EXECUTIVE SUMMARY: BUILDING A FINANCIAL WEB APPLICATION

This document serves as a guide on how the application was developed, covering every stage of the process and challenges that came up during the execution and how these were tackled and solved by the team

1. Extract, Transform and Load

This module consisted of solving the first part of the project, which was receiving the raw data and converting it into a usable format that could be passed to the rest of the application. The team decided to use Polars for the initial data handling given the size, and after the necessary transformations have it as a Pandas dataframe to ensure compatibility with the remainder of the project. Here, some basic feature engineering was applied (parsing dates as datetipe, removing null values when irrelevant to the analysis) and the information was merged per company and day reported, to ensure it included all necessary information. One python script was created that covered two cases: Bulk data processing, and streamed data processing. This helped keep functions clean and organized while targeting the two kinds of data the project handled.

2. Creation of the Financial Data Class

A Python class was created to handle all the processes related to financial data (everything outside preprocessing basically), which was initiated with the ticker of a company, and the csv files containing the information on companies and their daily prices. This class called the preprocessing pipeline defined to get the usable dataframe and handled the API calls to get new stock information, as well as defining and running the machine learning and predicting capabilities designed by the team which will be covered in the next section.

3. Machine Learning Approach to Predict Values

After defining the class and importing the cleaned and merged dataset, the team decided to use a an XGBoost regression to predict the price of a stock given the previous day's financial information. The advantage on the user's side by using this model is that we can show actual values, offering more interpretability; and from the developing team's perspective this model made things easier as it is more robust than regression trees or linear regression while needing less data preprocessing.

4. Trading Strategy

After configuring the machine learning algorithm and fine-tuning it, we shifted focus on creating a deployable trading strategy tool that could help users make decisions on the spot. The way we achieved this was by gathering the most recent data available and predicting the next day return. If the increase in price was over 2% of the historical price range of the selected stock, it suggested buying; if the decrease in price was over 5% of the historical price range, it suggested selling; and if the movement didn't cross these thresholds, it

suggested to hold. Since markets tend to move more when they lose than when they win, our tool considers a bigger margin for negative movements as well.

5. P&L and Past Prediction cross-validation

To further enhance the capabilities and usage of our application we designed two additional features: the first one performs a P&L Analysis on the historical information page for any stock in a selected range inputted by the user, so that he can understand past fluctuations and how these would have played out in the timeframe. The second one was a new page called predictions, where it is possible to select a stock and two dates and compare how the predicted stock returns fare against real-life values for that same period, while printing out the RMSE and MAE metrics. This last one gives users a glimpse of the work behind our model and trading tool and allows them to decide their risk affinity based on precision of past values.

6. Web Application Deployment

In order to present all the designed uses of our application to the user, the team used Python's library Streamlit which seamlessly integrates into the code and gives an easy to use markup "translator" interface. With it, a muti-page solution was developed, allowing users to change easily between the different functionalities we devised and use all features with at most 4 clicks.

7. Cloud-Based Deployment

Finally, the team deployed the streamlit application to the cloud using streamlit cloud service. This is a very straightforward way of doing so, as all that's needed is access to the GitHub repository that contains it and it gets deployed automatically, providing us with a shareable link that can be accessed from any device and at any given moment.

8. Challenges Faced

During the duration of the project one issue stood out above the rest: How to handle the share prices file due to its size. The team initially approached this issue by using GitHub's Large File System (LFS) to be able to commit and use it remotely. This, however, had to be changed as the file size pushed LFS's free tier limit to the end, and so the team decided to ignore the previous LFS commands and upload this file as a release using CLI to the repository. This ensured that the file could be accessed and called, but didn't incur in breaking LFS's limit.

Another issue was organizing the pages structure inside the repository so that Streamlit recognized it as a multi-page index. This was solved through ChatGPT promptly, as all that was needed was switching one of the files to the outside. Finally, the team struggled for a while with generating absolute paths that were contained in PATH for the scripts that ran preprocessing and financial calculations, as the multi-index structure tended to mess up with permissions and accessibilities.

9. Conclusions

After working on the model, the trading tool, and the web deployment, we can conclude the following:

1) Predicting stock prices is notoriously difficult, as XGBoost had the best performance but still had many problems when faced with big variations.
2) The size of the share prices dataset made deployment difficult due to loading times. We managed to cache the data once it's loaded on the page, thus reducing further reloading, but is still takes some time to load initially.
3) Streamlit is a great choice for building and handling webpages that feed the backend, as all inputs could be recorded and used. This opens a lot of future opportunities to create and deploy applications for multiple purposes.