Group Assignment: Development of an Automated Daily Trading System

Overview

This group assignment requires students to design and implement an automated daily trading system using Python. The system will comprise two primary components:

1. **Data Analytics Module**: This entails constructing a machine learning (ML) model for market movement forecasting. This module must work with at least five US companies.

2. **Web-Based Application**: A multi-page interactive web application, developed with **Streamlit**, which enables users to interact with the trading system, visualize predictive analytics, and comprehend the underlying methodologies.

Data

In this project, we will use data extracted from **SimFin** (https://www.simfin.com/). SimFin (short for "Simple Financials") is a platform that provides free and premium access to fundamental financial data for publicly traded companies. It allows users to retrieve historical and real-time financial metrics, making it an essential tool for financial analysis and quantitative trading strategies. SimFin offers structured datasets that include income statements, balance sheets, and cash flow reports, enabling users to develop robust predictive models and trading algorithms. By integrating SimFin into this project, you will gain hands-on experience working with real-world financial data, utilizing both historical records for model training and real-time API access for the web-based application.

Setting up SimFin platform

1. Register on the SimFin website (https://www.simfin.com/). To do it, just press "Get free account".

2. Once you are logged in, you will see an interface with a lot of options. Go to the left panel and you will see these two parts:

    1. **Bulk download**. This will allow us to download financial history data in one shot. This is necessary for the first part of the project, to train the machine learning model.

    2. **Data API**. This will allow us to access financial data in real-time, using the SimFin API. This is necessary for the second part of the project, the web app. Once we have our model trained, we will use the API to extract fresh financial information to feed our ML model to provide trading information.

Working with the SimFin API

1. Here you can find the documentation: https://simfin.readme.io/reference/getting-started-1

2. This API requires authentication, so you must provide the api-key on every request. To provide the key, just include this element in the header dictionary Authorization: api-key <here-your-api-key> .

3. SimFin provides a free tier (what we will use), so some limitations apply when using the Data API. Here you can check them: https://simfin.readme.io/reference/rate-limits. Concretely, the limitation is that **we cannot make more than 2 requests per second**. Take this into account when you develop the web app.

Project Requirements Definition

Part 1: Data Analytics Module

In this section, students will leverage machine learning techniques to develop a predictive model for stock market movements. The goal is to analyze historical stock price data, identify meaningful patterns, and implement an ML model capable of forecasting the next day's market behavior. This involves data preprocessing, model selection, performance evaluation, and the integration of trading strategies. The outcome will be a structured and data-driven trading decision-making framework, forming the backbone of the automated trading system.

1.1 Build an ETL to process input data.

This section focuses on the **Extract, Transform, Load (ETL)** process, which is crucial for preparing data for analysis and machine learning modeling. Students will be responsible for extracting historical stock market data and transforming it into a clean and structured format. Emphasis will be placed on handling missing data, normalizing values, and ensuring the dataset is optimized for predictive modeling.

Requirements

- Download and process financial data.
- Use Pandas or Polars library.

Instructions and suggestions

1. Download all necessary data from the SimFin **bulk download** section.
    1. **(Mandatory) Share prices**. This file includes all the daily share prices for 5 years. This dataset is the main dataset because it contains the minimum data you will need to define the target of your ML model. The project can be done only with this dataset.

    2. **(Mandatory) Companies**. This file includes just the list of companies, with its related information like the ticker name, the company name, the industry sector, the number of employees, etc.

    3. **(Optional) The rest of the files**. The rest of the information is not mandatory but can be useful to enrich your set of features for your ML model. There is information about the income statements, balance, etc. These fundamentals can be used as well as the input of your model.

2. Build your ETL to process data.

    1. First, decide what library to use. Polars or Pandas. In this case, we have a considerable amount of data, so if you are experiencing issues with Pandas, you can try Polars.

    2. Design and develop all transformation steps in a Jupyter Notebook. Then pass all these transformations into an automated Python script.

    3. Build your ETL process just for one company and then execute it for all companies you want to include in your application. You can build a unique ETL script with an input parameter specifying the company to use.

    4. Start simple, using the minimum set of data (just Share prices) and a minimum set of transformations.

    5. Make ETL work for 1 company and then move to the next step. Try to build the entire trading app with just one company. Then go back and include the rest.

1.2 Build a Machine Learning Model to Predict the Next Market Movement

This section focuses on developing a **machine-learning model** that, taking historical market data, makes predictions for the next trading session. The model's output will serve as the foundation for making informed trading decisions, forming an integral part of the automated trading system.

Requirements

- The model must make predictions daily.
- The model must make predictions one day ahead.
- The machine learning model must be built using Python.

Instructions and suggestions

1. Decide how to build the ML model. To build a machine learning model, there are several options, depending on the complexity you want. Here are several options.

    1. **Predicting if the price will rise or fall.** This is the simplest scenario. This involves a classification-based model. Given historical data, the model will predict if the price rises or falls the next day.

    2. **Predicting the next-day price.** This option requires a more sophisticated approach due to the complexity of stock market dynamics. This involves using regression-based models.

    3. Others. You can invent more options to define your ML models, for example:

        - Instead of predicting if the price will rise or fall, you can include more categories like high rise, low rise, stay, low fall, and high fall. This approach will involve the construction of a multi-class classification model, with more flexibility than the previous binary classification model.

2. Remember that the output of the model will be used later to make the trading decision (to buy, to sell, or to hold) if you decide to do it.

3. Start developing the code in a Jupyter Notebook, and then build a script to generate your model based on input data (the output of the ETL)

4. Start simple, don't complicate too much. The main goal is to make everything work.

    1. Start building a simple Logistic Regression that gives you if price will rise or fall.

    2. Export the model to make real-time predictions in the web application.

    3. Then, when everything is working, let's come back to improve the model.

1.3 (Optional) Build a Trading Strategy Based on Market Movements

A trading strategy is a set of predefined rules that determine when to buy, sell, or hold assets in response to market conditions. This section focuses on designing a systematic approach to making trading decisions based on the output of the machine learning model. The strategy should integrate predicted market movements, and any other business knowledge to maximize profit. Students are encouraged to backtest their strategy against historical data to evaluate its performance before implementation in a live environment.

Instructions and suggestions

1. Define your trading strategy. There are endless ways to set a trading strategy. For example, imagine we are using the simple binary classification model that predicts if the price will rise or fall. Here are some examples.

    1. **Strategy 1: Buy-and-Hold.**

        - If the price rises, I buy 1 share.
        - If the price falls, I hold.
        - I will sell only when I reach a predefined level of profit.

    2. **Strategy 2: Buy-and-Sell.**

        - If the price rises, I buy 1 share now.
        - If the price falls, I will sell 1 share now.

2. Apply the strategy to the output of the model. The output of the strategy must tell us 1 of these actions.
    1. If BUY, and how many shares (or how much money)
    2. If SELL, and how many shares (or how much money)
    3. If HOLD
3. Again, start simple. Define simple rules just to make everything work. Then you can come back and improve your trading strategy.

Part 2: Web-Based Trading System

This section focuses on developing a fully functional web-based trading system that integrates the machine learning model and API wrapper into an interactive user interface. The goal is to provide a seamless experience for users to analyze market trends and view the model-generated signals generated by your ML backend. Additionally, the system must be deployed to a cloud platform to ensure accessibility and scalability. Emphasis will be placed on user experience and efficient data retrieval.

2.1 Python API Wrapper for SimFin

Students will construct a Python API wrapper to interact with SimFin. The goal is to simplify the interaction with the API just by using Python. This wrapper consists of a set of classes to retrieve share prices or any other useful information.

Requirements

- Implement an object-oriented code to extract data from SimFin.
- The library must allow users to retrieve stock price data for specified companies in the specified period.

Instructions and suggestions

1. Define the structure of the API. Here I provide an example of a simple Python wrapper for the API. Take it as a suggestion and simplify or complicate it as you consider.
    1. Build a class PySimFin with the following capabilities.
        1. Constructor. This will create a new instance for the class to interact with the API. In this constructor, you can initialize the base endpoint and the header with the "api-key"
        2. get_share_prices (ticker: str, start: str, end: str). This method will return DataFrame with all prices for the provided ticker in the provided time range.
        3. get_financial_statement (ticker: str, start: str, end: str). This method will return DataFrame with financial statements for the ticker provided in the provided time range.

2. Include a logging mechanism to facilitate the development process.
3. Include error handling (exceptions) mechanisms.

2.2 Web Application Development

Students will develop an interactive web application with the following components:

2.2.1 Home Page

Requirements

- Present an overview of the trading system and its core functionalities.
- Provide information about the development team.
- Explain the system's purpose and objectives clearly for the end-user.

2.2.2 Go Live Page

Requirements

- Implement a **stock ticker selector** to allow users to choose from available companies.
- Display **real-time and historical stock market data** using the Python wrapper for SimFin.
- Apply the ETL transformations and predictive model to data extracted from SimFin.
- Show **model-generated trading signals** (if the price rises or falls, or suggestions about buying or selling if you have a trading strategy).

Optional (if you built the trading strategy)

- If you built a trading strategy, you could provide an interactive backtesting simulator allowing users to evaluate historical trade scenarios based on model predictions.
    - For example, with a given trading strategy, what would happen if I had started investing in AAPL the 01-01-2020? How much money would I have now?

Instructions and suggestions

- You can use any web application technology to build your web app, but the recommendation is that you use Streamlit, which is the library we see during the semester.
- To build a multi-page app in streamlit you can follow these instructions from Streamlit docs: https://docs.streamlit.io/get-started/tutorials/create-a-multipage-app
- Additionally, you can include more pages to support your work. For example, you can include a page to illustrate how you did all the ML process: feature selection, data processing, evaluation methods employed.

2.3 Cloud-Based Deployment

Requirements

- Deploy the Streamlit-based web application on **Streamlit Cloud** or an equivalent cloud hosting service (AWS, GCP, Heroku).
- Ensure **public accessibility** of the deployed application without requiring local installations.
- Provide a **public link** to the final deployed system.

Deliverables

- **GitHub Repository:** The group must provide a link to a public GitHub repository with the project. It must contain the following:
  - Well-documented and structured Python source code.
  - A requirements.txt file with all used dependencies and its versions.
  - A README.md file with general information about the project.
- **Executive Summary:** Brief document explaining briefly how you did every part, challenges, and conclusions.
- **Live Web Application:** Hosted on a cloud platform and accessible via a public link.
- **Video-Presentation:** Record a max. 10-minute group presentation of the final application. This video must contain the following;
  - **A summary of Part 1:** Include a summary of the work done in this analytics part. Include things like what data you used, how you processed it, what ML model you chose, evaluation metrics, and backtest simulations of your trading system.
  - **A web app demo:** Include a demonstration of how your application works. Provide different tickers and extract and show the interface of your application and the investment recommendations.

Grading

Grading is divided into two parts: project and presentation.

Project

- 50% of total grade. All members of the group will have the same grade.
- The project will be evaluated as a whole, depending on the complexity, methodology, and web application (look-n-feel, structure, etc.), video-presentation, etc.

- The quality level of ML model predictions or the level of profit made by your trading system is not part of the evaluation. Predicting the stock market behavior is a hard task and is not the goal of this project.

Participation

- 50% of the total grade. It is individual and depends on your contribution to the project.
- **All members of the group must participate in the project** to obtain a grade in this part. You should participate in the elaboration of the project to have a good performance here.
- The grade will be related to the amount of implication you have in the project

Optional Task

There is one optional task in the project: the trading strategy. This additional task, if done, will increase the grade up to 1 point in the final grade. So, the maximum grade attainable in the project is **11**.

Tips for Group Organization

You must act as a professional multidisciplinary team. Here you have some suggestions on how you can organize your group.

1. Sit down together and analyze what to do. Ensure you understand everything.
2. Divide the group into **2 teams.**
    1. **ML team**. In charge of the first part. The responsibility of this team is to train ML models to predict the market and establish trading strategies. If you have someone knowing about finance, much better.
    2. **DEV team**. This team is in charge of developing web applications, including API communication, the Streamlit web app, and the deployment to the cloud.
3. Designate a **Group Leader**. This is the person of reference to ask about what the rest of the people are doing. The group leader can work in any team but must be aware of what the other team is doing to not repeat work.
4. **Work incrementally, in iterations**. Build the first functioning version of the product as soon as possible. To achieve this, you need to start simple. Example:
    1. In ETL, don’t use all data sources at the beginning.
    2. In the ML part, don’t try to implement complicated neural networks as the beginning. You can start with something simple.
    3. In the API, start developing just the method to access the data you need.
    4. In the Streamlit web-app. Don’t worry too much about the look and feel, but on showing correctly the information about your ML backend.
    5. Do the optional tasks, if you want to do them, at the end.