The ETL Project

By,

Shagufta Zareen, Sanket Mishra, Steve Novis, and Shishir Tewari

Data analysis is an important part of an organization/business because it shapes the businesses decisions and their success or failure. However, just having the metrics is not enough, the data needs to be relevant to the organization and their goals. In this instance, we are simulating at a very elementary level what a hedge fund or investment bank may use in order to incorporate data analysis on the S&P 500 index and its futures contract.

Data such as these are the lifeblood of some firms, where they will utilize to create various algorithmic trading strategies such as spread trades, arbitrage opportunities, and other automated quantitative trading strategies. With that said, before being used in live trading, some firms opt to back test their strategies, however, one caveat of this is a large amount of data is required in order to test the validity of the model. Even with technological advances in storage capacity in data, it still is not robust enough to store significantly large amounts of data. Therefore, the use of databases plays a critical role in the storage and extraction of such datasets for firms. It was our goal to demonstrate in an elementary example how ETL could be utilized in this fashion.

**ETL** is a process in data warehousing responsible for pulling data from the source systems and placing it into a data warehouse. ETL involves the three components which compose its acronym – extract, transform, and load. In terms of our project, we first discussed what tools we wanted to use and what data we wanted to extract. In terms of **extracting** our data, our first decision came to what we wanted to pull. Initially, Shishir mentioned healthcare or medical data. However, we decided not to go with this topic as it requires a great deal of knowledge of what to do with that data and what it could mean for analyzers, so we pivoted. Eventually, we settled on financial data, as financial data is immense in volume, easily accessible, relevant to the market, and simply interpretable. After looking for compatible and detailed data sources, we settled with Yahoo Finance and Quandl. After we decided what data we wanted to use and where to get it from, we had to decide what tools we wanted to implement.

Out of all coding languages, as a team, we decided python was best suited for the task because it allowed not only the ability to easily convert CSV to a data frame but with the use of pandas and other libraries it would easily allow us to complete a multitude of the task under one language. In addition, it is extremely replicable for similar financial datasets, that have the same, generic structure. This also led us towards using python as we could incorporate the pandas’ module to make the transforming and loading process more straightforward. Both data sources were exported in CSV format for ease of use with python and pandas data frames. Although we’ll discuss it later, the stock data frame also had to be spliced to align with the futures data frame for merging. Lastly, we decided to use an RDBMS database, namely MySQL, to load our databases. Since the data is very similarly structured, MySQL made visualizing and joining data much easier.

In terms of **transforming** the data, we implemented several changes. Firstly, we converted the CSVs to Data Frames (Stocks and Futures). To do this, we simply utilized the pandas read\_csv function. This function allows us to replicate the same revisions on multiple similar datasets to potentially generate more financial analysis. The next step we implemented was filtering the raw data. We selected certain columns to load into the final data frames from our raw data. As stated earlier, financial datasets have relatively the same structure. Typically for daily data, it is always expected it will have an opening, close or adjusted close (settle), high, and volume columns. Since both datasets are analogous by nature, they are in themselves different entities and therefore we chose columns that we could use from both datasets that could work in conjunction with one another for analysis. With this in mind, we thought it was necessary to keep the open, adjusted close (or settle), and volume columns from both raw datasets.

After getting the columns we had predetermined, we created a new column (Stock Returns and Future Returns) for each data frame and calculated their percentage returns from the previous day. Of course, having the prices are beneficial and provides the initial infrastructure to running further, complex analysis using statistical methods. At the most elementary level, percent return is a key metric for analyzing the daily performance of the index and futures contract from financial data. That being said, we felt it was necessary to generate and include this column in both our data sets.

Once we had all the columns we wanted in both data sets individually, we renamed the columns so that it would be easier to identify the data once merged. As the datasets are similar, they had nearly identical column names. We appended Stock or Futures to the column name so that the data would be more distinguishable and less confusing once merged. Another crucial update we did, was sorting the values in each data frame based on ascending or descending data. Once we extracted the data, we realized that one data frame went up from 2017 while the other went down from 2018. To fix this, we simply changed one data frame to match the other. We decided to go with ascending order, as the endpoint for graph visualizations would show you trends to the current date.

Another note on the date column is that the stock raw data wasn’t contained to the range that we had chosen for analysis and instead gave us over 10 years of trading data. To match up the dates for a merge in MySQL, we spliced the stocks data frame to align with our future dates. The final edit that we made on the data was cleaning it up for output. We decided to keep the format in the date column the same. For our open and close column in both data frames, we utilized the map function to add a $ in front of all stock and future prices. We also added a thousand separator in the volume column. Lastly, we added a % and multiplied the returns by 100 to get it into percentage format. Notedly, the map function does change these columns into strings, but this change was purely aesthetic. If the data frames were needed for analysis (and required to stay as float types), these map functions would not have been used, or the uncleaned data frames would be used for merging. Again, these changes were done for readability and to help the analyzer get a better comprehension of what each column was suited towards.

In terms of how we **loaded** the database, we used the two data frames and loaded them in MySQL to store. Stocks and futures both datasets have structured and normalized data. This gives us more weight to utilize RDBMS databases (e.g. MySQL) vs NoSQL databases (e.g. MongoDB). Analyzing and visualizing data is relatively easy in structured and relational datasets compare to non-structured data. Additionally, as we had no sensitive or personal information, we didn’t have to incorporate the Flask framework.

We have used MySQL Workbench to visualize and analyze data of both datasets. We have used date columns in stocks and futures datasets to merge and analyze values side by side and identify relationship and differences within the same set of attributes. Instead of merging the data frames in python through pandas’ functions, we have also created simple join queries in MySQL workbench and confirmed merged dataset values. We did this as it is more relevant in the industry to merge your datasets through SQL.

Upon loading the Pandas data frames, we used the to\_sql method to load data into the data-base and the read\_sql\_query method to select and verify the data. We created tables using SQL Alchemy metadata reflection and used the filter functions to create database joins. After creating joins in SQL Alchemy, we converted the result back to the Pandas data frame for analysis and better visualization on the Jupyter notebook file. The final result of the database is as follows:

