SCHOOL OF
HUMAN SCIENCES
& TECHNOLOGY

# Development of an Automated Daily Trading System

Python II

**MASTER IN BIG DATA & BUSINESS ANALYTICS**

**GROUP 5**

Anastasia Chappel
Felix Goossens
Marta Pérez
Sergio Lebed
Youseff Hakim

## Index

# 1 ETL Process for Extracting, Transforming, and Loading Data

## 1.1 Overview:

The first step in our development of the automated daily trading system was to build an ETL pipeline for processing historical stock data, for it to be used for machine learning model training and trading signal prediction.

## 1.2 Step 1: Extracting Data

We first extracted the data from the SimFin platform. For the data extraction we used the SimFin API. The data we selected for the ETL process was the company basic information, their share prices, data about their income, and their balances.

## 1.3 Step 2: Transforming Data

Once we extracted, the data, we carried out multiple transformations from a selection of airline companies. We applied data cleaning (handled missing values, duplicates, and ensure consistency), normalization to ensure all financial data was standardized, and feature selection (filtered out features that might not be helpful or redundant).

## 1.4 Step 3: Building the script

We built a script that is responsible of extracting and processing financial data for stock market analysis, focusing on share prices, income statements, and balance sheets for the companies. The script utilizes the SimFin library to fetch the data. It is focused on automating the process of downloading and preparing financial data for machine learning or further analysis.

### 1.4.1 Environment Setup:

**dotenv**: we used the dotenv library to load environment variables, such as the API key from a .env file.

**SimFin Setup:** the sf.set_api_key() method is used to authenticate the API requests, while sf.set_data_dir() specifies where to store the downloaded data.

### 1.4.2 Data Loading Functions:

The script defines three main functions for loading different types of data:

- load_share_prices(ticker):
  - This function retrieves daily share price data for a specified company (ticker) using the SimFin API.
  - The data is filtered to include only the relevant company's ticker symbol, and the DataFrame is returned for further processing.
- load_income():
  - This function fetches annual income data. It includes columns for Revenue, Net Income, and Report Date.
  - The unnecessary columns are dropped, and the filtered data is returned.
- load_balance():
  - It follows a similar logic to the income function. This function loads balance sheet data for US companies.

### 1.4.3 Merging:

A function to merge the price, income and balance data was also built so that we get an ensembled result.

### 1.4.4 Normalization:

We also imported MinMaxScaler from sklearn.preprocessing to include a normalization process in the pipeline. This step is useful for preparing data for machine learning models, especially when working with features of different scales.

### 1.4.5 Retrieving the data

We also created a function designed to load the daily share price data for a given company, identified by its ticker symbol, and another one for retrieving the data by looking at the companies' name.

# 2 ML Model:

1. **DataLoader Class**:

**Function**: The DataLoader class is responsible for loading financial data from the SimFin API. It provides two primary methods:

One for loading **company information**.

Another for loading **share prices**.

2. **AccessCompanyData Class**:

**Function**: The AccessCompanyData class enables users to access company-specific data by providing the company name. It utilizes the DataLoader class to fetch the required data for the given company.

3. **CleanCompanyData Class**:

**Function**: This class handles the cleaning and merging of financial data:

It merges the **share price**, **balance sheet**, and **income statement** data.

The data is aligned by the **report date** so that all data points across the datasets match.

It uses the DataLoader class to fetch the financial data and processes it to return specific company information.

4. **EnhancingFeatures Class**:

**Function**: The EnhancingFeatures class adds additional **technical features** to the final DataFrame that are commonly used in stock price prediction:

**RSI (Relative Strength Index)**: Measures the speed and change of price movements.

**Moving Averages**: Includes both simple and lagged moving averages.

**Bollinger Bands**: A volatility indicator that measures price fluctuations.

**Volume Normalization**: Standardizes trading volume for better comparison.

**Price Trend Calculation**: Computes the trend of the stock price over a 10-day period.

**Handling Missing Values**: Missing data points are filled to ensure completeness.

5. **MarketAggregator Class**:

**Function**: Aggregates the individual company data into a market-level dataset by grouping the data based on the date.

6. **Data Handling Functions**:
   - **get_company_data**:

Uses the AccessCompanyData class to retrieve company-specific data (e.g., share price, balance sheet, income statement).

The CleanCompanyData class cleans and merges this data.

The EnhancingFeatures class adds the necessary technical features.

   - **plot_company_data**:

Visualizes the processed data (e.g., closing stock price over time) using charts.

- **run_ml_model**:

- Uses the preprocessed data to create features and labels for training a machine learning model (Random Forest Regressor and Classificatory).
- The target variable (y) is Return_1D**,** which represents the stock return for the next day.
- The features (X) include data columns relevant to stock returns prediction like Close**,** Volume**,** MACD**,** RSI, etc.
- 80% of the data is used for training, and 20% is used for testing**.**
- The model is evaluated using the R-squared score to assess how well the model fits the actual data.

## 3 Web-based trading system:

### 3.1 Visual and Interactive Features:

**Custom Styling:**

The app uses CSS styling to enhance the UI, adjusting background colors, fonts, titles, and descriptions for a polished look.

**Page Navigation:**

The page_index in st.session_state tracks the current page selection (e.g., "Home", "Go Live", "ML Predictions Overview, “Comparisons”).

The st.sidebar.radio widget enables users to easily navigate between different pages, with page_index keeping track of the selected page.

### 3.2 Home Page

Our home page included a printed welcome message, along with the description and main functionalities of our app. Furthermore, we included pictures of the team and of the logos of the selection of companies we worked with.

### 3.3 Airline Selection and Real-Time Market Analysis:

On the "Go Live" page, users can select different airline stock options (like American Airlines, Delta, etc.) via buttons.

Users can filter the market data by date range and time range (1D, 5D, 1M, YTD, Custom).

Once the data is fetched, it is visualized in an interactive chart (using Plotly) that shows the stock price over time.

The app also displays metrics such as the current price, daily price range, and trading volume.

#### 3.3.1 Interactive Tabs:

**Tab 1 (Interactive Chart):** Creates a line chart using Plotly (px.line) to visualize stock prices over time.

**Tab 2 (Historical Data):** Displays the stock data as a table, providing details such as the stock's open, high, low, close, and volume values.

### 3.4 ML Predictions Overview:

When the "Run ML Predictions" button is pressed, the app:

- Trains a machine learning model using the run_ml_model function.
- Loads a line graph with the historical stock price data for a specific ticker or for the whole market over a user-defined date range together with a predicted value for the closing price tomorrow.
- Prints a message with the predicted value inside a square in a green color if the price increased or red if it decreased and the trading signal.
- Displays a second line graph with the RSI for selling and buying decisions.

### 3.5 Comparison page

It displays a line-graph with the same data filtering functionalities as in Go-Live page, with the difference that from selecting different stocks you get a multiple lines graph.

## 4 Conclusions

Although challenging we found this project significantly useful and interesting and each of us learned a lot from it. We struggled mainly in the process of building the web app and making it operative in each of our laptops due to library compatibility, along with other issues. However as mentioned, we believe this project was the perfect way to wrap our Python learnings in Ie as it successfully introduces all the concepts learned and aggregates them together.