EECS E6893 Big Data Analytics - Fall 2023

Homework Assignment 2: Steaming Big Data Analytics & Data Analytics Pipeline

Due Friday, October 20th, 2023, by 5:00pm

Streaming Analytics

Abstract: In this section, we will demonstrate how to handle live streaming data, where information is received in real-time. Here, analytical operations need to be performed dynamically as the data arrives, without having a central repository of all the data to refer to. While the ideal choice would have been Twitter Streaming, due to recent organizational changes and the complexity of accessing the data, we will instead use the Finnhub API. By making frequent requests to this API for stock price data, we will simulate streaming analysis and showcase the process of handling real-time data.

Setup:
Step 1: Create your account using your Columbia email ID on Finnhub
Step 2: Login and Save your API Key (You will use your own unique API key in the following exercise)
Step 3: Create your GCP Dataproc Cluster as done for previous assignments.
Step 4: In your Dataproc cluster's Jupyter notebook run the following to install the Finnhub api

```
In [33]: !pip install finnhub-python
```

Skeleton Code:
To pull data from Finnhub for a particular stock – the syntax is as follows:

```
import finnhub
finnhub_client = finnhub.Client(api_key="YOUR UNIQUE API KEY")
print(finnhub_client.stock_candles('STOCK NAME', 'DATA RESOLUTION', start_unix_timestamp, end_unix_timestamp))
```

This code needs to be modified as per the requirements of the question. A detailed explanation of the parameters can be found at this link under the "Stock Candles" section.

Tasks (20 points)
Read all 3 parts collectively to decide the approach to be taken.

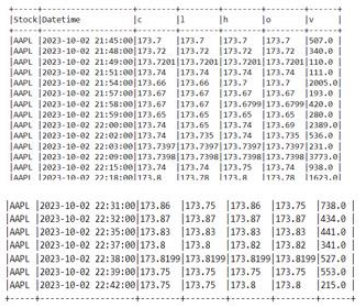
(1) Pull the stock data of "AAPL" at 1-minute level data resolution every 5 minutes. Each such pull made once every 5 minutes should have data from the (current timestamp - one hour) to the current timestamp when you hit the api i.e. one hour's worth of data in each pull. Let this process run for 30 minutes. At the end of your program, your code should have generated ~1.5 hours' worth of data for the stock and the api should have been called 7 times (once every 5 minutes, over a period of 30 minutes).

(2) In each data pull, the data should be incrementally loaded into a Spark data frame where the dataframe schema will be ['Stock Name', "UTC Timestamp", "c", "l", "h", "o", "v"] where UTC timestamp will be obtained by converting the UNIX timestamps of each entry the api call will return under the "t" key.

Incremental loading entails fetching data in stages. Initially, data covering an hour, say from timestamp x to x+60 minutes, is retrieved and stored in a dataframe. In the subsequent pull, fetch data from x+5 minutes to x+65 minutes. The data from x+5 to x+60 minutes, already present in the dataframe, gets replaced by the new pull (though it might remain the same in our exercise, real-world scenarios could witness changes in the same data points pulled at different intervals. HINT: To do this filtering operation explore the concept of antijoins). The data points from x+60 to x+65 minutes are directly added. This process is repeated for all pulls within the next 30 minutes, resulting in a data frame containing data from x to x+1.5 hours. (It may exactly not be 1.5hours since the api sometimes may not return the exact timestamp till which you request due to lags and a few datapoints maybe missing here and there which is fine)

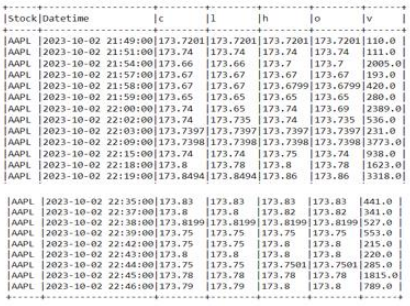
Sample flow of incremental update operation:

Data Currently in dataframe



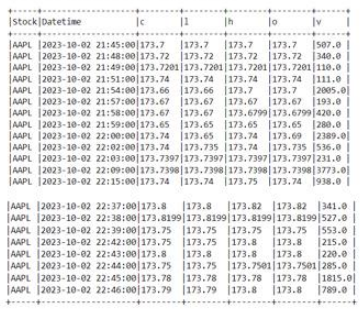
Stock	Datetime	c	l	h	o	v
AAPL	2023-10-02 21:45:00	173.7	173.7	173.7	173.7	507.0
AAPL	2023-10-02 21:48:00	173.72	173.72	173.72	173.72	340.0
AAPL	2023-10-02 21:49:00	173.7201	173.7201	173.7201	173.7201	110.0
AAPL	2023-10-02 21:51:00	173.74	173.74	173.74	173.74	111.0
AAPL	2023-10-02 21:54:00	173.66	173.66	173.7	173.7	2005.0
AAPL	2023-10-02 21:57:00	173.67	173.67	173.67	173.67	193.0
AAPL	2023-10-02 21:58:00	173.67	173.67	173.6799	173.6799	420.0
AAPL	2023-10-02 21:59:00	173.65	173.65	173.65	173.65	280.0
AAPL	2023-10-02 22:00:00	173.74	173.65	173.74	173.69	2389.0
AAPL	2023-10-02 22:02:00	173.74	173.735	173.74	173.735	536.0
AAPL	2023-10-02 22:03:00	173.7397	173.7397	173.7397	173.7397	231.0
AAPL	2023-10-02 22:09:00	173.7398	173.7398	173.7398	173.7398	3773.0
AAPL	2023-10-02 22:15:00	173.74	173.74	173.75	173.74	938.0
AAPL	2023-10-02 22:18:00	173.8	173.78	173.8	173.78	1623.0
AAPL	2023-10-02 22:31:00	173.86	173.75	173.86	173.75	738.0
AAPL	2023-10-02 22:32:00	173.87	173.87	173.87	173.87	434.0
AAPL	2023-10-02 22:35:00	173.83	173.83	173.83	173.83	441.0
AAPL	2023-10-02 22:37:00	173.8	173.8	173.82	173.82	341.0
AAPL	2023-10-02 22:38:00	173.8199	173.8199	173.8199	173.8199	527.0
AAPL	2023-10-02 22:39:00	173.75	173.75	173.75	173.75	553.0
AAPL	2023-10-02 22:42:00	173.75	173.75	173.8	173.8	215.0

Data from latest api pull



Stock	Datetime	c	l	h	o	v
AAPL	2023-10-02 21:49:00	173.7201	173.7201	173.7201	173.7201	110.0
AAPL	2023-10-02 21:51:00	173.74	173.74	173.74	173.74	111.0
AAPL	2023-10-02 21:54:00	173.66	173.66	173.7	173.7	2005.0
AAPL	2023-10-02 21:57:00	173.67	173.67	173.67	173.67	193.0
AAPL	2023-10-02 21:58:00	173.67	173.67	173.6799	173.6799	420.0
AAPL	2023-10-02 21:59:00	173.65	173.65	173.65	173.65	280.0
AAPL	2023-10-02 22:00:00	173.74	173.65	173.74	173.69	2389.0
AAPL	2023-10-02 22:02:00	173.74	173.735	173.74	173.735	536.0
AAPL	2023-10-02 22:03:00	173.7397	173.7397	173.7397	173.7397	231.0
AAPL	2023-10-02 22:09:00	173.7398	173.7398	173.7398	173.7398	3773.0
AAPL	2023-10-02 22:15:00	173.74	173.74	173.75	173.74	938.0
AAPL	2023-10-02 22:18:00	173.8	173.78	173.8	173.78	1623.0
AAPL	2023-10-02 22:19:00	173.8494	173.8494	173.86	173.86	3318.0
AAPL	2023-10-02 22:35:00	173.83	173.83	173.83	173.83	441.0
AAPL	2023-10-02 22:37:00	173.8	173.8	173.82	173.82	341.0
AAPL	2023-10-02 22:38:00	173.8199	173.8199	173.8199	173.8199	527.0
AAPL	2023-10-02 22:39:00	173.75	173.75	173.75	173.75	553.0
AAPL	2023-10-02 22:42:00	173.75	173.75	173.8	173.8	215.0
AAPL	2023-10-02 22:43:00	173.8	173.8	173.8	173.8	220.0
AAPL	2023-10-02 22:44:00	173.75	173.75	173.7501	173.7501	285.0
AAPL	2023-10-02 22:45:00	173.78	173.78	173.78	173.78	1815.0
AAPL	2023-10-02 22:46:00	173.79	173.79	173.8	173.8	789.0

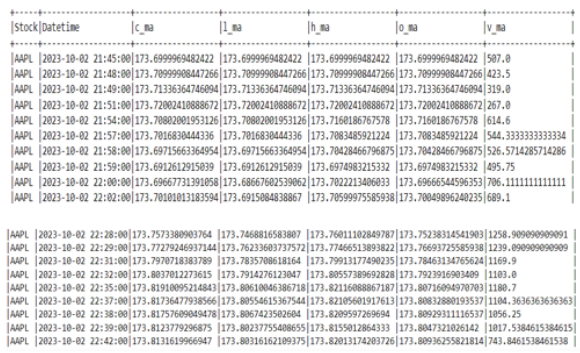
Final dataframe state after incremental update



Stock	Datetime	c	l	h	o	v
AAPL	2023-10-02 21:45:00	173.7	173.7	173.7	173.7	507.0
AAPL	2023-10-02 21:48:00	173.72	173.72	173.72	173.72	340.0
AAPL	2023-10-02 21:49:00	173.7201	173.7201	173.7201	173.7201	110.0
AAPL	2023-10-02 21:51:00	173.74	173.74	173.74	173.74	111.0
AAPL	2023-10-02 21:54:00	173.66	173.66	173.7	173.7	2005.0
AAPL	2023-10-02 21:57:00	173.67	173.67	173.67	173.67	193.0
AAPL	2023-10-02 21:58:00	173.67	173.67	173.6799	173.6799	420.0
AAPL	2023-10-02 21:59:00	173.65	173.65	173.65	173.65	280.0
AAPL	2023-10-02 22:00:00	173.74	173.65	173.74	173.69	2389.0
AAPL	2023-10-02 22:02:00	173.74	173.735	173.74	173.735	536.0
AAPL	2023-10-02 22:03:00	173.7397	173.7397	173.7397	173.7397	231.0
AAPL	2023-10-02 22:09:00	173.7398	173.7398	173.7398	173.7398	3773.0
AAPL	2023-10-02 22:15:00	173.74	173.74	173.75	173.74	938.0
AAPL	2023-10-02 22:37:00	173.8	173.8	173.82	173.82	341.0
AAPL	2023-10-02 22:38:00	173.8199	173.8199	173.8199	173.8199	527.0
AAPL	2023-10-02 22:39:00	173.75	173.75	173.75	173.75	553.0
AAPL	2023-10-02 22:42:00	173.75	173.75	173.8	173.8	215.0
AAPL	2023-10-02 22:43:00	173.8	173.8	173.8	173.8	220.0
AAPL	2023-10-02 22:44:00	173.75	173.75	173.7501	173.7501	285.0
AAPL	2023-10-02 22:45:00	173.78	173.78	173.78	173.78	1815.0
AAPL	2023-10-02 22:46:00	173.79	173.79	173.8	173.8	789.0

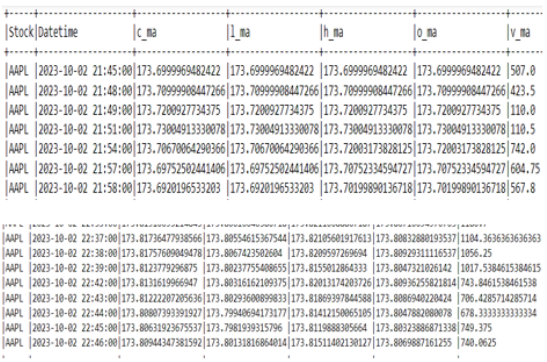
(3) Every 5 minutes, after inserting data into the dataframe created in part (2), compute the 30-minute moving averages for each stock's "c", "l", "h", "o", and "v" values. Store these moving averages incrementally in a separate PySpark dataframe with the schema ["Datetime", "c_MA", "l_MA", "h_MA", "o_MA", "v_MA"], where "Datetime" represents the end timestamp of the moving average window.

Moving Averages for the data called till 2023-10-02 22:42:00



Stock	Datetime	c_ma	l_ma	h_ma	o_ma	v_ma
AAPL	2023-10-02 21:45:00	173.6999969482422	173.6999969482422	173.6999969482422	173.6999969482422	507.0
AAPL	2023-10-02 21:48:00	173.7099990844726	173.7099990844726	173.7099990844726	173.7099990844726	423.5
AAPL	2023-10-02 21:49:00	173.71336364746094	173.71336364746094	173.71336364746094	173.71336364746094	319.0
AAPL	2023-10-02 21:51:00	173.7200241088672	173.7200241088672	173.7200241088672	173.7200241088672	267.0
AAPL	2023-10-02 21:54:00	173.7080200195312	173.7080200195312	173.7160186767578	173.7160186767578	614.6
AAPL	2023-10-02 21:57:00	173.7016830444336	173.7016830444336	173.7083485921224	173.7083485921224	544.3333333333334
AAPL	2023-10-02 21:58:00	173.69715663364954	173.69715663364954	173.70428466796875	173.70428466796875	526.5714285714286
AAPL	2023-10-02 21:59:00	173.6912612915039	173.6912612915039	173.6974983215332	173.6974983215332	495.75
AAPL	2023-10-02 22:00:00	173.6966773139058	173.68667602539062	173.7022213486033	173.6966654596353	706.1111111111111
AAPL	2023-10-02 22:02:00	173.70101013183594	173.69150848388867	173.70599975585938	173.70049896240235	689.1
AAPL	2023-10-02 22:28:00	173.7573380903764	173.7468816583807	173.76011102849787	173.7523831454903	1258.909090909091
AAPL	2023-10-02 22:29:00	173.7727924693714	173.7623360373757	173.7746651389382	173.7669372558593	1239.090909090909
AAPL	2023-10-02 22:31:00	173.7970718383789	173.7835708618164	173.79913177490235	173.7846313476562	1169.9
AAPL	2023-10-02 22:32:00	173.8037012273615	173.7914276123047	173.8055738969282	173.7923916903409	1103.0
AAPL	2023-10-02 22:35:00	173.8191009521484	173.8061004638671	173.8211608886718	173.8071609497070	1180.7
AAPL	2023-10-02 22:37:00	173.8173647793856	173.8055461536754	173.8210560191761	173.8083288019353	1104.3636363636363
AAPL	2023-10-02 22:38:00	173.8175760904947	173.8067423502604	173.8209597269694	173.8092931116537	1056.25
AAPL	2023-10-02 22:39:00	173.8123779296875	173.8023775548065	173.8155012864333	173.8047321026142	1017.5384615384615
AAPL	2023-10-02 22:42:00	173.8131619966947	173.8031616210937	173.8201317420372	173.8093625582181	743.8461538461538

Moving Averages for all the data present after next api call at 2023-10-02 22:46:00



Stock	Datetime	c_ma	l_ma	h_ma	o_ma	v_ma
AAPL	2023-10-02 21:45:00	173.6999969482422	173.6999969482422	173.6999969482422	173.6999969482422	507.0
AAPL	2023-10-02 21:48:00	173.7099990844726	173.7099990844726	173.7099990844726	173.7099990844726	423.5
AAPL	2023-10-02 21:49:00	173.7200927734375	173.7200927734375	173.7200927734375	173.7200927734375	110.0
AAPL	2023-10-02 21:51:00	173.7300491330078	173.7300491330078	173.7300491330078	173.7300491330078	110.5
AAPL	2023-10-02 21:54:00	173.7067006429366	173.7067006429366	173.7200317382812	173.7200317382812	742.0
AAPL	2023-10-02 21:57:00	173.6975250244140	173.6975250244140	173.7075233459472	173.7075233459472	604.75
AAPL	2023-10-02 21:58:00	173.6920196533203	173.6920196533203	173.7019898013671	173.7019898013671	567.8
AAPL	2023-10-02 22:37:00	173.8173647793856	173.8055461536754	173.8210560191761	173.8083288019353	1104.3636363636363
AAPL	2023-10-02 22:38:00	173.8175760904947	173.8067423502604	173.8209597269694	173.8092931116537	1056.25
AAPL	2023-10-02 22:39:00	173.8123779296875	173.8023775548065	173.8155012864333	173.8047321026142	1017.5384615384615
AAPL	2023-10-02 22:42:00	173.8131619966947	173.8031616210937	173.8201317420372	173.8093625582181	743.8461538461538
AAPL	2023-10-02 22:43:00	173.8122207205636	173.8029360089833	173.8186939784588	173.8086940220424	706.4285714285714
AAPL	2023-10-02 22:44:00	173.8080739339192	173.7994069417717	173.8141250065105	173.8047882080078	678.3333333333334
AAPL	2023-10-02 22:45:00	173.8063192367553	173.7981939315796	173.8119888385664	173.8032386871338	749.375
AAPL	2023-10-02 22:46:00	173.8094347381592	173.8013181864014	173.8151140213127	173.8069887161255	740.0625

IMPORTANT NOTE: The data should be pulled anytime between 12pm – 4pm between Monday – Friday since that is the only time you will get real time stock data from this API, when the markets are operational. Plan your assignment timeline accordingly. You will most probably not be able to pull data outside these timings accurately and on weekends.

Airflow Data Pipelining:

Task 1 Helloworld (35 pts)

Q1.1 Read through the tutorial slides and install Airflow either on your local laptop or on a VM of GCP. You can also use google cloud composer if you know how to use that. (20 pts)

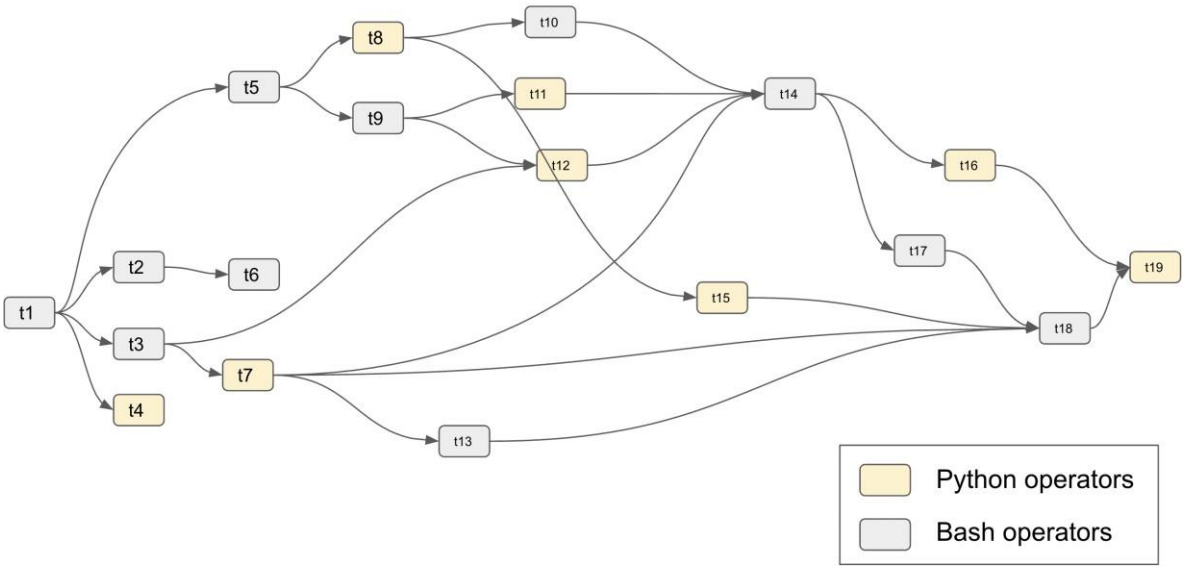
(1) Provide screenshots of terminals after you successfully start the webserver and scheduler.
(2) Provide screenshots of the web browser after you successfully login and see the DAGs.

Q1.2 Run helloworld with SequentialExecutor (15 pts)

(1) Provide screenshots of Tree, Graph, and Gantt charts after execution. (10 pts)
(2) Explore other features and visualizations you can find in the Airflow UI. Choose two features/visualizations (other than tree, graph, and Gantt), explain their functions and how they help monitor and troubleshoot the pipeline, use helloword as an example. (5 pts)

Task 2 Build workflows (45 pts)

Q2.1 Implement the DAG below (20 pts)



For each kind of operator, use at least 3 different commands. For example, you can choose sleep, print, count functions for Python operators, and echo, run bash script, run python file for Bash operators.

(1) Provide screenshots of Tree and Graph in airflow. (5 pts)
(2) Manually trigger the DAG and provide screenshots of Gantt. (10 pts)
(3) Schedule the first run immediately and then schedule running the program every 30 minutes. Describe how you decide the start date and schedule interval. Provide screenshots of running history after two repeats (first run + 2 repeats). On your browser, you can find the running history. (5 pts)

Q2.2 Stock price fetching, prediction, and storage every day (25 pts)

(1) Schedule fetching the stock price of [AAPL, GOOGL, META, MSFT, AMZN]. Use Yahoo! Finance data downloader https://pypi.org/project/yfinance/.

(2) Preprocess data if you think necessary.

(3) Train/update 5 linear regression models for stock price prediction for these 5 corporates. Each linear model takes the "open price", "high price", "low price", "close price", "volume" of the corporate in the current day as the feature and predicts the "high price" for the next day.

(4) Calculate the relative errors for the last 5 days, i.e., (prediction made from yesterday's data for today - actual price today) / actual price today, and update the prediction date and 5 errors into a table, e.g., a csv file.

(5) Provide screenshots of your code, the resultant errors csv, and the Airflow DAG. Describe with screenshots briefly how to build this workflow, e.g., what the DAG is with the various tasks, how you manage the cross tasks communication, how you setup the airflow scheduler…

Pointers for Q2.2: (You have to think how to create the different tasks and the overall Airflow execution DAG based on the question requirements, below are pointers that may give you some ideas)

- Pull historical data for each corporate till the current date and store data for each in a csv. (Set period in history to "max" yf.Ticker(company_tag).history(period='max'))

- Use this csv to incrementally train the linear regression models for each corporate's data. The dataset for each model X and y should be created such that each X(d) = ["open price", "high price", "low price", "close price", "volume"] of **date d** and the corresponding y = "High price" for **date d+1** as stated in the question.

- Assume current date (the date till which you have pulled the data) is **d_current.**

  For each corporate,

  - Train/Update a model using data from [X,y] till (d_current – i days) and use this model to predict the y for the (d_current – (i-1) day). Repeat the training and prediction for i = 5,4,3,2,1 (make predictions for last 5 days and calculate relative errors for each prediction)
  - For each value of i keep storing the relative errors in a csv for each model for each corporate on each of the 5 testing days.
  - The final errors csv should look something like below:
    If the latest date till which you pulled data was 11/30/2022 then your final errors csv should look like:

	A	B	C	D	E	F	G
1		Date	APPLE	GOOGLE	META	MICROSOI	AMAZON
2	0	11/26/2022	-0.00501	-0.00083	-0.001	-0.00167	0.005777
3	1	11/27/2022	-0.00091	-0.00755	-0.00097	-0.00698	-0.00609
4	2	11/28/2022	0.026959	0.007073	0.005526	0.003579	0.008291
5	3	11/29/2022	0.019557	0.0088	0.008127	0.011455	-0.0204
6	4	11/30/2022	0.009074	0.010989	-0.00189	0.009092	0.014896

To give an example: the value in cell C2 represents the relative error produced by the linear regression model trained on Apple data till 11/25/2022 and the prediction made for 11/26/2022. Similarly, the value in cell C3 represents the relative error produced by the updated linear regression model trained on Apple data till 11/26/2022 and the prediction made for 11/27/2022. The value in cell F6 represents the relative error produced by the linear regression model trained on Microsoft data till 11/29/2022 and the prediction made for 11/30/2022.

Homework Submissions

Streaming Analytics:

- PDF with screenshots of your code, brief explanation of the code worflow and the results i.e. dataframe holding the streamed data and the dataframe holding the moving averages.
- Code file for Streaming Analytics section

Airflow:

- Provide your screenshots, answers and code as mentioned in the individual questions.