**Technical Report**

James Barraza, Zach Childers, Benji Nguyen

The goal of this project was to create an ETL pipeline for financial data on S&P 500 companies.

**Extract**

We used two data sources:

1. A public CSV on Datahub that contained high-level financial information on each S&P 500 company, including their sector and various statistics.
2. The AlphaVantage financial API which we used to grab monthly closing price information on each of the S&P 500 companies. The API returns information in the form of JSON.

Reading data from the CSV was simply a manner of converting the CSV to a Pandas dataframe.

For the AlphaVantage API, we created a loop that could collect monthly closing prices from October 2019 to March 2020 for each company. Because there is a limit on the number of API hits for free accounts, we had to stagger calls to get around the time restrictions. We then parsed the JSON and grabbed the necessary information to construct a CSV/Pandas dataframe that contained exactly what we wanted.

To query for the specific JSON we want, we pass in the stock symbol and the month we’re interested in into the API and pull out the element in “Monthly Time Series” “[Month]” “4. close” in the JSON. We then put that information into our skeleton dataframe, which has a row for each stock and a column for the symbol and each of the six months.

**Transform**

For each data source, we generated a dataframe in Pandas and programmatically generated an integer primary key using the row index.

For the CSV, we dropped all the columns we didn’t want and kept columns for the symbol, company name, sector, current price, price per earning, dividend and yield, and yearly high/lows. We then created our own column that took the differential between the yearly high and low to measure how volatile the stock for that company was. We then applied renaming to the columns so that it could be formatted into our Postgres SQL table.

For the API JSON, we simply took the information we cared about (monthly closing price per company for each of the six weeks), and renamed the columns from date to something more SQL-friendly. We also created a differential column that took the difference between the most recent closing price in our data (in this case, end of March 2020) and the oldest date (October 2019 here). We also dropped any rows that had empty data.

**Load**

We created a Postgres database (named ‘finance\_db’) which contains two tables, each reflecting the data we grabbed from one of our data sources:

-A table called “company\_info” which contains the high-level company information we grabbed from the public CSV on DataHub.

-A table called “closing\_prices” which contains the specific month-by-month closing price information we grabbed from each company off of the Alpha Vantage JSON.

We used Postgres because our data had well-defined columns we could reliably pull. Had the information been more variable, we may have opted for the free-form MongoDB. But because we had specific columns we knew we needed, we chose to go with Postgres.

Our columns were as follows:

company\_info:

id INT PRIMARY KEY: An ID column.

symbol TEXT: The ticker symbol.

company\_name TEXT: Name of the company.

sector TEXT: Which of the eleven sectors the company falls under.

price INT: Price at the time of data collection.

price\_earnings INT: Price per earnings.

dividend\_yield INT: Dividend yield.

year\_low INT: Lowest price over the last 52 weeks.

year\_high INT: Highest price over the last 52 weeks.

year\_diff INT: Differential between high and low.

closing\_prices:

id INT PRIMARY KEY: An ID column.

symbol TEXT: The ticker symbol.

date\_2019\_10 INT: Closing price end of October 2019.

date\_2019\_11 INT: “”

date\_2019\_12 INT: “”

date\_2020\_01 INT: “”

date\_2020\_02 INT: “”

date\_2020\_03 INT: “”

diff INT: Differential between first and last period’s closing price.

The data was loaded in via SQLAlchemy. One could theoretically join these two tables on symbol.

**Tips & Tricks**

-You can programmatically generate an integer ID primary key pretty easily using numpy. This might be helpful if your dataset didn’t come with one and you might want one.

-Find an API! This makes it easier to pinpoint exactly what data you’re going to be pulling from a website with tons of data. This will make transforms easier.

-Make use of the sleep function if your API has time restrictions.

-Dig a lot on datahub, Kaggle, or other websites. There are some handy CSVs with a lot of information, and you may be able to skip doing any taxing pre-processing.