UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

Assignment 3
Analysis of stock pricing

SUBJECT NUMBER & NAME	94693 Big Data Engineering / Autumn 2021		
ASSESSMENT ITEM NUMBER/TITLE	Assignment 3		
DUE DATE	June 13 2021 11:59pm		
AUTHOR DETAILS	Donovan Tay	12964300	donovan.w.tay@student.uts.edu.au
	Surbhi	13453954	surbhi.tyagi@student.uts.edu.au
	Min Zhu	12456810	min.zhu-1@student.uts.edu.au
	Catherine Cheung	12808655	catherine.k.cheung@student.uts.edu.au
DELIVERABLES	Notebook: /notebookes/FF_III_final.ipynb + FF_III_final.html Report: /reports/EXPERIMENT REPORT - Assignment3_FinalFantasyIII_Final.pdf		

UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

Contents

Business Objectives

The business objective of this report is to create streaming data pipelines of stock trade data from Datagen, to feed the following outputs:

1. KSQL data streams and relevant tables of business KPIs
2. A tumbling 60 second window of stock transactions, updated every 10 seconds
3. Visualisations updated every 10 seconds with 3 charts showing the latest 'traded amount vs stock', 'traded quantity vs share price' and the 'latest 10 sell-price values along time' of a particular stock.
4. Machine learning models that provide predictions on whether a stock trade side is "BUY" or "SELL" (Classification) and the quantity of stock bought or sold (Regression).

Business Understanding

Our client is interested in tracking stock trades made by users, in order to predict what reaction the users have when the price of stocks varies. As a proof of concept, our team is building the end-to-end data pipeline for the stock data feed all the way to the machine learning model to predict the quantity of shares to buy or sell.

As a prototype system, the stock price data is currently generated randomly using the Datagen data generator in Kafka. This means that our predictive model is not expected to generate a meaningful output and the focus of this project is the data flows to the predictive model, as well as KPI queries from KSQL and the window stream, plus plot. Real world data mapping from the client's stock trading platform will be mapped and piped through in Phase 2 of the project.

Data Understanding

The schema for the data generated from the stock_trade resource from the Datagen module in Kafka is documented at
https://github.com/confluentinc/kafka-connect-datagen/blob/master/src/main/resources/stock_trades_schema.avro.
From the source, the data dictionary for the data set is:

Column	Description	Expected range
UserID	The simulated user who executed the trade	"User_[1-9]{0,1}"
account	Simulated accounts assigned to the trade	"ABC123", "LMN456", "XYZ789"
price	A simulated random trade price in pennies	5 - 1,000
symbol	Simulated stock symbols	"ZBZX", "ZJZZT", "ZTEST", "ZVV", "ZVZZT", "ZWZZT", "ZXZZT"
quantity	A simulated random quantity of the trade	1 - 5,000
side	A simulated trade side (buy or sell or short)	"BUY", "SELL"
event_timestamp	Generated timestamp of when event happened	2021-06-12 12:37:...

Data Preparation

Data generation

Input data for this system was generated from the Stock Trade resource in Datagen. It was configured as a Producer in Kafka, with the configuration code contained in Appendix 1. As the generated data from the stock_trade data set is already predefined by Datagen, there were no data cleaning operations carried out on the data.

Data pipelines

KSQL

The KSQL data stream was created through the datagen stock-trades connector with raw json content, which was cleansed to a stream with columnar data and subsequently filtered into 2 streams:

1. Data for stocks sold
2. Data for stocks bought

Figure 1: Stream and Table creation through KSQL

As the figure above demonstrates, the filtered streams subsequently led to the creation of tables with metrics relevant for clients accessing the data (i.e. highest price, lowest price, and average price of stocks per 10 second window per user).

A tumbling window was selected for the table creation as it would demonstrate the timely access of updated stock price metrics for clients without overlaps, as is the case with hopping windows.

The KSQL code utilised and example outputs are contained in Appendix 2 - KSQL Data.

Spark Streaming

In Spark Streaming, two stream data frames created:

1. stock_string_view
   A complete view of the Datagen Stock Trade data, including the generated output values, as well as the schema information.
2. stock_trade_view
   A stream data frame containing all of the fields from the Stock Trade data stream, to drive the machine learning models and

A sample of the output of the stock_trade_view:

event_key	event_topic	event_timestamp	account	symbol	side	price	quantity	userid
ZJZZT	stock-trades	2021-06-12 12:37:...	LMN456	ZJZZT	SELL	726	4550	User_6
ZWZZT	stock-trades	2021-06-12 12:37:...	LMN456	ZWZZT	BUY	812	3205	User_8

ZJZZT	stock-trades	2021-06-12 12:37:...	LMN456	ZJZZT	SELL	288	2959	User_6
ZTEST	stock-trades	2021-06-12 12:37:...	ABC123	ZTEST	SELL	99	740	User_4
ZJZZT	stock-trades	2021-06-12 12:37:...	XYZ789	ZJZZT	SELL	991	523	User_3
ZTEST	stock-trades	2021-06-12 12:37:...	LMN456	ZTEST	SELL	419	1952	User_5
ZTEST	stock-trades	2021-06-12 12:37:...	XYZ789	ZTEST	SELL	104	1169	User_2

Table 1: Sample output from Stock_Trade_View

Additionally, a tumbling window stream with a 1 minute watermark was created, with the following attributes:

- 60 seconds in length
- Refreshed every 10 seconds

This streaming window was used to aggregate the amount of transactions for each stock symbol:

window	symbol	count
{2021-06-12 13:48...	ZBZX	29
{2021-06-12 13:48...	ZVZZT	19
{2021-06-12 13:49...	ZWZZT	23
{2021-06-12 13:48...	ZBZX	16
{2021-06-12 13:48...	ZVV	23
{2021-06-12 13:48...	ZTEST	21
{2021-06-12 13:48...	ZWZZT	11
{2021-06-12 13:48...	ZWZZT	23
{2021-06-12 13:48...	ZXZZT	24
{2021-06-12 13:49...	ZVV	23
{2021-06-12 13:48...	ZXZZT	24
{2021-06-12 13:48...	ZXZZT	18

Table 2: Sample output of windowed stream, counting symbol transactions in the last 60 seconds

Data analysis

Using the data pipelines from the spark streaming data frames, we were able to create a set of visualisations for the client, which refresh in real-time as shown in Figure 2. The information provided by the three graphs includes:

(a) the relationship (or the lack thereof) in traded quantity and stock price in the latest 200 transactions with a line of best fit,
(b) the latest 10 sell-prices of a stock. For the current prototype, the stock 'ZTest' was selected.
(c) the total traded amounts in the most recent 200 transactions across all stocks.

All graphs shown here are updated simultaneously with the most recent data every 10 seconds. This allows quick insights into the latest stock trends. Once connected to the actual stock data in the upcoming project phase, the current code can be modified with relative ease to include tailor-made graphs for further stock analysis.

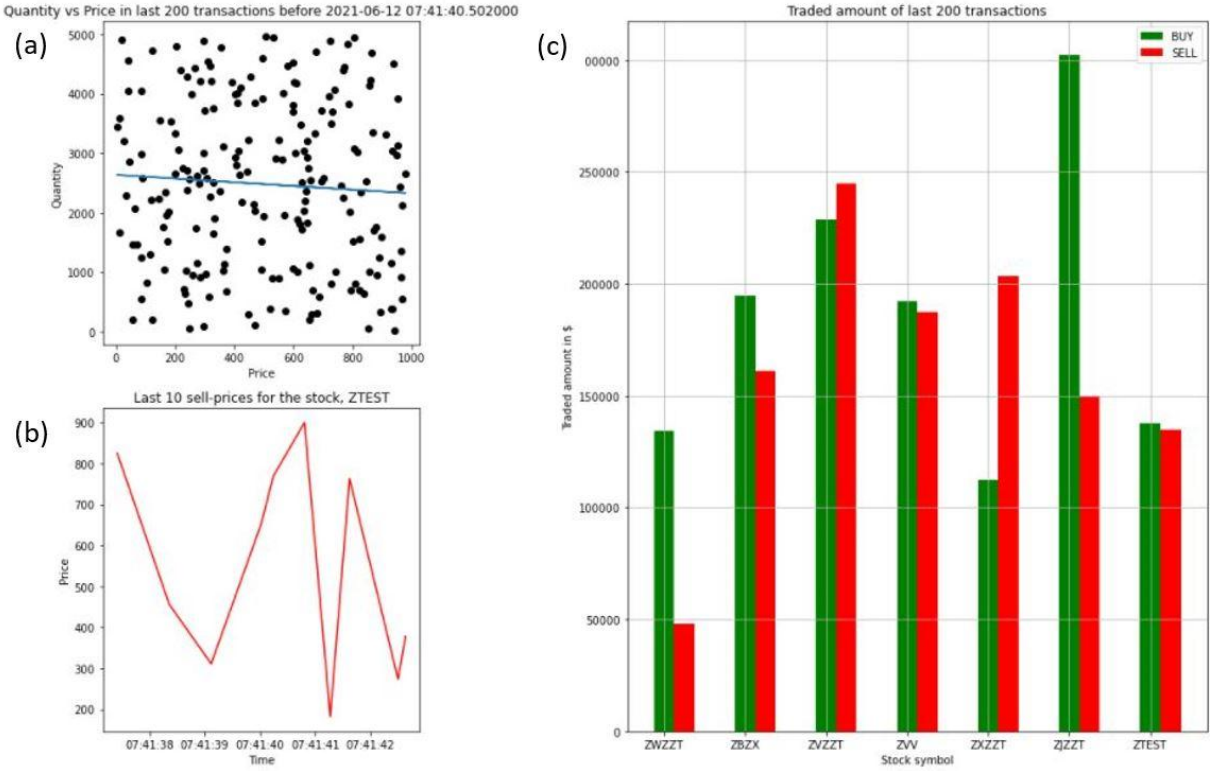


Figure 2: Sample visualisations are generated with (a) a scatter plot showing any possible relationship between the stock price and the traded quantity and a line of best-fit; (b) a line graph displaying the last 10 sell-price of the stock, 'ZTest'; and (c) a column chart of the total traded amounts in the latest 200 transactions for each stock. Note: share prices are displayed in pennies in subplots (a) and (b).

UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

Machine Learning Models and Predictions

As a proof of concept, 2 machine learning models were trained and tested to predict quantity of stock traded, and stock side classification. For this project, a linear regression model was utilised for the stock quantity prediction and a Random Forest model was selected for the side classification prediction.

Model training

The training and test data set was divided into the following portions for both models, utilising 10,000 rows of data:

- Training – 80%
- Test – 20%

The evaluation method selected for the purpose of model selection was:

- Linear Regression (i.e. Quantity Prediction) - RMSE
- Random Forest Classification (i.e. Buy/Sell Classification) - AUROC and accuracy.

Feature selection

Features were grouped into categorical and numerical before the machine learning model is trained.

- Linear Regression
  5 features ('userid','account','symbol','side','price') were selected as predictors of the model. The categorical features ('userid','account','symbol','side') were pre-processed by one-hot encoding.

- Random Forest Classification
  5 features ('userid','account','symbol','quantity','price') were selected as predictors of the model. The categorical features ('userid','account','symbol') were pre-processed by one-hot encoding.

Model real-time predictions and evaluation

Using the Linear Regression model in Pyspark, it was possible to make predictions on the quantity traded, based on the features in the Stock Trade View. A sample of these predictions is contained in Appendix 3 - ML Model Outputs.

The training and test results were:

- Root Mean Squared Error (RMSE) on train data = 1441.293
- Root Mean Squared Error (RMSE) on test data = 1449.018

Similarly, a Random Forest Classification model was used to predict whether a stock trade was either “BUY” or “SELL” and a sample output is also contained in Appendix 3 - ML Model Outputs.

Using the Random Forest Classification model in Pyspark, the training and test results were:

- AUROC on train data = 0.566
- AUROC on test data = 0.520
- Accuracy on train data = 0.545
- Accuracy on test data = 0.516

From the results above, it can be seen that the random results of the data generator cause some trouble for our predictive model and the accuracy of results is ~50% for the test set. However, as this is a proof of concept for the streaming data pipelines to the machine learning models, we have achieved our aim of porting streaming data for training / testing a machine learning model. In Phase 2, the aim will be to port live stock data from the client’s stream, which will improve the accuracy of our predictive models.

Further work and enhancements

The current development of the prototype of real-time stock analysis using data streaming with Kafka, Spark, and Python has shown great potential in extracting insights efficiently and effectively. Based on this, the following recommendations are made for Phase 2 of the project:

(a) As a proof of concept, the streaming data pipelines were created using the Datagen Stock Trade data, as the client's live stock trade data was commercial-in-confidence. In the next phase, the live data shall be mapped in so that the plots and predictive models are able to access it.

(b) Currently, the streamed data are aggregated and summarised in 2 tables. With a real data source, more fields may be available and more tables may be required for gaining adequate insights into the stock trends.

(c) Visualisations could provide quick identification of useful trends, and the 3 sample plots demonstrated here can be further refined. Since more stocks are anticipated in the real stock datasets. It would be important to provide the ability for a subset of selected stocks to be displayed for examination. For instance, stocks could be grouped by industry-types, or specific stocks of interest could be compared simultaneously. The duration of the collected data and/or the number of transactions could also be adjusted in the display.

(d) Real-time predictions using models such as time-series analysis is also recommended beside the ML models used in this report to allow for timely and effective decision making.

Appendix 1 - Kafka Producer Stream

Kafka producer configuration

```
{
  "name": "datagen-stocktrades",
  "config": {
    "connector.class": "io.confluent.kafka.connect.datagen.DatagenConnector",
    "key.converter": "org.apache.kafka.connect.storage.StringConverter",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "kafka.topic": "stock-trades",
    "max.interval": "100",
    "iterations": "-1",
    "quickstart": "Stock_Trades"
  }
}
```

Appendix 2 - KSQL Data

KSQL Code

Initial Stream Creation

```
CREATE STREAM stocks (payload STRUCT<userid VARCHAR,
                                     account VARCHAR,
                                     side VARCHAR,
                                     symbol VARCHAR,
                                     quantity BIGINT,
                                     price BIGINT>)
WITH ( KAFKA_TOPIC='stock-trades',
            VALUE_FORMAT='JSON',
            PARTITIONS=1);
```

Clean Stream

```
CREATE STREAM stock_clean
  AS SELECT  payload->userid AS userid,
             payload->account AS account,
             payload->side AS side,
             payload->symbol AS symbol,
             payload->quantity AS quantity,
             payload->price AS price
 FROM  stocks;
```

Filtered Streams (x2)

```
CREATE STREAM STOCK_SELL
             AS SELECT * FROM stock_clean
             WHERE side = 'SELL';
CREATE STREAM STOCK_BUY
             AS SELECT * FROM stock_clean
             WHERE side = 'BUY';
```

Aggregated Tables with Windows (x2)

```
CREATE TABLE price_buy
        AS SELECT userid,
        max(price) as HIGH_PRICE,
        min(price) as LOW_PRICE,
        round(avg(price),2) as AVG_PRICE
FROM  STOCK_BUY
WINDOW TUMBLING (SIZE 10 SECONDS)
GROUP BY userid
EMIT CHANGES;

CREATE TABLE price_sell
        AS SELECT userid,
        max(price) as HIGH_PRICE,
        min(price) as LOW_PRICE,
        round(avg(price),2) as AVG_PRICE
FROM  STOCK_SELL
WINDOW TUMBLING (SIZE 10 SECONDS)
GROUP BY userid
EMIT CHANGES;
```

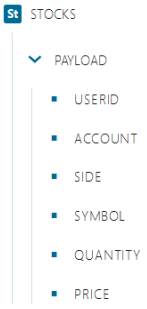
Stream Data Samples

Initial Stream

DATA

PAYLOAD
{"USERID":"User_5","ACCOUNT":"ABC123","SIDE":"SELL","SYMBOL"...
{"USERID":"User_6","ACCOUNT":"XYZ789","SIDE":"BUY","SYMBOL":...
{"USERID":"User_3","ACCOUNT":"ABC123","SIDE":"SELL","SYMBOL"...

SCHEMA

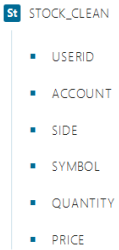


Clean Stream

DATA

USERID	ACCOUNT	SIDE
User_7	ABC123	SELL
User_4	LMN456	SELL
User_6	ABC123	SELL

SCHEMA

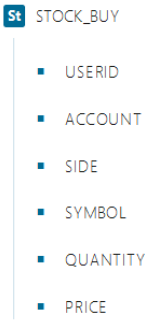


Filtered Streams

DATA

USERID	ACCOUNT	SIDE
User_9	XYZ789	BUY
User_1	XYZ789	BUY
User_5	ABC123	BUY

SCHEMA



Aggregated Tables with Window

DATA

	HIGH_PRICE	LOW_PRICE	AVG_PRICE
	936	21	592.44
	868	10	440.3
	936	21	617.63

SCHEMA

Appendix 3 - ML Model Outputs

Quantity Prediction

event_key	event_topic	event_timestamp	account	symbol	side	price	quantity	userid	userid_ind	userid_ohe	account_ind	account_ohe	symbol_ind	symbol_ohe	side_ind	side_ohe	features	prediction
ZTEST	stock-trades	2021-06-13 04:37:...	ABC123	ZTEST	BUY	930	2072	User_8	0	(8,[0],[1.0])	2	(2,[],[])	2	(6,[2],[1.0])	1	(1,[],[])	(18,[0,12,17],[1....	2481.052
ZVZZT	stock-trades	2021-06-13 04:37:...	XYZ789	ZVZZT	BUY	27	3816	User_5	5	(8,[5],[1.0])	1	(2,[1],[1.0])	1	(6,[1],[1.0])	1	(1,[],[])	(18,[5,9,11,17],[...	2432.537
ZBZX	stock-trades	2021-06-13 04:37:...	XYZ789	ZBZX	SELL	177	1909	User_2	1	(8,[1],[1.0])	1	(2,[1],[1.0])	4	(6,[4],[1.0])	0	(1,[0],[1.0])	(18,[1,9,14,16,17...	2439.304
ZXZZT	stock-trades	2021-06-13 04:37:...	XYZ789	ZXZZT	BUY	107	4485	User_4	4	(8,[4],[1.0])	1	(2,[1],[1.0])	5	(6,[5],[1.0])	1	(1,[],[])	(18,[4,9,15,17],[...	2597.957

ZBZX	stock-trades	2021-06-13 04:37:...	ABC123	ZBZX	SELL	379	4462	User_1	3	(8,[3],[1.0])	2	(2,[],[])	4	(6,[4],[1.0])	0	(1,[0],[1.0])	(18,[3,14,16,17],...	2512.322
ZWZZT	stock-trades	2021-06-13 04:37:...	XYZ789	ZWZZT	BUY	390	604	User_1	3	(8,[3],[1.0])	1	(2,[1],[1.0])	6	(6,[],[])	1	(1,[],[])	(18,[3,9,17],[1.0...	2626.288
ZVZZT	stock-trades	2021-06-13 04:37:...	ABC123	ZVZZT	BUY	996	989	User_8	0	(8,[0],[1.0])	2	(2,[],[])	1	(6,[1],[1.0])	1	(1,[],[])	(18,[0,11,17],[1....	2482.265
ZBZX	stock-trades	2021-06-13 04:37:...	XYZ789	ZBZX	SELL	732	1297	User_6	6	(8,[6],[1.0])	1	(2,[1],[1.0])	4	(6,[4],[1.0])	0	(1,[0],[1.0])	(18,[6,9,14,16,17...	2582.387
ZVV	stock-trades	2021-06-13 04:37:...	ABC123	ZVV	SELL	634	728	User_7	8	(8,[],[])	2	(2,[],[])	3	(6,[3],[1.0])	0	(1,[0],[1.0])	(18,[13,16,17],[1...	2460.798
ZJZZT	stock-trades	2021-06-13 04:37:...	LMN456	ZJZZT	SELL	548	4466	User_5	5	(8,[5],[1.0])	0	(2,[0],[1.0])	0	(6,[0],[1.0])	0	(1,[0],[1.0])	(18,[5,8,10,16,17...	2404.284

Side Prediction

side	userid	account	symbol	quantity	price	prediction	probability
SELL	User_2	LMN456	ZVZZT	10	676	1	[0.47111712280514...
SELL	User_2	LMN456	ZVZZT	102	60	1	[0.46948318328653...
SELL	User_2	LMN456	ZVZZT	247	662	1	[0.49183140851943...
SELL	User_2	LMN456	ZVZZT	259	563	1	[0.49618185259590...
SELL	User_2	LMN456	ZVZZT	923	87	1	[0.48552230309955...
SELL	User_2	LMN456	ZVZZT	981	340	0	[0.50476007015263...
SELL	User_2	LMN456	ZVZZT	1002	638	0	[0.50907995923077...
SELL	User_2	LMN456	ZVZZT	1127	424	0	[0.50476007015263...
SELL	User_2	LMN456	ZVZZT	1172	972	0	[0.51213259971621...
SELL	User_2	LMN456	ZVZZT	1574	645	0	[0.51354072290865...