NOVA IMS Information Management School

MDSAA

Master Degree Program in

Data Science and Advanced Analytics

Machine Learning Operations

Stock Prediction Pipeline with Kedro & Co.

Paulo Martins, r2015469
Beatriz Fonseca, r20201599
Chiel Groeneveld, 20240662

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

Project available on GitHub

INDEX

1. INTRODUCTION

This ambitious project leverages *Kedro*, *MLflow*, *Docker*, and *Prefect* to build a full machine learning pipeline — from data ingestion and preprocessing to model training, evaluation, prediction on unseen data, and data drift analysis – with the objective of determining if the *QQQ ETF* (NASDAQ's Exchange-Traded Fund (ETF).) obtain returns above average in the next 5-day period.

The dataset consisted of historical daily stock data, obtained from the popular **Yahoo Finance** website**,** on large-cap *NASDAQ* companies (i.e. *AAPL*, and *MSFT* for our proof of concept), as well as the *QQQ ETF*. Which represents the *NASDAQ-100* index and is widely used as a proxy for technological sector performance and overall market sentiment. Making use of this data, the goal is to predict whether the return on QQQ over the next 5 trading days will be positive or negative, thus enabling informed short-term investment decisions. With the main objective being to capture patterns in historical price movements and technical indicators that can signal short-term momentum or reversal.

Despite its potential use as support for tactical trading strategies and improved decision-making for investors seeking to optimize their entry and exit points. We surmise that the best use case for such a signal is for large financial operators such as banks, and very sophisticated investors to hedge their positions in the market, against potential financial downsides in their other ventures. As such, this proof of concept has the potential to unlock interesting avenues in Options trading, Fund and Wealth Management alike.

To evaluate model performance, the *F1-score* was chosen as the primary success metric, as it balances precision and recall, which is especially important in financial forecasting where both false positives (predicting a profitable return when it is not) and false negatives (missing a profitable opportunity) can be costly.

2. PROJECT PLANNING

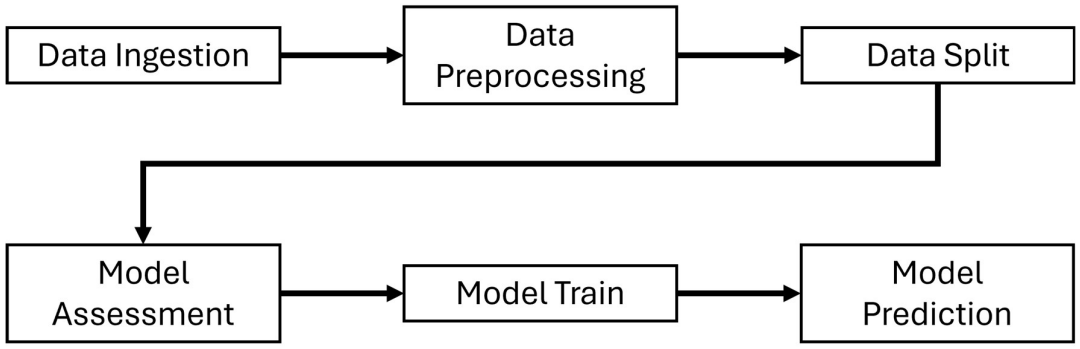
The project was organized using a sprint-based **Agile** methodology to ensure iterative progress and early feedback. The work was divided into four main sprints over eight weeks, each lasting two weeks.

- **Sprint 1 (Weeks 1–2):** Focused on conceptual design and a minimum viable prototype, including initial data collection, feature selection, and a Jupyter-based proof of concept.
- **Sprint 2 (Weeks 3–4):** Transitioned the prototype into a structured *Kedro* pipeline, establishing modular architecture and reusable nodes.
- **Sprint 3 (Weeks 5–6):** Integrated advanced model features and improved performance through optimization and validation and added *MLflow* for experiment tracking.
- **Sprint 4 (Weeks 7–8):** Prepared the pipeline for production by adding monitoring with Prefect, containerizing the application using *Docker*, and finalizing deployment documentation.

Throughout the project, a prioritized backlog was maintained using story points for task estimation, with regular sprint reviews and retrospectives conducted to guide progress. Pair programming was used to ensure high code quality. This structured planning approach enabled delivery of a robust, production-ready machine learning pipeline on schedule.

3. PIPELINE DESIGN

The image below and the accompanying descriptions provide an overview of the implemented *Kedro* project.



- **Data Ingestion:** Collects stock tickers using the ***Yahoo Finance API*** and creates feature stores for data using **Hopsworks** – this last one, which, implements our Great Expectations.
- **Data Preprocessing:** Performs data engineering and updates the feature stores, creates the target feature and creates the feature store, updates the data format based on the target and handles missing values.
- **Data Split:** Splits the dataset into train and test sets, this latter used as a hold-out on which the model will not train used only for predictions.

- **Model Assessment:** A set of models are compared for performance using a pipeline also containing feature selection and data scaling. The models are tested using cross-validation and optimized using *OpTuna*.
- **Model Train:** The champion model from the previous pipeline is trained on the entire train dataset.
- **Model Prediction:** The trained model from the previous pipeline is used to make predictions on the test dataset.

In addition, two more pipelines that follow outside of this general workflow were designed:

- **Data Unit Tests:** Uses Great Expectations to check the ingested data quality and formatting.
- **Data Drift:** Produces data drift results and dashboards for uni- and multi-variate analysis.

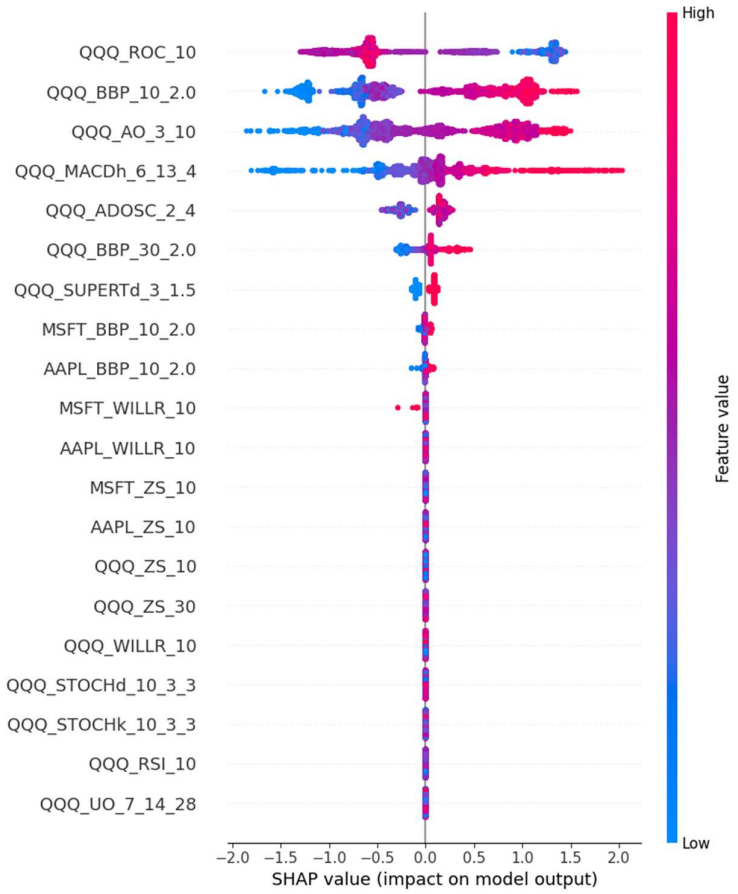
4. RESULTS & DISCUSSION

The collected data was initially clean and free of missing values, which facilitated the ingestion process greatly. However, the feature engineering routine implemented by the pipeline introduced missing values of several types, some were due to sliding windows that either look forward or backwards; while others were due to the nature of mathematical operations performed and for which no valid result could be obtained. To address this, some features were entirely removed, others were imputed with zeros and trailing and leading rows in the dataset containing missing values were dropped.

A brief exploratory analysis using the `Data Wrangler` extension in *VSCode* indicated that most features approximated a normal distribution, providing confidence to proceed with model development.

Model assessment included models such as *Logistic Regression*, *Random Forest*, and *XGBoost*, of which, *Logistic Regression* emerged as the best-performing model, achieving a validation **F1-score** of *0.8749*. The model was then retrained on the full training set, and a *SHAP* summary plot was generated to interpret feature importance. This analysis revealed that the *Rate of Change*, *Bollinger Bands*, *Awesome Oscillator*, and *MACD histogram* all indicators obtained from *QQQ* itself were found as most influential features in driving predictions. *(see the image below - page 5)*.

Holding these findings true – **with a rather large grain of salt as this is not an Econometrics paper**- it appears to reveal at least empirically that, most of the information about the next 5 days' worth of **average movements in the QQQ are already contained within the price** of the current day itself. Albeit, having tested with a small combination of stock tickers, this leaves open the possibility of an explanatory model that makes use of a richer set of features, and demands naturally more extensive, and rigorous testing.



Finally, the model was evaluated on the test set, achieving an **F1-score** of *0.8532*. which was found to be very significant and a source of much joy for the team, in terms of the quality of their work. Not only due to how high these tentative results appear to be; but also, because of the **absence of both overfit/underfit**, as well as **any leakages** in the pipeline.

As such, the team congratulates themselves for having been able to integrate *OpTuna* seamlessly and idiomatically, into the *Kedro* framework with excellent results.

5. RECOMMENDATIONS

Even though this project was commissioned by the professor as proof of concept, we would describe the current work as a ready to deploy early-stage prototype, given that much of it is already containerized, and ready to be served. Naturally, this does not imply that it is a finished product or close to a done deal, quite the opposite.

For instance, to tackle horizontal scaling of the dataset to include more stock tickers (each stock indicator can generate over 90 indicators), as well as vertical scaling with finer time-intelligence granularity Flask would have to eventually be adopted – as **PySpark** does not offer good native support for **Time Series Forecasting** - enabling a better exploration of the search space; presenting unique challenges in parallelization and code optimization.

Also worth noting is that the models used in the pipeline for prediction, despite providing good results at face value, were by no means state of the art, and as such, greater consideration can be put into the integration, training and eventual deployment of state of the art models. We expect these models which are yet to have been integrated tightly into frameworks such as **Scikit-Learn** to incur in increased development but could offer improved results.

Finally, we found ambiguity dealing with the **Hopsworks** *feature store* as we had expectations for our data at both the ingestion, as well as at the processing stages. The *update* pattern implemented to update the data expectations, after preprocessing, left us unable to parallelize two of our pipelines, as they depended on a `. *Json*` file to assist in versioning them, and for that reason, impeded us from containerizing (via Docker) our entire training orchestration. For this reason, our proposed alternative would be to run both pipelines sequentially and manage their relationship via a *Database*. This hypothesis also invites us to consider long term storage of all data, and storage in a more structure solution such as a lightweight DB such as *SQLite*.

Lastly, improving both the *Documentation* – generated rather crudely via Sphynx – as well as the, improving the coverage of the Integration and Unit Tests would be candidates for the immediate improvement. The former would assist in the onboarding of future workers, which could get to know the codebase more quickly, the latter, as our current coverage sits around *35%* for the entire `src/` codebase, and around *75%* for the crucial component outputs. Integration and unit tests offer a stable platform from which to develop, and could help mitigate potential issued before they arise, as the project prepares to expand and complexify, and as such, be crucial for its short to medium term success.

6. PROJECT PACKAGES

The project was developed using the following main packages:

Package	Version
docker	7.0.0
evidently	0.3.0
great-expectations	0.15.12
hopsworks	4.2.6
hsml	3.7.2
kedro	0.19.5
kedro-docker	0.6.2
kedro-mlflow	0.12.2
mlflow	2.11.3
nannyml	0.10.5
Numpy	1.24.4
optuna	3.6.1
optuna-integration	4.4.0
pandas	1.5.3
pandas-ta	0.3.14b0
prefect	2.20.20
pytest	8.0.2
pytest-cov	6.2.1
scikit-learn	1.4.0
shap	0.48.0
xgboost	3.0.2
ydata-profiling	4.6.0
yfinance	0.2.63

The complete list of dependencies can be found in the project root folder in the *requirements.txt* file.