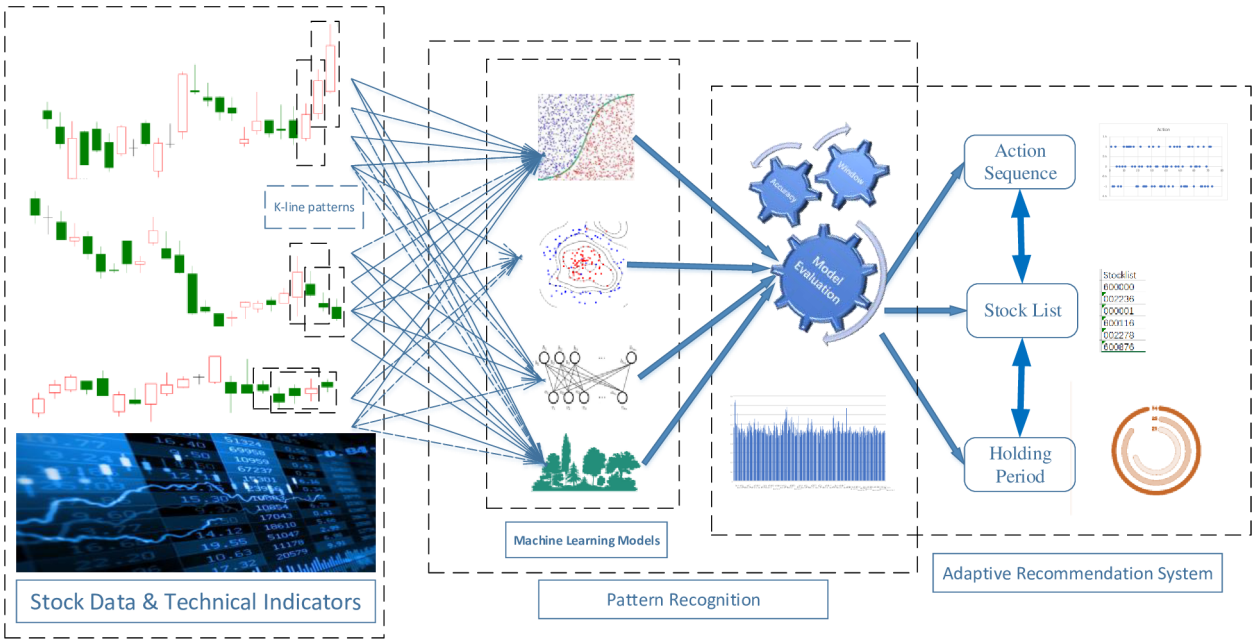
Quiz 5: Data Streams [280 pts]



Technical Analysis of Stock Trading[1]

Assigned on Oct 26 2022 and due on Nov 15 2022 11:59 PM. The scores will be awarded out of a max of 280 and eventually divided by 20. This quiz will account for 14% of your final grade.

This quiz has three problems based on code in my github repo. To begin,

```
git clone https://github.com/singhj/big-data-repo
```

Please submit pull requests for any improvements. The code in this repo generates various types of data streams.

[1] Technical analysis of stock movements is a mysterious field. One wonders why someone who publishes papers on it would do so. We suspect that much research in this field is never published. This paper, the source for the diagram, is but one example.

Moving Averages [150 pts]

Calculation of moving stock price averages are part of many trading strategies (reference).

We will be using the two moving averages strategy, with the shorter-term MA being 10-day and the longer average being 40-day. When the shorter-term MA crosses above the longer-term MA, it's a buy signal, as it indicates that the trend is shifting up. This is known as a **golden cross.** Meanwhile, when the shorter-term MA crosses below the longer-term MA, it's a sell signal, as it indicates that the trend is shifting down. This is known as a **dead/death cross.** Both types of crosses are shown in the figure below:

To simulate a data stream, you are given a python program `aapl-feeder.py` which produces stock prices for Apple (Symbol: aapl) and pipes it, line by line. We will only be concerned with the Close price. The command `aapl-feeder.py | nc -lk 9999` can be run on the master machine of your spark cluster to feed the Close data into pyspark.

1. [30 pts] Set up the stream to feed data into a pyspark DStream.
2. [30 pts] Use DStream windowing to accumulate two DStreams: aaplsum and aaplct, respectively the sum and count of prices.
3. [30 pts] Divide these two DStreams (**aaplsum and aaplct**) to produce aaplavg, thus creating moving average DStreams for both 10-day MA and 40-day MA .
4. [60 pts]. Compare the two moving averages (shorter-term MA and the longer MA) to indicate buy and sell signals. Your output[2] should be of the form `[( <date> buy <symbol>), ( <date> sell <symbol>), etc]`.

For each case, submit your spark code. For part 4, also submit the output.

[2] `aapl-feeder` uses data for only one symbol: AAPL, which is implied. Your output for this problem will be `[( <date> buy AAPL), ( <date> sell AAPL), etc]`.

Counting Unique Words [60 pts]

Most news outlets distribute their news through rss feeds for use by news reader programs. The file `news-feeder.py` reads such news headlines.

Write a hyperloglog algorithm.script that counts the number of ***unique*** words in the news headlines. The desired usage is:

```
news-feeder.py | unique_word_count.py
```

Note that

- I'd prefer it if you wrote your own implementation of hyperloglog. Please be scrupulous about citing any sources you use off the internet
- This question does not require Spark. A laptop capable of running Python is sufficient!
- `news-feeder.py` reads the news from various sources. As a result, it is not 100% repeatable. It is a function of when you happen to run it.
- The news sources have different policies on rate-limiting their news feed. If you overuse them, they have a right to throttle the data.
- The average time between sentences in `news-feeder.py` is set to 1 sec but you may change the `time.sleep(...)` line in the code or comment it out entirely!
- Your `unique_word_count.py` should be thoroughly tested [20 pts]. Count the unique words using some other method and see how far the result of using hyperloglog differs from the actual count.

Bloom Filter [70 pts]

We are interested in flagging headlines that have AFINN scores of -4 or -5.

Most news outlets distribute their news through rss feeds for use by news reader programs. The file `news-feeder.py` reads news headlines.

- [30 pts] Create a Bloom Filter[3] for detecting bad words (i.e., AFINN of -4 ot -5) in a headline. string whose size is approximately 1000 bits[4]. It should be designed to run in pyspark.
- [40 pts] Use DStream windowing to filter incoming headlines, using the Bloom filter created in the first part to emit only the headlines with flagged words in them.

- What to submit:
  a. Your pyspark (or scala) code and
  b. A recorded video session showing the streaming filter in action. The session should be no more than 120 seconds in length. [To do this, create a zoom meeting, set it to *record to the cloud*, plan what you are going to say, then start the meeting with just yourself, share the screen, and go through the demo. Soon after you end the meeting, zoom will send you a recording URL, which you may watch – and submit the URL]

[3] There are many implementations of Bloom filters on the internet. You can consult them if you wish but *the purpose of this exercise is to write your own.*

[4] How to store a bit-vector in a file? Since characters like \000 denote EOF (end of file), and \000 is a valid sequence of bits, we have to use base64-encoded files that can be safely accessed. Create the bit array in your program and write it as a base64-encoded file (call it `bloom.txt`) so it can be stored in HDFS and safely accessed from pySpark.