



EECS E6893 Big Data Analytics

HW2: Streaming Big Data Analytics & Data Analytics Pipeline

TUTORIAL 1

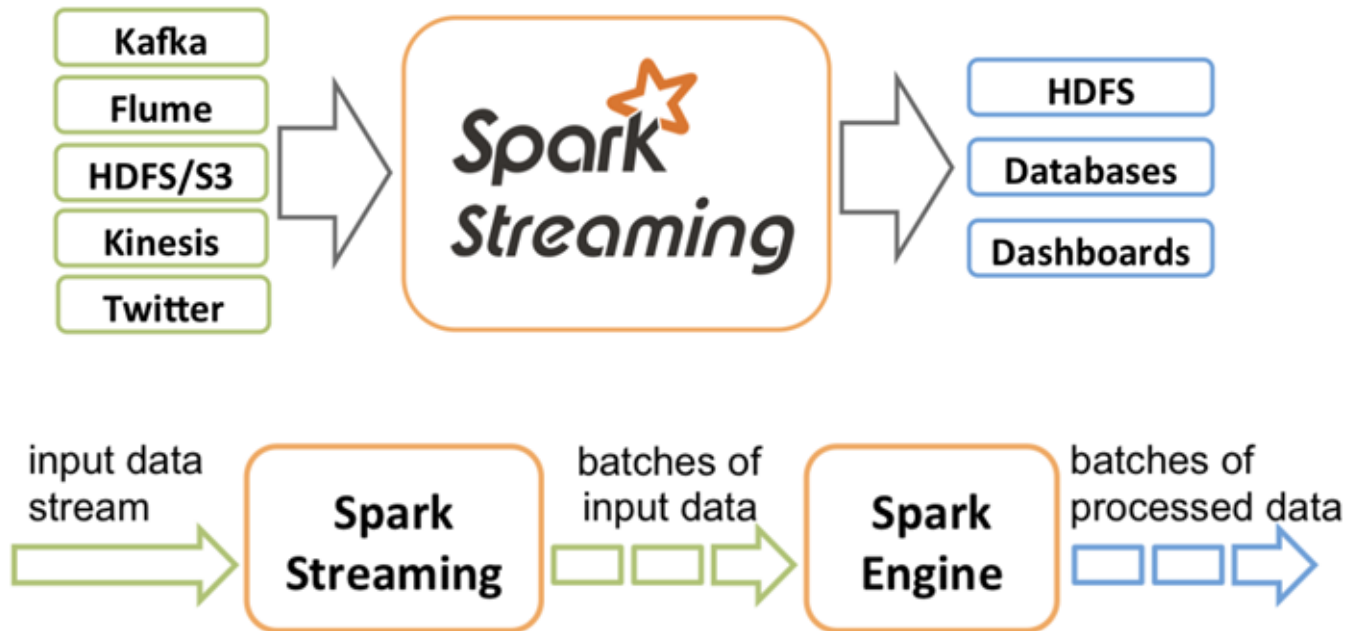
September 25, 2026

Agenda

- Streaming Analytics on Massive(Previously Polygon) stock data

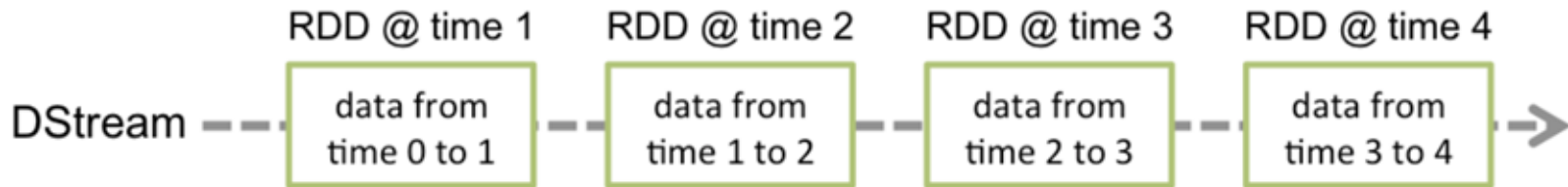
STREAMING ANALYTICS

Spark Streaming



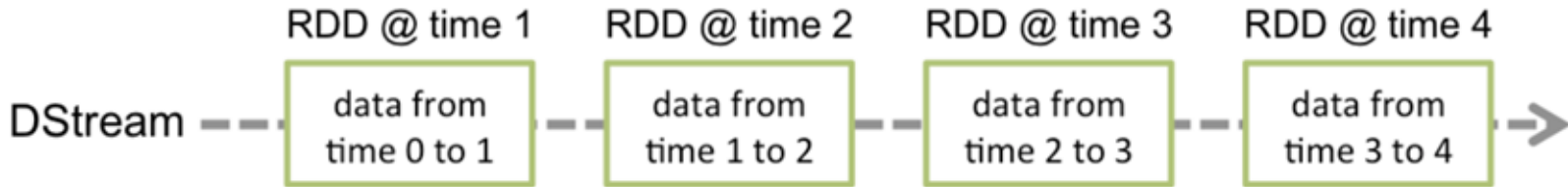
<https://spark.apache.org/docs/latest/streaming-programming-guide.html>

DStream



- A basic abstraction provided by Spark Streaming
- Represents a continuous stream of data
- Contains a continuous series of RDDs at different time

Dstream Example



First, we import `StreamingContext`, which is the main entry point for all streaming functionality. We create a local `StreamingContext` with two execution threads, and batch interval of 1 second.

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
```

```
# Create a local StreamingContext with two working thread and batch interval of 1 second
sc = SparkContext("local[2]", "NetworkWordCount")
ssc = StreamingContext(sc, 1)
```

Using this context, we can create a DStream that represents streaming data from a TCP source, specified as hostname (e.g. localhost) and port (e.g. 9999).

```
# Create a DStream that will connect to hostname:port, like localhost:9999
lines = ssc.socketTextStream("localhost", 9999)
```

This `lines` DStream represents the stream of data that will be received from the data server. Each record in this DStream is a line of text. Next, we want to split the lines by space into words.

```
# Split each line into words
words = lines.flatMap(lambda line: line.split(" "))
```

This `lines` DStream represents the stream of data that will be received from the data server. Each record in this DStream is a line of text. Next, we want to split the lines by space into words.

```
# Split each line into words
words = lines.flatMap(lambda line: line.split(" "))
```

`flatMap` is a one-to-many DStream operation that creates a new DStream by generating multiple new records from each record in the source DStream. In this case, each line will be split into multiple words and the stream of words is represented as the `words` DStream. Next, we want to count these words.

```
# Count each word in each batch
pairs = words.map(lambda word: (word, 1))
wordCounts = pairs.reduceByKey(lambda x, y: x + y)

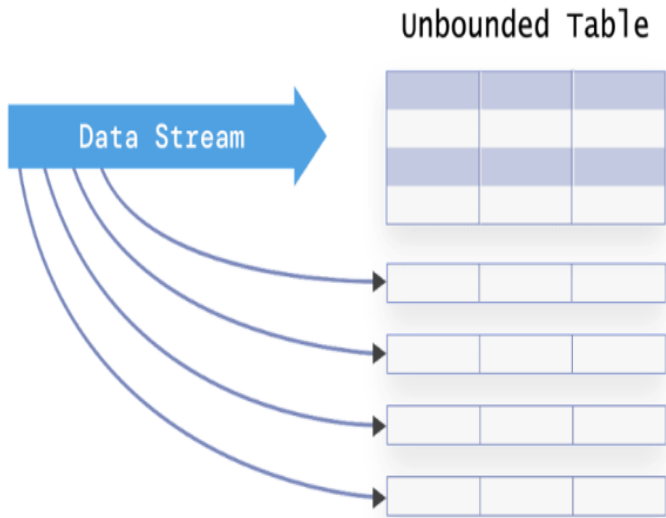
# Print the first ten elements of each RDD generated in this Dstream to the console
wordCounts.pprint()
```

The `words` DStream is further mapped (one-to-one transformation) to a DStream of `(word, 1)` pairs, which is then reduced to get the frequency of words in each batch of data. Finally, `wordCounts.pprint()` will print a few of the counts generated every second.

Note that when these lines are executed, Spark Streaming only sets up the computation it will perform when it is started, and no real processing has started yet. To start the processing after all the transformations have been setup, we finally call

```
ssc.start() # Start the computation
ssc.awaitTermination() # Wait for the computation to terminate
```

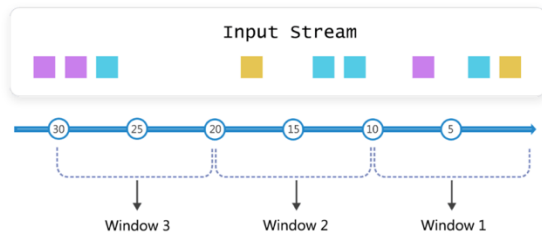
Spark Structured Streaming



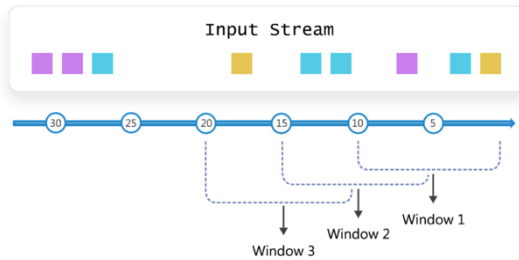
New data in the
data stream
=
New rows appended
to an unbounded table

Data stream as an unbounded table

Tumbling Window

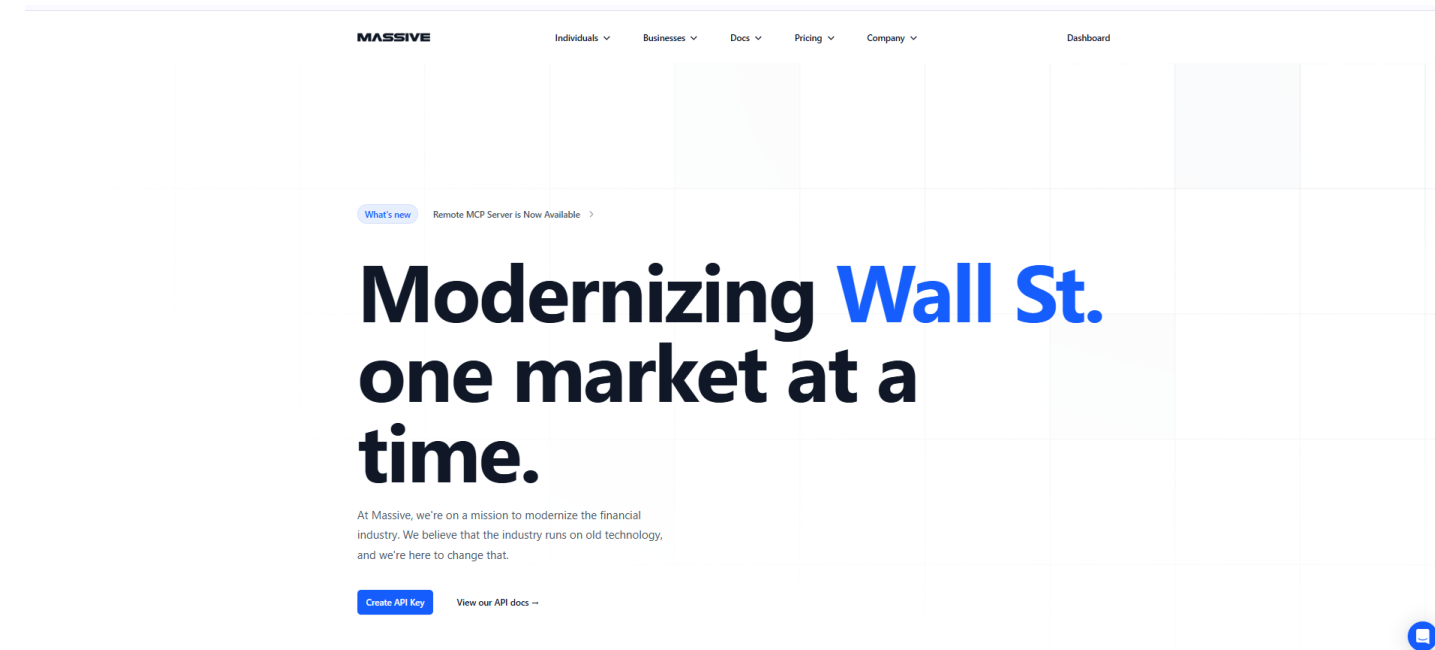


Sliding Window



Streaming Analytics Setup

- Step 1: Create your account using your Columbia email ID on [Massive](#)



Streaming Analytics Setup

- Step 2: Login and Save your API Key from Dashboard (You will use your own unique API key in the following exercise)

Products ▾

Docs ▾

Company ▾

Pricing

Contact



Dashboard

API Keys

Add API Key

Name	API Key
Default	Copy

Streaming Analytics Setup

- Step 3: Pull data from Massive

REST API

Getting Started

Market Data Endpoints ^

- Aggregates (Bars)
- Grouped Daily (Bars)
- Daily Open/Close
- Previous Close
- Trades
- Last Trade
- Quotes (NBBO)
- Last Quote
- Snapshots ▾
- Technical Indicators ▾

Reference Data Endpoints ^

- Tickers
- Ticker Details V3
- Ticker Events
- Ticker News
- Ticker Types
- Market Holidays
- Market Status
- Stock Splits V3
- Dividends V3
- Stock Financials VX
- Conditions
- Exchanges

Aggregates (Bars)

GET /v2/aggs/ticker/{stocksTicker}/range/{multiplier}/{timespan}/{from}/{to}

Get aggregate bars for a stock over a given date range in custom time window sizes.

For example, if timespan = 'minute' and multiplier = '5' then 5-minute bars will be returned.

Parameters

stocksTicker* AAPL

Specify a case-sensitive ticker symbol. For example, AAPL represents Apple Inc.

multiplier* 1

The size of the timespan multiplier.

timespan* day

The size of the time window.

from* 2023-01-09

The start of the aggregate time window. Either a date with the format YYYY-MM-DD or a millisecond timestamp.

to* 2023-02-10

The end of the aggregate time window. Either a date with the format YYYY-MM-DD or a millisecond timestamp.

adjusted true

Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to false to get results that are NOT adjusted for splits.

sort asc

Sort the results by timestamp. asc will return results in ascending order (oldest at the top), desc will return results in descending order (newest at the top).

limit

Response Object

```
{
  "adjusted": true,
  "next_url": "https://api.polygon.io",
  "queryCount": 2,
  "request_id": "6a7e466379af0a71039d",
  "results": [
    {
      "c": 75.0875,
      "h": 75.15,
      "l": 73.7975,
      "n": 1,
      "o": 74.86,
      "t": 1577941200000,
      "v": 135647456,
      "vw": 74.6999
    },
    {
      "c": 74.3575,
      "h": 75.145,
      "l": 74.125,
      "n": 1,
      "o": 74.2875,
      "t": 1578027600000,
      "v": 146535512,
      "vw": 74.7026
    }
  ],
  "resultsCount": 2,
  "status": "OK",
  "ticker": "AAPL"
}
```

Streaming Analytics Setup

- Step 4: Create your GCP Dataproc Cluster as done for previous assignments and setup the Massive API

```
import requests
import pandas as pd
from datetime import datetime

# Both api.massive.com and api.polygon.io are valid
BASE_URL = 'https://api.massive.com'
API_KEY = 'YOUR_API_KEY_HERE'

def get_histdata_massive(ticker, start_date, end_date, timespan='minute', multiplier=1):
    """
    Fetches aggregate stock bars from Massive / Polygon API.
    Dates/timestamps can be provided as YYYY-MM-DD or millisecond timestamps.
    """
    url = f"{BASE_URL}/v2/aggs/ticker/{ticker}/range/{multiplier}/{timespan}/{start_date}/{end_date}?apiKey={API_KEY}"
    response = requests.get(url)

    if response.status_code == 200:
        data = response.json()
        if 'results' not in data or not data['results']:
            print("No results returned for the specified window.")
            return pd.DataFrame()

        df = pd.DataFrame(data['results'])
        # Convert millisecond epoch 't' to human-readable datetime
        df['timestamp'] = pd.to_datetime(df['t'], unit='ms', utc=True)

        # Select and rename relevant columns
        df = df[['timestamp', 'o', 'h', 'l', 'c', 'v']]
        df.columns = ['UTC Timestamp', 'o', 'h', 'l', 'c', 'v']
        return df
    elif response.status_code == 403:
        print("Error 403: Forbidden. Verify your API key or ensure you are querying valid historical dates allowed on your tier.")
        return pd.DataFrame()
    else:
        print(f"Error {response.status_code}: {response.text}")
        return pd.DataFrame()

# Example test run for a past market day
test_df = get_histdata_massive('AAPL', '2024-10-03', '2024-10-03')
print(test_df.head())
```

TO DO

1. Pull the stock data of “AAPL” at **1-minute level 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 entire 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).

TO DO

2. In each data pull, the data should be **incrementally** loaded into a Spark data frame where the data frame schema will be [“Stock Name”, “UTC Timestamp”, “c”, “l”, ”h”, ”o”, ”v”]

Sample flow of incremental update operation:

Data Currently in dataframe

Data from latest api pull

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	173.74	173.69	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	22:40:00	173.75	173.75	173.75	173.75	215.0
AAPL	2023-10-02	22:41:00	173.75	173.75	173.75	173.75	215.0
AAPL	2023-10-02	22:42:00	173.75	173.75	173.8	173.8	215.0

	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

	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

TO DO

3. Every 5 minutes, after inserting data into the dataframe, compute the **30-minute moving averages** for the 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	173.70999908447266	173.70999908447266	173.70999908447266	173.70999908447266	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	173.72002410888672	173.72002410888672	173.72002410888672	173.72002410888672	267.0
AAPL	2023-10-02 21:54:00	173.70802001953126	173.70802001953126	173.7160186767578	173.7160186767578	614.6
AAPL	2023-10-02 21:57:00	173.7016830444336	173.7016830444336	173.7083485921224	173.7083485921224	544.333333333334
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	173.69667731391058	173.68667602539062	173.7022213406033	173.69666544596353	706.1111111111111
AAPL	2023-10-02 22:02:00	173.70101013183594	173.6915084838867	173.70599975585938	173.70049896240235	689.1
AAPL	2023-10-02 22:28:00	173.7573380903764	173.74668816583807	173.76011102849787	173.75238314541903	1258.9090909090909
AAPL	2023-10-02 22:29:00	173.77279246937144	173.76233603737572	173.77466513893822	173.76693725585938	1239.090909090909
AAPL	2023-10-02 22:31:00	173.7970718383789	173.7835708618164	173.79913177490235	173.78463134765624	1169.9
AAPL	2023-10-02 22:32:00	173.8037012273615	173.7914276123047	173.80557389692828	173.7923916903409	1103.0
AAPL	2023-10-02 22:35:00	173.81910095214843	173.80610046386718	173.82116088867187	173.80716094970703	1180.7
AAPL	2023-10-02 22:37:00	173.81736477938566	173.80554615367544	173.82105601917613	173.80832880193537	1104.3636363636363
AAPL	2023-10-02 22:38:00	173.81757609049478	173.8067423502604	173.8209597269694	173.80929311116537	1056.25
AAPL	2023-10-02 22:39:00	173.8123775296875	173.8023775408655	173.8155012864333	173.8047321026142	1017.5384615384615
AAPL	2023-10-02 22:42:00	173.8131619966947	173.80316162109375	173.82013174203726	173.80936255821814	743.8461538461538
AAPL	2023-10-02 22:44:00	173.80807393391927	173.79940694173177	173.81412150065105	173.8047882080078	678.3333333333334
AAPL	2023-10-02 22:45:00	173.80631923675537	173.7981939315796	173.8119888305664	173.80323886871338	749.375
AAPL	2023-10-02 22:46:00	173.80944347381592	173.80131816864014	173.81511402130127	173.8069887161255	740.0625

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	173.70999908447266	173.70999908447266	173.70999908447266	173.70999908447266	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	173.73004913330078	173.73004913330078	173.73004913330078	173.73004913330078	110.5
AAPL	2023-10-02 21:54:00	173.70670064290366	173.70670064290366	173.72003173828125	173.72003173828125	742.0
AAPL	2023-10-02 21:57:00	173.69752502441406	173.69752502441406	173.70752334594727	173.70752334594727	604.75
AAPL	2023-10-02 21:58:00	173.6920196533203	173.6920196533203	173.7019890136718	173.7019890136718	567.8
AAPL	2023-10-02 22:37:00	173.81736477938566	173.80554615367544	173.82105601917613	173.80832880193537	1104.3636363636363
AAPL	2023-10-02 22:38:00	173.81757609049478	173.8067423502604	173.8209597269694	173.80929311116537	1056.25
AAPL	2023-10-02 22:39:00	173.8123775296875	173.80237755408655	173.8155012864333	173.8047321026142	1017.5384615384615
AAPL	2023-10-02 22:42:00	173.8131619966947	173.80316162109375	173.82013174203726	173.80936255821814	743.8461538461538
AAPL	2023-10-02 22:43:00	173.8122207205636	173.80293600898933	173.81869397844588	173.8086940220424	706.4285714285714
AAPL	2023-10-02 22:44:00	173.80807393391927	173.79940694173177	173.81412150065105	173.8047882080078	678.3333333333334
AAPL	2023-10-02 22:45:00	173.80631923675537	173.7981939315796	173.8119888305664	173.80323886871338	749.375
AAPL	2023-10-02 22:46:00	173.80944347381592	173.80131816864014	173.81511402130127	173.8069887161255	740.0625

To Submit:

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

Important Notes:

- You may not get the data for each minute level timestep, there may be a few data points missing here and there which is fine but ensure that the majority of the expected timestamps are present in each pull.