MARIN-BERTIN Guillaume
BURTON ELMO Jaishan

Datastream Project -- RealTime streaming application with Kafka -- Collect trading data using Yahoo API

Introduction

The aim of this project is to create a real time streaming application with Kafka, collecting financial data from Yahoo Finance API. The main goal was to design a pipeline that could handle both historical data and real-time data streams, to predict stock prices of large companies like AXA, HSBC, Toyota, Alibaba, and Google. We developed several models, including batch model with Scikit-learn and incremental adaptive models with River, while visualizing the results in an interactive Streamlit dashboard.

To do this, here are the keys steps of the project that we followed :

1. Retrieve financial data of ours companies from the Yahoo Finance API
2. Creating an automated data pipeline: storing data in CSV files for batch models and streaming data to Kafka for incremental and online models.
3. Implementing three types of prediction models: full-batch regression with sklearn, incremental batch regression with River, and online regression with River.
4. Designing a Streamlit dashboard to visualize predictions and monitor model performance in real time.

The data used for the project spans from September 1, 2024, to today and are as follows.

- Open : The stock price at which the market opened for the day
- High : The highest price reached by the stock during the day.
- Low : The lowest price reached by the stock during the day.
- Close : The last traded price during the day.
- Volume : The total number of transactions during the day.
- Market_cap : Total value of the entire company
- Pe_ratio : Financial analysis indicator (stock price/earnings per share)
- Dividend_yield : Dividend yield (annual dividend per share / current stock price).

The target variable will be **close**. The times for retrieving the data vary depending on the market time of the company's location.

1 – Python files explications

1.1 – data_collection.py

This Python script is responsible for collecting financial price data from the Yahoo Finance API, storing it locally in CSV format for batch processing, and streaming the data to Kafka for real-time processing.

Data Retrieval

The script fetches financial data (open,close …) for specified tickers (AXA, Google, Toyota, HSBC and Alibaba) between two dates, using hourly intervals.

Data Storage

The retrieved data is structured into a Pandas DataFrame and saved as CSV files in the data directory. Each file corresponds to a company, enabling batch processing for batch models.

Kafka Integration

The script sends the collected data as JSON messages to a Kafka topic (full_data). Each message includes financial data.

Logging and Exception Handling

Informational messages are printed during data collection to log progress. In cases where no data is found for a ticker, a warning is displayed.

Real-Time Data Streaming

The script also checks that the data is sent in the "correct order », simulating real-time data updates.

1.2 – Predictions files

1.2.1 batch_regression.py

This Python script implements a batch-based linear regression model using the Scikit-learn library. It trains the model on historical data stored in CSV files and sends the results to Kafka for further processing or visualization.

Data Retrieval

The script loads historical financial data from CSV files located in the data directory. It splits the data into training and testing sets.

Model Training

Using Scikit-learn's LinearRegression, the model is trained on the training set. Predictions are then generated for the testing set.

Performance Evaluation

The script calculates key metrics such as MAE, MSE and RMSE to assess the model's performance. These metrics are calculated for each company.

Kafka Integration

The predicted results, along with the true values and metrics, are sent as JSON messages to a Kafka topic named ‘result’. This integration allows real-time visualization on a dashboard.

1.2.2 batch_incremental_regression.py

This Python script implements a hybrid batch-incremental regression model using the River library. It combines batch training on historical data with incremental updates based on real-time data streamed from Kafka.

Batch Training

The script trains the model using historical data (from July to mid-November) stored in CSV files. It utilizes River's LinearRegression wrapped in a pipeline with a StandardScaler for feature normalization.

Real-Time Predictions

After the batch training phase, the script listens to the ‘full_data’ Kafka topic for real-time financial data. For each new data point, the model makes predictions and updates itself incrementally to adapt to changes in the data.

Performance Evaluation

Key metrics, such as MAE, MSE, and RMSE, are continuously updated as new predictions are made. This ensures a dynamic assessment of the model's performance. These metrics are calculated for each company

Kafka Integration

Predicted values, true values, and updated metrics are sent back to Kafka as JSON messages to the ‘result’ topic. This data feeds into a dashboard for real-time visualization.

1.2.3 – adaptive_regression.py

The Python script implements an online prediction model using River. We retrieve data for each company over time to make online predictions and adapt the model based on the results (validate_and_train_model). Metrics are calculated for each company and then sent to the topic ‘result’ (get_metrics_for_company). To ensure more consistent results, the metrics are calculated after a few training data points have been processed.

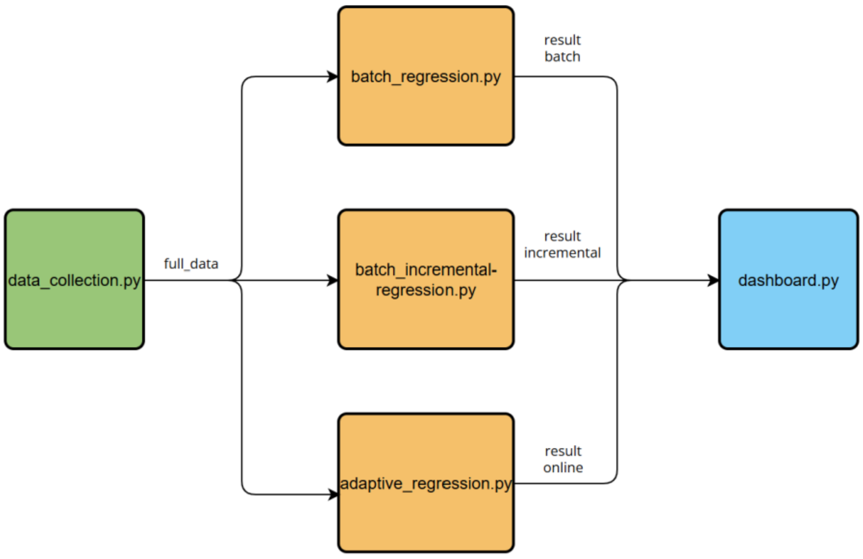
1.3 – dashboard.py

The script is designed to retrieve the metrics of the three models in order to build a graph of the predictions for each company. We iterate through a loop to consume Kafka messages that contain the company name, the prediction, the expected value, and the metrics. Each company has a DataFrame that is updated with the new data.

The metrics are displayed in a table, and graphs are generated using Plotly to show the predictions and the actual values.

2 – Architecture

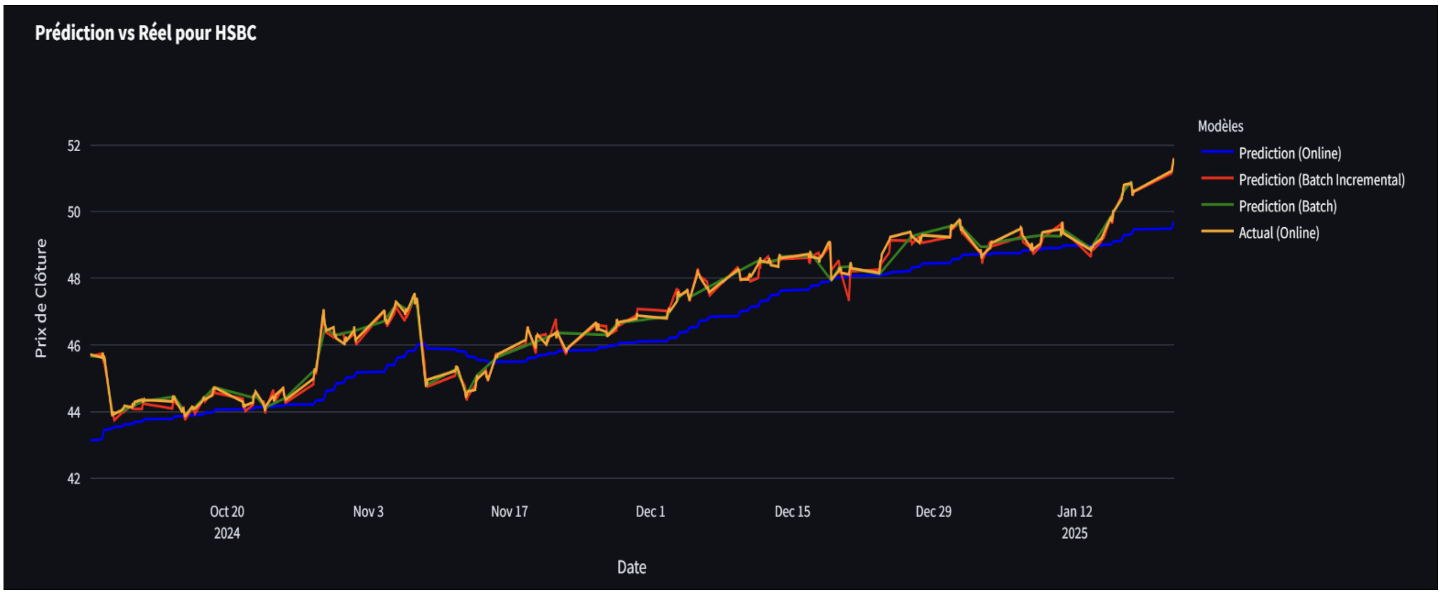
First, the data is retrieved from the Yahoo Finance API and sent to the ‘full_data’ topic (data-collection.py). Then, this data is used to make predictions in parallel within the files batch_regression.py, batch_incremental_regression.py, and adaptive_regression.py. The results are stored in different topics to distinguish between them and make it easier to retrieve and create the graphs in dashboard.py.



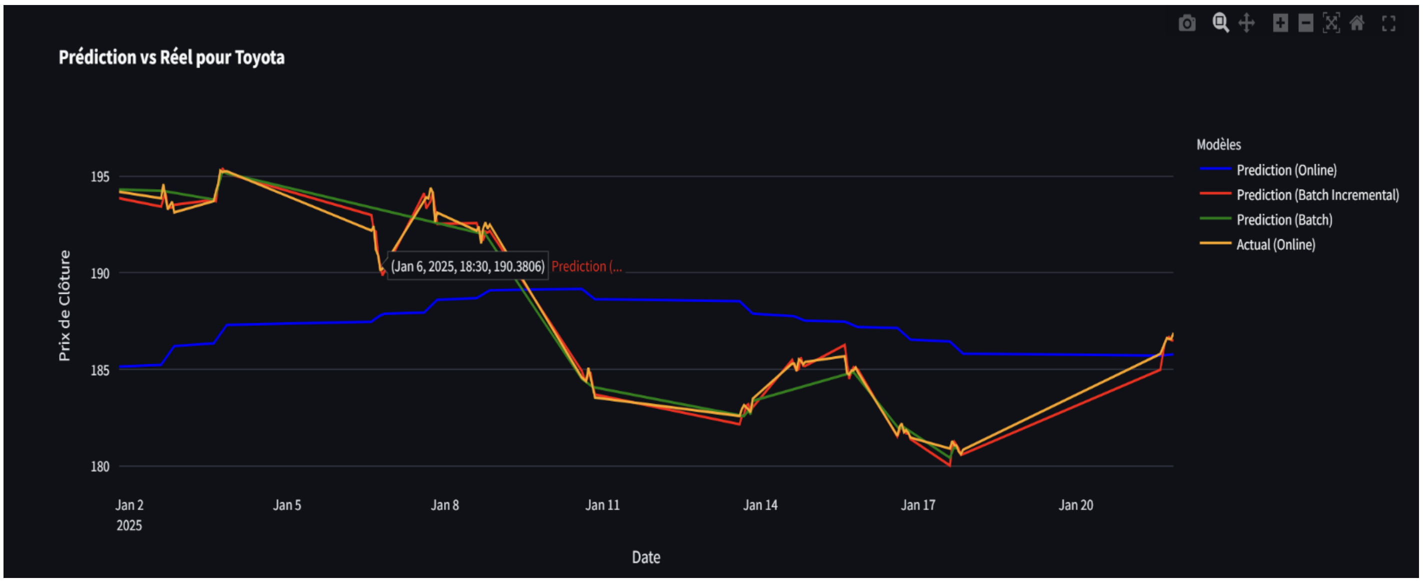
3 - Results

This section outlines the performance of the implemented prediction models: the online regression and batch incremental model with River, and the full-batch regression model developed with Scikit-learn.

For the results, there are 5 graphs (one for each company) with 4 curves (the curve for actual values and predictions from batch model, incremental model, and online model). Here, we have an example for HSBC where the curve for actual values is in orange. We can see that the batch and incremental models are close to reality, while the online model follows the trend without being too close.



The results are generally similar across the 5 companies. In the graph below, we have the predictions for January for Toyota. As mentioned earlier, the online model manages to follow the trend with a lag and lacks precision. We can see that the batch and incremental models have similar performances. However, there is a notable detail: the batch model struggles more to identify a drift, as observed on January 6 at 6:30 PM.



To complement the graphical representation, we provide a detailed comparison of the models' performance through a metrics table. This table summarizes key evaluation metrics, including Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE), for each prediction model across all companies. The table enables a side-by-side comparison of the three models :

Company	**Online Model MSE**	**Online Model MAE**	**Online Model RMSE**	**Batch Incremental Model MSE**	**Batch Incremental Model MAE**	**Batch Incremental Model RMSE**	**Batch Model MSE**	**Batch Model MAE**	**Batch Model RMSE**
AXA	0.333	0.4622	0.5771	0.8239	0.4097	0.6789	0.3016	0.3189	0.5492
HSBC	0.8433	0.8043	0.9183	0.8258	0.331	0.6806	0.2305	0.3195	0.4801
Toyota	27.5298	3.3289	5.2469	5.4534	1.8379	2.0447	6.3021	0.3197	2.5104
Alibaba	12.0614	2.7797	3.4729	3.4067	1.364	1.7782	3.6192	0.3196	1.9024
Google	46.2691	5.1443	6.8021	10.6021	2.5814	3.4211	12.7031	0.3197	3.5641

Based on the observations from both the graphs and the metrics table, we can determine the best-performing models for each company as follows:

AXA and **HSBC**: The **Batch** Model outperformed the other approaches, delivering the most accurate predictions. This result is expected given the stability of their financial data, where the absence of concept drift made the Batch Model's reliance on historical data an advantage.

Google, **Alibaba**, and **Toyota**: The **Batch Incremental** Model proved to be the most effective, leveraging its hybrid nature of batch pre-training and online updates to adapt to the slight concept drift in these datasets. It was followed closely by the Batch Model, while the Online Model (River) consistently showed lower accuracy for these companies.

Worst Performing **Model**: Across all companies, the **Online** Model consistently showed the lowest performance in terms of accuracy. While its real-time updating capability is beneficial in scenarios with frequent data changes, its lighter structure and lack of batch pre-training limited its accuracy in this context.

4 – Conclusion

We are obtaining rather good results with the models. The online model is the fastest but the least accurate. The incremental model is the most accurate over the long term, while the batch model is also accurate but struggles more with new data and drift concepts. This explains why, for AXA and HSBC, where stock prices have been more "stable" in recent months, the batch model performs better.

An improvement in performance could involve grouping companies by time zone and including more companies within the same time zone. This could help the models better capture the specific temporal patterns of these markets, thereby improving overall performance.

Additionally, data collection is currently set to hourly intervals to simplify processing (data retrieval time) and visualization. However, to analyze the market more effectively and achieve better performance with the online models, data should be collected much more frequently, such as every minute.