Stock Data ETL Report

Introduction:

The purpose of this ETL report is to query the APIs from Finnhub and AlphaVantage to read in both real-time and historical data for a collection of sixteen stocks and cryptocurrencies into dataframes to then be loaded into an SQL database. A Kakfa pipeline will be used to funnel the data from a producer to a consumer via Databricks notebooks. The stocks are Visa, JPMorgan Chase, Bank of America, Mastercard, Apple, Microsoft, MGP, Quaker Food, Comcast, Version, AT&T, T-Mobile US, Amazon, Walmart Home Depot, and Costco. The Cryptocurrencies are Bitcoin, Etherium, Dogecoin, and Litecoin. In addition, we will be pulling in economic survey data from the US Census Bureau for the purposes of cleaning.

Data sources:

Finnhub API base:

https://finnhub.io/api/v1{arguments}?token={token}

AlphaVantage API base:

http://alphavantage.co/query{arguments}&apikey={key}

US Economic Census Data:

https://data.census.gov/cedsci/table?q=ECNNAPCSIND2017.EC1700NAPCSINDPRD

Extract:

Real-Time Stock Data:

1. Set up a producer class in a Kafka Databricks notebook and connect to the cluster.
2. Define a topic name in which to send and later extract the data.
3. For each ticker in the stock list return a quote using the API call: *https://finnhub.io/api/v1/quote?symbol={*SYMBOL*}&token={API TOKEN}*

   Where SYMBOL is the given stock symbol and API TOKEN is the token given to allow access to the API by finnhub.

4. Define each returned API request as a dictionary with *json.loads()*.
5. Store each dictionary from step 4 in a list.
6. For each item in the list, use the *produce()* function to send the message to the topic.
7. Create a new Databricks notebook, within which define a consumer class and connect to the cluster.

8. Call the the *poll()* function on the consumer to populate a list with messages in the topic created from the producer class created in step 1.
9. In the consumer, create a dictionary populated with values from each message in the list from step 7.

Historical Stock Data:

1. Set up a producer class in a Databricks notebook and connect to the cluster.
2. Define a topic name in which to send and later extract the data from.
3. For each ticker in the stock list return a quote using the API call: *https://alphavantage.co/query?function=TIME_SERIES_DALY&symbol={SYMBOL}&apokey={KEY}*

   Where *SYMBOL* is the given stock symbol and *KEY* is the key given to allow access to the API by AlphaVantage.

4. Store the returned API call in a dictionary with data from keys 'Date', 'Ticker', 'Open', 'High', 'Low', 'Close', 'Volume'.
5. Append the dictionary to a list of dictionaries.
6. Create a dataframe to hold the data in the dictionaries with columns 'Ticker', 'Date', 'Open', 'High', 'Low', 'Close', 'Volume'

Historical Crypto Data:

1. For each crypto in the stock list return a quote using the API call: *https://alphavantage.co/query?function=DIGITAL_CURRENCY_DAILY&symbol={SYMBOL}&apokey={KEY}*

   Where *SYMBOL* is the given crypto symbol and *KEY* is the key given to allow access to the API by AlphaVantage.

2. Store the returned API call in a dictionary with data from keys 'Date', 'Currency, 'Open', 'High', 'Low', 'Close', 'Volume'.
3. Append the dictionary to a list of dictionaries.
4. Create a dataframe to hold the data in the dictionaries with columns 'Currency, 'Date', 'Open', 'High', 'Low', 'Close', 'Volume'

US Economic Census Data:

1. Create a storage container within a datalake.
2. Download the .csv file from the US Census website and upload it to the storage container.
3. Create a new Databricks notebook and create a mountpoint to the datalake.
4. Define a pandas dataframe as the data within the .csv file containing the economic census data using *spark.read.csv()*.

Transform:

Real-Time Stock Data:

1. Rename the keys in the dictionary:
    a. ‘c’ to ‘Close Price’
    b. ‘o’ to ‘Open Price’
    c. ‘h’ to ‘High Price’
    d. ‘l’ to ‘Low Price’
    e. ‘d’ to ‘Change Price’
    f. ‘dp’ to ‘Percent Change’
    g. ‘t’ to ‘Timestamp’
    h. ‘ticker’ to ‘Ticker’
2. Reorder the columns so that the order is: ‘Ticker’, ‘Open Price’, ‘Close Price’, ‘Low Price’, ‘High Price’, ‘Change’, ‘Percent Change’, ‘Timestamp’

Historical Stock Data:

1. Convert the ‘Date’ column in the dataframe to ‘yyy-MM’dd’ format.

Historical Cryptocurrency Data:

1. Convert the ‘Date’ column in the dataframe to ‘yyy-MM’dd’ format.

US Economic Census Data:

1. Define the header as the first row with *header_row = dataframe.iloc[0]*
2. Define the dataframe to exclude the [0] row with *dataframe.values[1:]*
3. Define the dataframe to only include rows where the column [‘2017 NAICS CODE’] = ‘3112, 312140, 334111, 31-33, 44-45, 51, 52, 522110, 522210’
4. Define dataframe to only include columns[‘2017 NAICS code’], [‘Meaning of NAICS code’], [‘Number of establishments’], [‘Sales, value of shipments, or revenue of NAPCS collection code ($1,000)’]

Load:

Real-Time Stock Data:

1. Create a SQL database in the Azure cloud.
2. Connect to the database in Azure Data Studio on the SQL server within which it exists.
3. Create a table named ‘dbo.realstock’. Define columns for ‘Ticker’, ‘Open Price’, ‘Close Price’, ‘Low Price’, ‘High Price’, ‘Change’, ‘Percent Change’, ‘Timestamp’.
4. Define the database, table, username and password for the SQL database created in step 1.
5. In the consumer class created in step 6 of the Extract stage for Real-Time Stock Data
    a. Create a mount point to a datalake.
    b. Save the data as a .csv file to the datalake for backup purposes.

    c. Write the message data created in the Extract step for Real-Time Stock Data into the SQL database table.
6. Create a data factory in Azure Cloud Services.
7. Within the data factory, create a new pipeline.
8. Within the pipeline, create a new Databricks notebook object and assign the Producer class created in step 1 of the Extract stage for Real-Time Stock Data .
9. Create a trigger set for 1 minute connected to the producer notebook object in the pipeline.
10. Create a new Databricks notebook object and assign the Consumer class created in step 6 of the Extract stage for Real-Time Stock Data.
11. Running this data factory will see the calls to the API in the Producer class populate a dataframe which will then be sent as messages to the consumer on an interval. The consumer class will then gather the messages from the defined topic load them to the created SQL database.

Historical Stock and Cryptocurrency Data:

1. Define a data location for the historical stock data and crypto data within the datalake created in step 1 of the Load section for Real-Time Stock Data.
2. Save the historical stock data and historical crypto data as separate .csv files to the given data locations within the datalake.
3. Call *.repartition* on the historical stock data dataframe and define the *.save* option with the location chosen for the historical stock data .csv.
4. Call *.repartition* on the historical crypto data dataframe and define the *.save* option with the location chosen for the historical crypto data .csv.
5. Create a table in the SQL database from step 2 in the load section for Real-Time Stock Data named ‘dbo.hstock’ for the historical stock data. Define columns for ‘Ticker’, ‘Date’, ‘Open’, ‘High’, ‘Low’, ‘Close’, and ’Volume’.
6. Write the data from the historical stock dataframe to the dbo.hstock table in the created SQL database from step 2 in the load section for Real-Time Stock Data.
7. Create a table in the SQL database from step 2 in the load section for Real-Time Stock Data named ‘dbo.hcrypto’ for the historical crypto data. Define columns for ‘Currency, ‘Date’, ‘Open’, ‘High’, ‘Low’, ‘Close’, and ’Volume’.
8. Write the data from the historical crypto dataframe to the dbo. hcrypto’ table in the created SQL database from step 2 in the load section for Real-Time Stock Data.

US Economic Census Data:

1. Write the dataframe created for the US Economic Census Data to a .csv file in the storage container in the datalake created in step 1 of the Extract stage for US Economic Census Data by calling *.repartition().*