Technical Exam Solutions

1: Exploratory Data Analysis

A Jupyter Notebook was prepared (EDA.ipynb in the repository) with the Exploratory Analysis that was carried out. In it, the dimensions of the dataset, the ranges and distributions of the variables were analyzed. In addition, invalid or duplicate values of the variables were checked, dropping them if necessary. Seaborn was then used to detect relationships between variables to see if aggregation was possible. This provided a first approximation of the bot's behavior when detecting the presence of outliers.

Subsequently, some metrics such as the values of the transactions and the outcome for each instant of time were calculated. Also, by plotting the total amounts of sales and purchases and plotting them on a pie chart, a first idea of the negative outcome result was obtained.

The dataset was then grouped and aggregated by hours and minutes, and alternative plots were obtained using the Plotly Express tool. A grouping by purchasing and sales chains was also carried out, to detect the most influential purchasing and sales chains. To simplify the analysis, only the 50 most influential buying or selling chains were included. Finally, the final outcome metrics, the size of the longest sales chain, and the price difference at the beginning and end of that chain were obtained.

2: Proof of Concept

Taking the dataset cleaning and aggregation operations carried out in the EDA to obtain metrics and graphs, a Python script was written to perform the extraction, transformation and loading of the results called "etl.py".

The system structure for this Proof Of Concept performs batch processing. A Flask server receives requests to send files in ORC format from a client (in this case implemented on the same machine). For each trade file received, the server adds the dataset to a trades table in a MySQL database. In addition, it performs ETL on the new dataset, and adds it to the previously processed data, storing it in ORC format to make it available to the visualization tool.

This is done with the files in the “src” folder of the repository, which contains:

1. A Python script that implements functions for connecting and querying the MySQL database using the MySqlConnector library (connectDB.py).

2. A Python script for the Flask client that sends the ORC files located in the directory (client.py).

3. A Python script that implements the Flask server that receives the requests and calls the necessary methods to insert the received dataset into the SQL database, and in turn calls the ETL method and stores the result in pandas dataframes in files ORC (server.py). The server uses the methods of the “connectDB.py” script to connect to the DB, create the table in the DB and insert the new data in bulk (bulk load). The ETL method is in the "etl.py" script. The transformed and aggregated data is exported to ORC files in the “Aggregated Datasets” folder of the repository.

Finally, the visualization tool was implemented in the Python script “Report.py”. The Plotly Dash tool was chosen because it allows you to generate interactive charts and add various controllers by integrating them into an html dashboard. In addition, the graphics can be exported in this format for distribution in other ways.

The "Report.py" Python script loads the aggregated data into the ORC files, and generates the report dashboard. The dashboard has a callback to periodically update itself, rereading the data to incorporate new results if available. In addition, a date selector allows to obtain previous reports.

3: High-End Solution

In this case, a solution with data processing based on Streams is analyzed. Figure 1 (Arq.pdf file in docs) shows the proposed architecture. The main elements that make it up are analyzed below, as well as the changes that they would undergo with respect to the implementation of the Proof Of Concept.

Stream Processing:

The bots send the transactions using Kafka producers, posting them to a topic. The system receives the stream data through a pySpark consumer, which inserts the received data into the SQL transactional database and also processes it, generating the aggregate files used by the visualization and reporting tool.

Should ingestion and ETL tasks be too heavy a load for a single node, they can be distributed across nodes in a cluster using Apache Spark Stream and pySpark's RDD capabilities.

Database size:

In the case of the Proof Of Concept, to analyze the disk space and the delay in the insertions in the DB, datasets similar to the one provided (trades.orc) were generated, with an average size of 1 million rows +-100,000, keeping the 5 original columns. The delay in the insertion of these datasets in the DB was measured, returning a value of more than 6 minutes in the last cases. After inserting 30 datasets in addition to the one provided, the database table has approximately 31 million rows, as expected, and its size on disk is 1.52 GB.

If it is considered that the High End solution will receive data from multiple bots in charge of different markets, the size is multiplied by the number of bots, and by the number of months that the system is expected to be operational. Currently the system does not allow searching the database to perform other types of aggregations, but to add more features in the future, it will be necessary to partition the database: the first criterion is to partition by bot, the second will be to partition by month.

Storage of aggregated datasets:

The aggregated datasets are part of the Data Warehouse, but they are relatively small in size, so the addition of multiple data ingestion channels does not imply an implementation change in the first instance. They can also be partitioned by date (monthly) and by bot, so that queries to retrieve certain portions of them are faster.

Dashboard generation:

Controls must be added to the tool proposed for the Proof Of Concept for the selection of data from the bot to be analyzed. Likewise, more selectors can be incorporated to analyze ranges of dates, or subsets of transactions that require analysis at a higher resolution. In addition, it may be necessary to provide the option to compare the evolution of variables of the different markets, as well as the performance of the bots. In addition to the report dashboard, metric reports can be generated to be sent by email or through an application.

Regarding the scalability of the visualizations generated with Dash, the transactions for a day according to the size of the dataset provided cannot be graphed without aggregation. If the implementation of this functionality is essential, possible alternatives are to run the application in a distributed way with Dash Enterprise, or to use another visualization library.