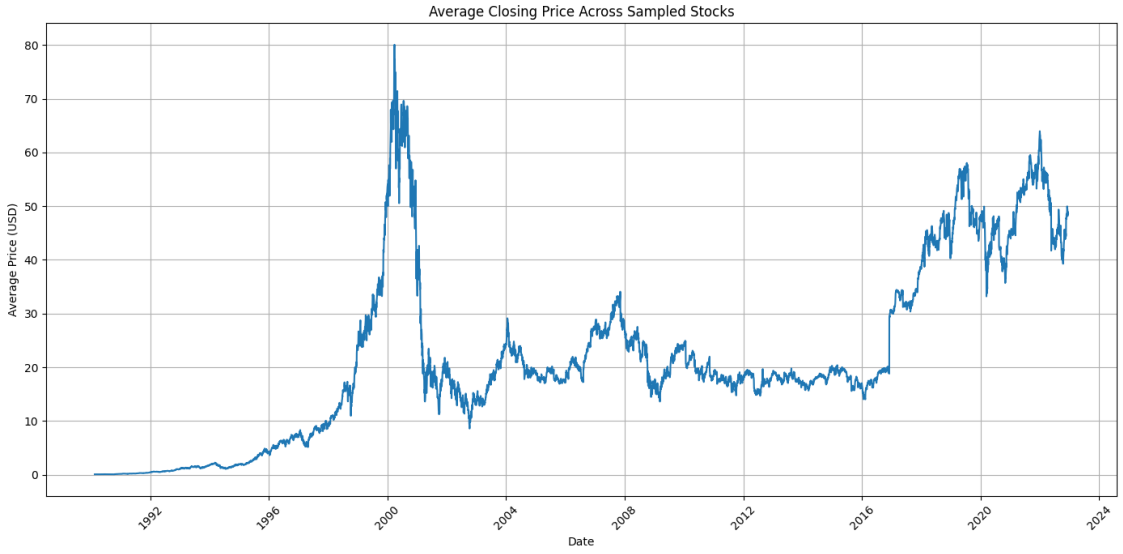
```
StructType([
    StructField("Date", StringType(), True),
    StructField("Low", DoubleType(), True),
    StructField("Open", DoubleType(), True),
    StructField("Volume", IntegerType(), True),
    StructField("High", DoubleType(), True),
    StructField("Close", DoubleType(), True),
    StructField("Adjusted_Close", DoubleType(), True)
])
```

The merged result from 20 stocks yielded nearly 80,000 rows, highlighting the data's scale.

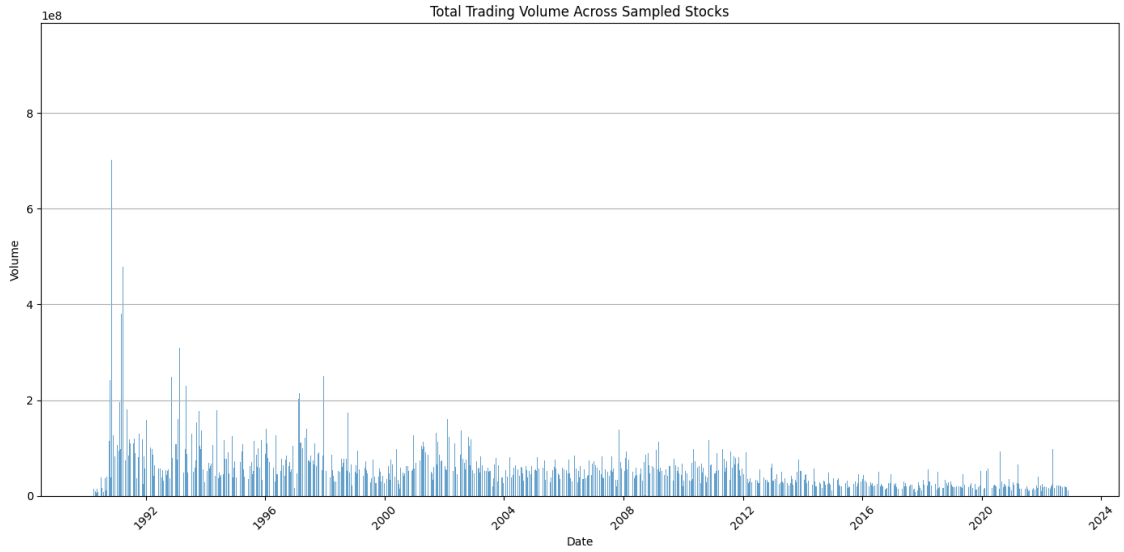
2.3 Market-Wide Trends

To visualize general market behavior:

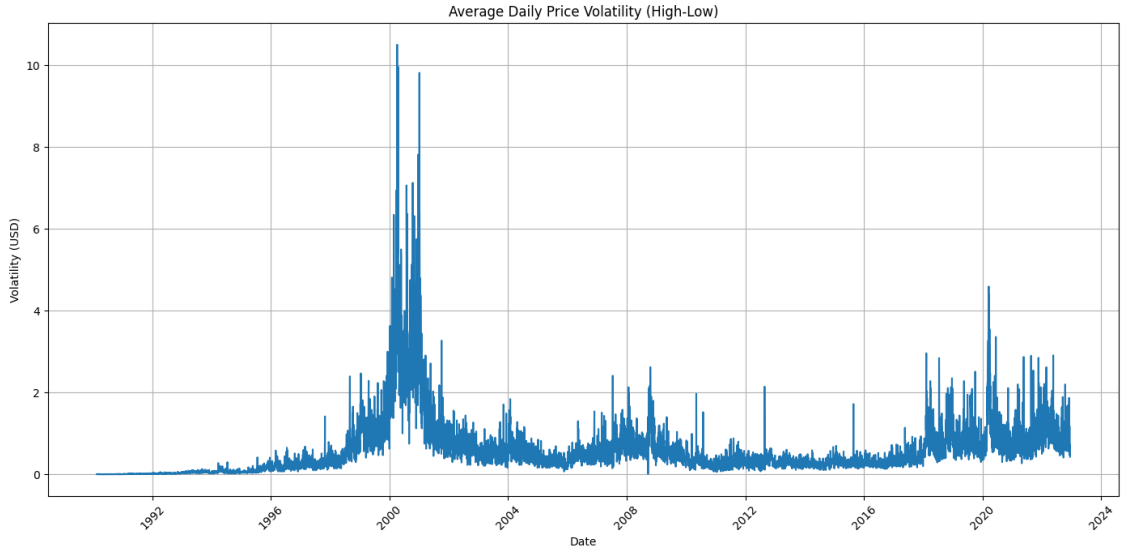
- We computed average closing prices by date and plotted their time trends.
- Total trading volume and price volatility (`High - Low`) were also computed and visualized.



(a) Average closing price across sampled stocks



(a) Total trading volume over time



(b) Average daily price volatility (High - Low)

Figure 2: Market-wide indicators: price, volume, and volatility

2.4 Outlier Analysis

We used the IQR method to flag extreme outliers in `Volume` and `DailyReturn`. Over 9.45% of data points were flagged for volume, and 10.55% for returns. Below is a sample of detected outliers:

Sample Volume Outliers:

```
+------+----------+---------+-----------+-----------+
|Symbol|      Date|   Volume| LowerBound| UpperBound|
+------+----------+---------+-----------+-----------+
```

```
|  CSCO|2014-02-13|153739400|-2.880675E7|1.2281445E8|
|  CSCO|2013-11-14|243255400|-2.880675E7|1.2281445E8|
|  CSCO|2013-08-15|130110100|-2.880675E7|1.2281445E8|
|  CSCO|2013-05-16|201626500|-2.880675E7|1.2281445E8|
+------+----------+---------+-----------+-----------+
```

Sample Return Outliers:

```
+------+----------+--------------------+----------+----------+
|Symbol|      Date|         DailyReturn|LowerBound|UpperBound|
+------+----------+--------------------+----------+----------+
| AIPUY|2022-12-12|-0.02686131709342...|       0.0|       0.0|
| AIPUY|2022-12-08|0.036156489496120825|       0.0|       0.0|
| AIPUY|2022-12-06|-0.03857140314011347|       0.0|       0.0|
+------+----------+--------------------+----------+----------+
```

2.5 Correlation and Distribution Insights

We converted a sample to Pandas for visual analysis. Correlation matrix revealed strong multicollinearity among `Open`, `High`, `Low`, and `Close`, with weak correlation between `Volume` and returns.

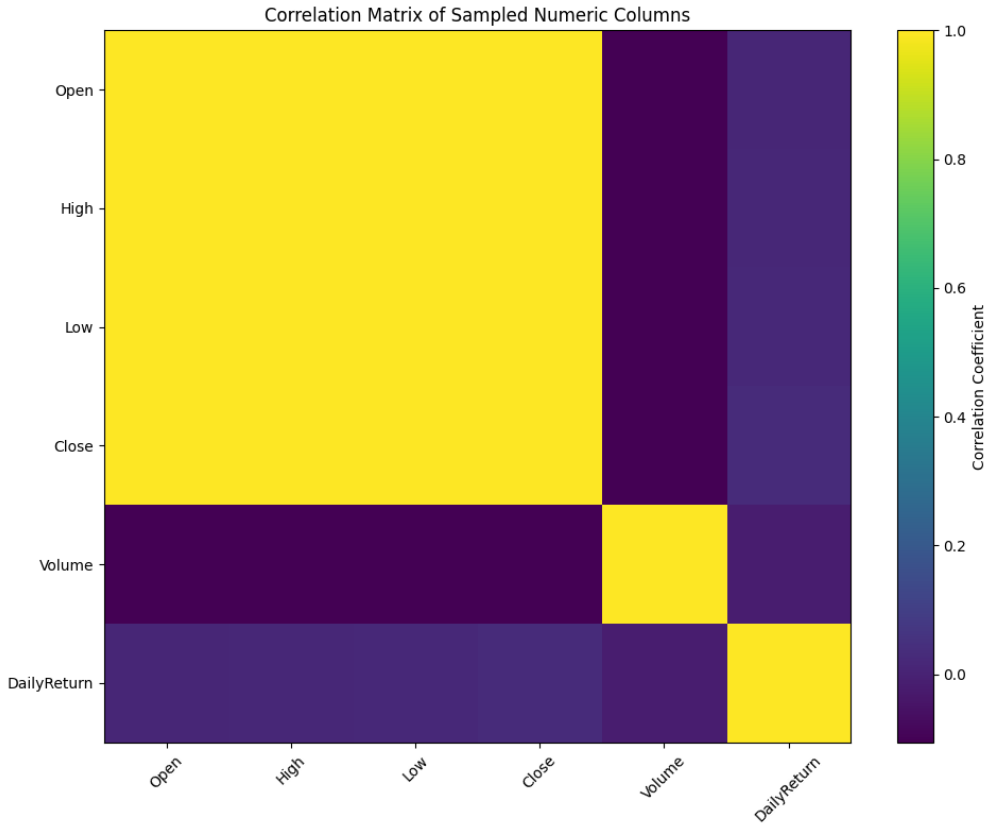


Figure 3: Correlation matrix among numeric features

3 Feature Engineering

3.1 Return and Volatility Computation

Daily returns were calculated using lag windows in PySpark:

```
combined_df = combined_df.withColumn("PrevClose", lag("Close", 1).over(window_spec))
combined_df = combined_df.withColumn("DailyReturn",
    (col("Close") - col("PrevClose")) / col("PrevClose"))
```

We then calculated average return and volatility per stock, filtering for those with more than 500 trading days.

Top 3 Stocks by Average Return:

```
+------+--------------------+--------------------+-----------+
|Symbol|           AvgReturn|          Volatility|TradingDays|
+------+--------------------+--------------------+-----------+
|  AUTR|0.004226934351660896| 0.12415432977666538|       2994|
| AIPUY|0.001395491989984...|0.036751769256370824|       2555|
|  ISRG|0.001356723705433...|0.031622103192064835|       5659|
```

Top 3 Stocks by Volatility:

```
+------+--------------------+--------------------+-----------+
|Symbol|           AvgReturn|          Volatility|TradingDays|
+------+--------------------+--------------------+-----------+
|  AUTR|0.004226934351660896| 0.12415432977666538|       2994|
|   UAL|9.208234591147626E-4| 0.04181226274169283|       4243|
| AIPUY|0.001395491989984...|0.036751769256370824|       2555|
```

3.2 Derived Feature Summary

- **AvgReturn**: mean of daily returns
- **Volatility**: std dev of daily returns
- **TradingDays**: count of rows per stock
- **Outlier Flags**: binary columns for volume/return anomalies

4 Data Processing Pipeline

4.1 Unified Pipeline Design

We designed a modular pipeline in PySpark:

1. Read and parse CSV/JSON into structured DataFrames
2. Convert types and format dates

3. Compute lag features, rolling statistics, and outliers
4. Filter low-quality stocks (e.g., ¡500 days)
5. Generate training splits using sliding windows
6. Normalize features (optional)

To ensure scalability and reproducibility, we implemented **Parquet checkpointing** at two key stages: (1) after concatenation, cleaning, and scaling of the raw data, and (2) after generating fold-level datasets using a 5-year train, 1-year validation, and 1-year test split. These checkpoints significantly reduced reprocessing time and enabled modular experimentation across different modeling pipelines.

4.1.1 Sliding Window Strategy

Our modeling used time-based sliding windows. Each stock was treated independently, and window-based subsets were extracted chronologically to mimic real-world backtesting conditions.

5 Dimensionality Reduction and Final Modeling Strategy

Given the high dimensionality of features extracted from the original dataset — including raw prices, engineered volatility indicators, moving averages, and lag features — we adopted Principal Component Analysis (PCA) to reduce the dimensional space and mitigate overfitting.

Using PySpark's MLlib, we first performed feature scaling (standardization) on numerical columns and vectorized categorical columns. PCA was then applied to extract the top $k = 20$ components, which explained a substantial portion of the variance while reducing noise and redundancy.

To maintain model reliability, we fit PCA on fold 6 and reused it across all other folds and the test set. The transformation was integrated into a streamlined pipeline for each training split.

5.1 Sliding Window Folds and Time-Weighted Ensemble

To reflect realistic deployment conditions, we constructed our model using 7 folds based on chronological stock behavior. For each fold, we trained two models — a linear regression and a factorization machine regressor — both optimized through early experimentation on fold 0.

Instead of simply averaging predictions, we implemented a recency-weighted ensemble strategy. The logic behind this was that newer data often carries more relevance in financial modeling. Thus, fold 6 (most recent) received a weight of 7, fold 5 received 6, and so on, down to fold 0 with weight 1. The ensemble prediction for each test instance was calculated as the weighted average of all model outputs across folds.

5.2 Evaluation Results

Our models were evaluated using RMSE, MAE, MSE, and R^2 on two separate datasets: `sliding_time_splits_forbes2000` and `sliding_time_splits_nasdaq`. The final performance metrics are reported below.

Table 1: Final Ensemble Model Results (Forbes2000 vs Nasdaq)

Dataset	RMSE	MAE	MSE	R^2
Forbes2000	0.078433	0.027189	0.008305	0.017433
Nasdaq	0.065031	0.022865	0.004225	0.015129

Although R^2 values remain low — which is expected in noisy stock return prediction tasks — both datasets exhibited stable RMSE and MAE across folds, suggesting the derived features and PCA transformation provided meaningful predictive signals.

5.3 PySpark Infrastructure Highlights

Our entire pipeline was built on PySpark to handle scale and modularity. Key MLlib components used include:

- **Data Preprocessing:** `VectorAssembler`, `StandardScaler`, `StringIndexer`, and `OneHotEncoder` for feature vector construction and normalization.
- **Dimensionality Reduction:** `PCA` to compress numerical features.
- **Modeling:** `FMRegressor` was chosen for its ability to model interactions in sparse data efficiently.
- **Hyperparameter Tuning:** `ParamGridBuilder` and `CrossValidator` were used to select the best model parameters.
- **Evaluation:** `RegressionEvaluator` helped assess performance using RMSE and R^2.

This infrastructure allowed for robust, scalable experimentation and reproducibility throughout the modeling workflow.

6 Conclusion and Reflections

Our PySpark-powered pipeline enabled us to process a vast stock dataset with reliability and scalability. We successfully identified high-return and high-volatility stocks, and created meaningful engineered features like `DailyReturn`, `Volatility`, and `OutlierFlags`. EDA highlighted patterns of multicollinearity and anomalies, guiding robust preprocessing. The modular design now sets a strong foundation for model development in later milestones.

Successes. Our most successful achievement was the careful engineering of derived features from raw market data. We crafted volatility metrics, return features, lag-based variables, and time-aware splits that retained the underlying signal. Additionally, our use

of PCA and a weighted ensemble helped smooth over volatility and made the model more robust to overfitting.

Challenges. A persistent challenge was balancing generalization with performance. Despite sophisticated preprocessing, financial data is inherently noisy, and overfitting remained a threat. We tackled this through dimension reduction, validation-based tuning, and fold-averaged inference. Moreover, handling and aligning thousands of files (both CSV and JSON) across companies pushed the limits of our Spark pipeline, especially during join and outlier detection steps.

Final Thoughts. This project demonstrates that combining scalable infrastructure (PySpark), smart statistical engineering (PCA and time-weighted folds), and rigorous evaluation strategies can yield a robust and generalizable stock prediction model — even in a notoriously challenging and volatile domain.