

**Project: Big Data Architectures**

**SPROUT**

**Ariston Lim**

**Hung Nguyen**

**Julio Candela**

**Nejada Karriqi**

**Valdemar Hernández**

**Big Data Management**

**Sergi Nadal, Alberto Abelló, Oscar Romero**

**June 2020**

**Table of Contents**

[**1. Introduction**](#_ochrocl7nqj0) **3**

[1.1 Data](#_oqguereejrer) 3

[1.2 Model](#_lcu27i9f9s48) 4

[1.3 Algorithms](#_1qkocjgtkwmn) 4

[**2. Functional Architecture**](#_86roq8p0aegm) **5**

[2.1 Batch Layer](#_oin07hudvdni) 5

[2.2 Speed Layer](#_d3wfzy8ayp46) 5

[2.3 Serving Layer](#_e0yvgrbarf5d) 5

[**3. Tool Selection**](#_r8r18q51x762) **6**

[3.1. Data Ingestion](#_qiwtan2nrbuc) 6

[3.2. Batch Processing](#_j4pf9v8sijcx) 7

[3.3. Stream Ingestion and Stream Dispatcher](#_mmh68zn640rf) 7

[3.5. Stream Processing](#_3mddjy8z31pd) 7

[3.6. Graph Modelling](#_j05ffx5p7y4l) 7

[3.7. Real-time View](#_fcgq2djybqqk) 8

[**4. Use Cases and Data Flows**](#_ycy0r5mvbt2t) **8**

[**5. PoC Description**](#_sieulg5tt0af) **11**

[5.1. Batch Data Processing with Spark](#_3r42i4741rrr) 11

[5.2. Sentiment Analysis with Real-time feedbacks](#_4wid4kax09yu) 12

[5.3. DataView - Graph Modelling with Neo4j](#_9hhj6zdyot8s) 12

[5.4. Data Visualization with ElasticSearch](#_x8ocu7udgo3d) 13

[5.5. Product Recommendation with Neo4j and ElasticSearch](#_xc1924p9870t) 13

[**Appendix 1 - Data Flow Figures**](#_bbdjr6gutw60) **14**

[**References**](#_x03gr8lo9bv1) **16**

# 1. Introduction

The project described in this document provides a Big Data solution for Sprout, a startup company that works with a “Farm-To-Table” concept to provide local, organic products of the highest quality to its customers. To ensure this, data management and analysis is required in order to identify what the customers want (types of products) and how they want it (quality of products) based on their purchases and their feedback. The output of this implementation will allow the company to take the appropriate actions to maintain the best relationship possible with the customers, and, in consequence, make the business thrive, whether it is by suggesting the most suitable set of products, recommending prices based on the market or reassessing which products to promote depending on the feedback from customers.

## 1.1 Data

Sprout generates several data files to store information regarding different parts of the business, such as the purchase orders, price history by region for their available products and services or a list of suppliers for the organic products including their location. To identify what the customers usually buy, their purchase orders will be considered. Data arrives on a constant basis throughout the day, with Sprout estimating an average of 1000 daily orders per region after the first five years of operation, with each order containing one or more different types of products. This information is stored in a daily CSV file containing the orders history, with parameters such as the Order ID, Product Name, Quantity, Date, Unit Price, Customer ID among others; these CSV files will be considered the first input for the project’s implementation and is expected to amount for more than 1TB of information after 8 to 10 years of operation. For testing purposes, we are using a dataset from Kaggle [7] to simulate orders made to Sprout.

Additionally, the European Union, through its Open Data Portal [1], provides several datasets in the form of Excel files, updated on a monthly basis, containing the market prices for selected animal [2], fruit [3], dairy [4], vegetable [8] and vegetal [5] products, which will be a second input to make sure the prices set by the company are competitive.

The third and final input is the customer feedback, obtained from the company’s own app, which requests the customer’s input once the order has been delivered. Each feedback received includes ratings (in a scale from 1 to 5) and comments regarding the quality of products received, the price, time of delivery and overall experience while placing the order. For this input, the company should be ready to receive one feedback per order placed in the best-case scenario, which again would result in an average of 1000 comments and ratings per day, with an estimated size of 280 bytes (considering Twitter’s limit as a reference parameter). This data, stored by the company in JSON-type format, will be treated as an incoming stream, in order to obtain a better insight on the products with good or bad reviews.

## 1.2 Model

The output of this model should accomplish two goals for the analysts, first, organize the incoming data so it can be later used for specific data analysis related to the products provided by the company, and second, allow a quick visualization of which products and services receive the best (or worst) feedback from the customers.

For the first goal, given the different types of input files available and the uncertainty on the requirements for the data analysis (depending on what the company needs at any point in time in the future), a data lake would help in this regard, storing the raw data mentioned earlier, such as the orders placed, the market prices from the EU and the reports containing feedback from the users. This data, stored in the data lake, will be part of the batch layer, where several Spark processes will be implemented to retrieve the necessary information to perform the graph analytics in the serving layer.

Regarding the second goal, the visualization of the feedback, it is expected that the incoming data from the users, coming from the application, will be received in a JSON-type format. The processing for this data will be part of the Speed layer, which will perform Spark Streaming operations to send data to the data lake and also to the visualization tool.

## 1.3 Algorithms

The raw data stored in the data lake will be transformed so it can be used for a Collaborative Filtering [6] through the use of property graphs. The idea of Collaborative Filtering is to recommend a product to an user A based on orders from user B when they have the same preferences or same opinions about the same products. The recommendations are built upon the existing ratings of other users. Products which come out of the Collaborative Filtering will be filtered. Products which have received bad reviews will be removed to form the final recommended list.

Meanwhile, in the speed layer, the processed data from the customer feedback will be used to visualize details about their perception of the company’s products, helping the company to react accordingly. The output of the stream processing will be stored in JSON documents, which will provide an easy structure for queries that need to show results from one product in particular, or aggregate results by product, resulting in a quick visualization at any point in time.

# 2. Functional Architecture

The proposed architecture was designed to provide a strong Big Data component into our business application ensuring its main characteristics as velocity, variety, volume, veracity and value. The architecture is composed by the following layers:

## 2.1 Batch Layer

The batch layer will be in charge of the data ingestion, data storage in the Data Lake and batch processing. The objective of this layer is to normalize the data from different sources, formats and sizes so that the serving layer can consume it for Analytics. For the purpose of this project, the data ingestion step won’t be part of the implementation; however, it represents an important step in the overall architecture. The Data Lake will store data from web-sales sources and feedback (including ratings and comments) from customers in csv, excel and json format. Finally, the data will be processed in the batch processing step using Spark in order to generate the required input format for the Neo4j data view. In this layer, the business users or analysts still don’t have access to the data. These batch jobs will be executed daily, at the end of the day, to have updated information for analytical purposes.

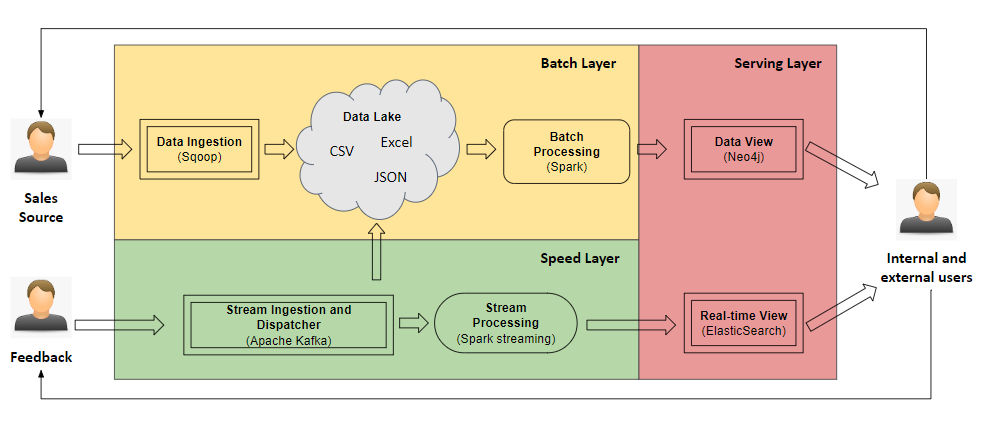
## 2.2 Speed Layer

The speed layer will have the responsibility to ingest, dispatch and process the streaming data in order to generate real-time views for both internal and external users about the perception of the company’s organic products. The PoC implementation won’t take into account the stream ingestion and dispatcher steps done by Apache Kafka, but the architecture considers it so important to enhance velocity. The data ingestion layer will collect the customer’s feedback which includes comments and ratings about orders, products, experience, etc. The dispatcher, which will be executed by the same tool, will receive and send the stream to both the Data Lake and the stream processing. At the end of this layer, Spark streaming, the processing tool, will turn comments and ratings into a qualification for the product resulting in the input for the real time analysis. The structure of the flow also ensures that the feedback matched with the corresponding product providing veracity to the process. As mentioned before, flexibility, velocity and veracity are keys in order to get real value for the company during this stage.

## 2.3 Serving Layer

The serving layer will be responsible for providing real value to the business by generating an analytical data view from the batch layer flow and producing real time insights from the speed layer flow. The first data view will be implemented in Neo4j and it will allow graph analytics with sales and feedback information. The graph structure will allow users to exploit big amounts of data and get rapid analytics for later actions. Specifically, the project aims to find the best recommendations for customers and validate its response according to the feedback in an iterative process. For this reason, the new metadata of recommendations resulting from the graph algorithms is also stored. The second data view will be implemented in ElasticSearch and it will allow us to store, index and visualize real time information about the customer’s perception of our products. Velocity is a key at this point, since automatic actions should be taken to deal with unsatisfied customers and enhance satisfied ones. The main importance of this layer is to generate knowledge from the previous layers thereby providing value to the business in order to make fast and data oriented decisions.

The following diagram shows the final Big Data architecture for Sprout:



# 3. Tool Selection

In this section, for each module in our architecture, we will discuss potential tools to choose to be implemented in our project based on feasibility, performance and scalability.

## 3.1. Data Ingestion

Apache Sqoop is used for bulk data transferring between datastores. Raw sale data files will be moved to a data lake which contains a large amount of data coming from various sources. Data can be transformed to HDFS files which support distributed processing. Another option would be export data from databases to external files and then load them to the HDFS system, which requires more effort.

## 3.2. Batch Processing

Sales data is stored in a data lake which serves multi-purpose processing. Data can be in the form of raw data (csv, json,...) or data stored in a database. In this step, data will be processed and transformed to become input for Neo4j in the next stage. Apache Spark will be used in this module for large scale data processing since Spark is much faster than Hadoop due to in-memory computing.

## 3.3. Stream Ingestion and Stream Dispatcher

Real-time feedback is generated by users and we need a tool to ingest these data before processing them. Apache Kafka is capable of handling trillions of events a day, thus, it will not have any problems when our business scales fast. Active MQ and RabbitMQ are also offering pub-sub message management but Kafka has higher throughput, which is more suitable for large-scale systems. We also use Kafka to dispatch the stream to the serving layer and the data lake. Logstash is an alternative solution for dispatching but using Kafka will save us time for integrating different tools together.

## 3.5. Stream Processing

Feedbacks will be sent to a stream processing module to extract new insights from them. The output of this process will be passed to ElasticSearch and stored in our data lake for further processing. For example, rating for each product will be updated continuously according to the feedback and top 10 products in a certain amount of time are obtained and displayed on the home page to attract more customers and have a better recommendation. Each feedback has information about a specific product, a rating score and a comment from a user. We have studied Spark Streaming and it perfectly fits for this problem. Spark Streaming provides scalability, high-throughput, fault-tolerant stream processing. Streams of feedback will be processed in batches.

## 3.6. Graph Modelling

Neo4j will be our choice to build a recommendation system which serves both buyers and farmers. Since we are focusing on graph analytics, property graph and graph algorithms are better choices than knowledge graphs. We will implement this part in the SDM project.

## 3.7. Real-time View

Elasticsearch is chosen for live data visualization and it will be hosted on Amazon Web Services (AWS) platform. We also consider Elastic Cloud but we prefer AWS since we have more tools to integrate with from AWS stack. We will implement this part in the SDM project.

# 4. Use Cases and Data Flows

**Data Ingestion**

1. **Sales & Price Data**

Our source of data will be the CSV files coming from the sales record and the excel files obtained from the European Union’s Open Data Portal. The sales record will be stored in daily bases and will be handled by a batch job, while excel files will be obtained monthly. This data will help us to keep track of our business performance and to maintain the same or better level of product prices. For the data extraction and transferring process, we are going to use SQOOP, the command line interface used to transfer data. After having finished with data extraction and transfering, the ingested data will be preprocessed if needed. During the preprocessing, we can perform different transformations, updates and filtering, according to our requirements. Then the data will be transformed and stored in Data Lake in CSV format. Furthermore, through a batch processing the CSV files will be prepared and sent to Neo4j for analysing purposes.

1. **Feedback Data**

As the source for our stream data (feedback) will serve the Sprout Web Application. All the feedback from the users, regarding the purchases or the farming experience, will be received in JSON format. Once receiving the data, we are going to use Apache KAFKA to ingest and transform the data streams. If the stream transformation will be successful we will send the data to the next stage where we are going to perform the sentiment analysis using Spark Streaming. If for any reason, the transformation fails, we will generate an error, keep this record in our data lake and end this process. The transformed data will be stored on the data lake in the original format (JSON). After the sentiment analysis, batches of processed data will be sent to ElasticSearch for analyzing purposes.

**Data Processing**

1. **Neo4j Analysis**

Using Apache Spark, we will transform the sales and pricing data into csv files. Additionally, we will also transform the feedback data from JSON into csv files. These data will then be the input source of Neo4j. We will perform:

* 1. Collaborative Filtering to provide product recommendation.
  2. Based on the feedback data, evaluate the accuracy and customer satisfaction from our product recommendation.
  3. Best sold products.

We will store the results as sub-graphs within Neo4j and show the results of the product recommendation to the customers.

1. **Streaming Analysis**

We will use Spark Streaming to perform sentiment analysis. The raw streaming data (in JSON format) will contain timestamp, user\_id, product\_id, rating (integer), and comments (text). For each comment in the feedback data, we will assess the sentiment whether it is positive, neutral, or negative based on the keywords. Afterwards, we will send the processed feedback data to ElasticSearch. We will create visualisations using Kibana, for example:

1. Daily Average Rating
2. Daily Average Rating per Product
3. Average Rating per Users
4. % of Positive Reviews per Day

The diagrams for Data Ingestion and Data Processing are available in Appendix 1 - Data Flow Figures.

**Performance and quality metrics**

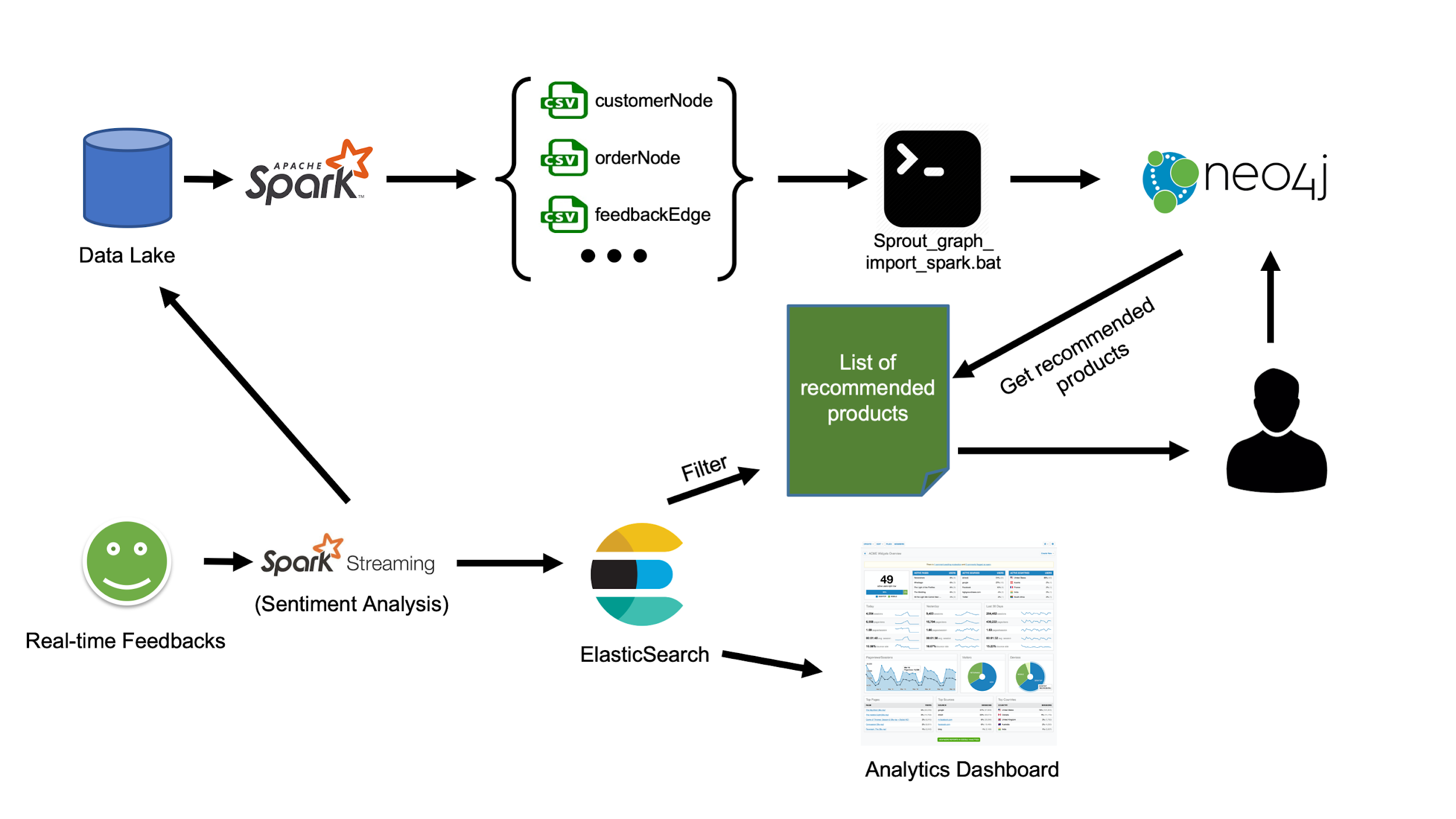
In order to assure a high performance of our architecture, it’s pivotal to continuously look at some metrics of our processes such as:

* Historical percentage of utilization to know the right moment to scale
* Number of I/O operations should be reduced to improve performance (using Spark)
* Percentage of work done in parallel (batch and speed layer)
* Throughput should also be reduced to reduce response times of I/O (Spark)
* Number of job failures (Apache Spark)
* Execution time from the batch processing and the stream processing
* Response query time from Neo4j and ElasticSearch

# 

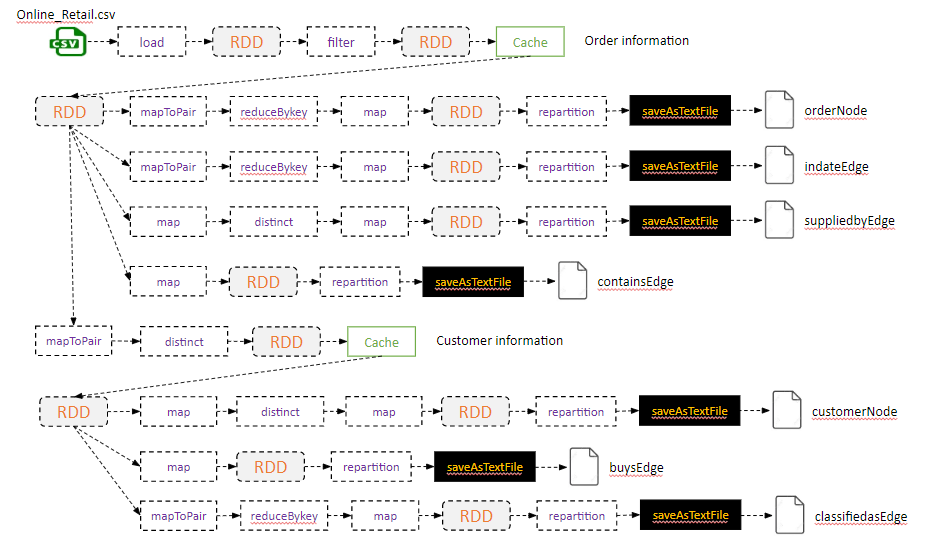
# 5. PoC Description

In this section we will demonstrate how we are building our product recommender with the architecture shown in the below diagram

****

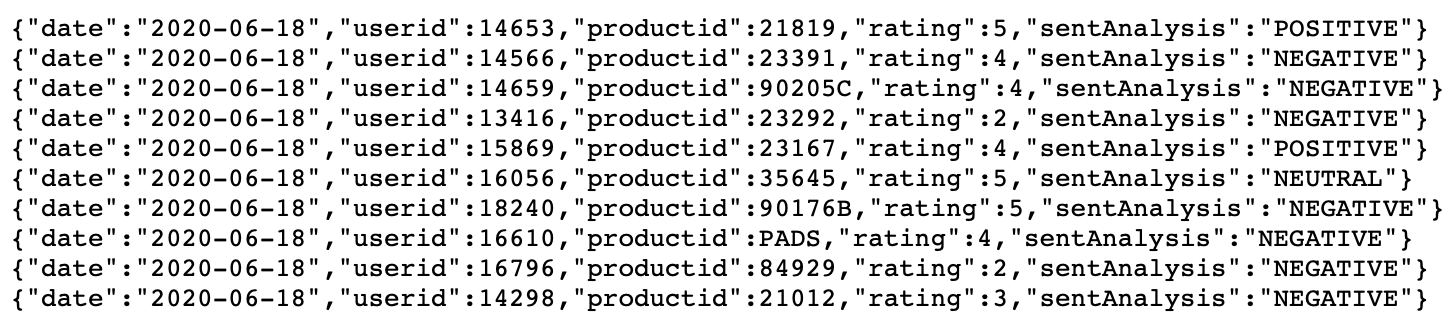
## 5.1. Batch Data Processing with Spark

The data coming from the data lake is processed with Spark to be transformed to be able to be imported to Neo4j. This stage consists of 3 main processes which covers transactions (csv), EU market prices (excel) and feedback (json). We cache RDDs which are going to be used in the next steps in order to harness parallelism. As an example, we show the RDD flow diagram for the **transactions** process. The source code can be found in the following github repository: <https://github.com/JulioCandela1993/BDM_Sprout> (inside folder ***DataIntegration\Spark****)*.



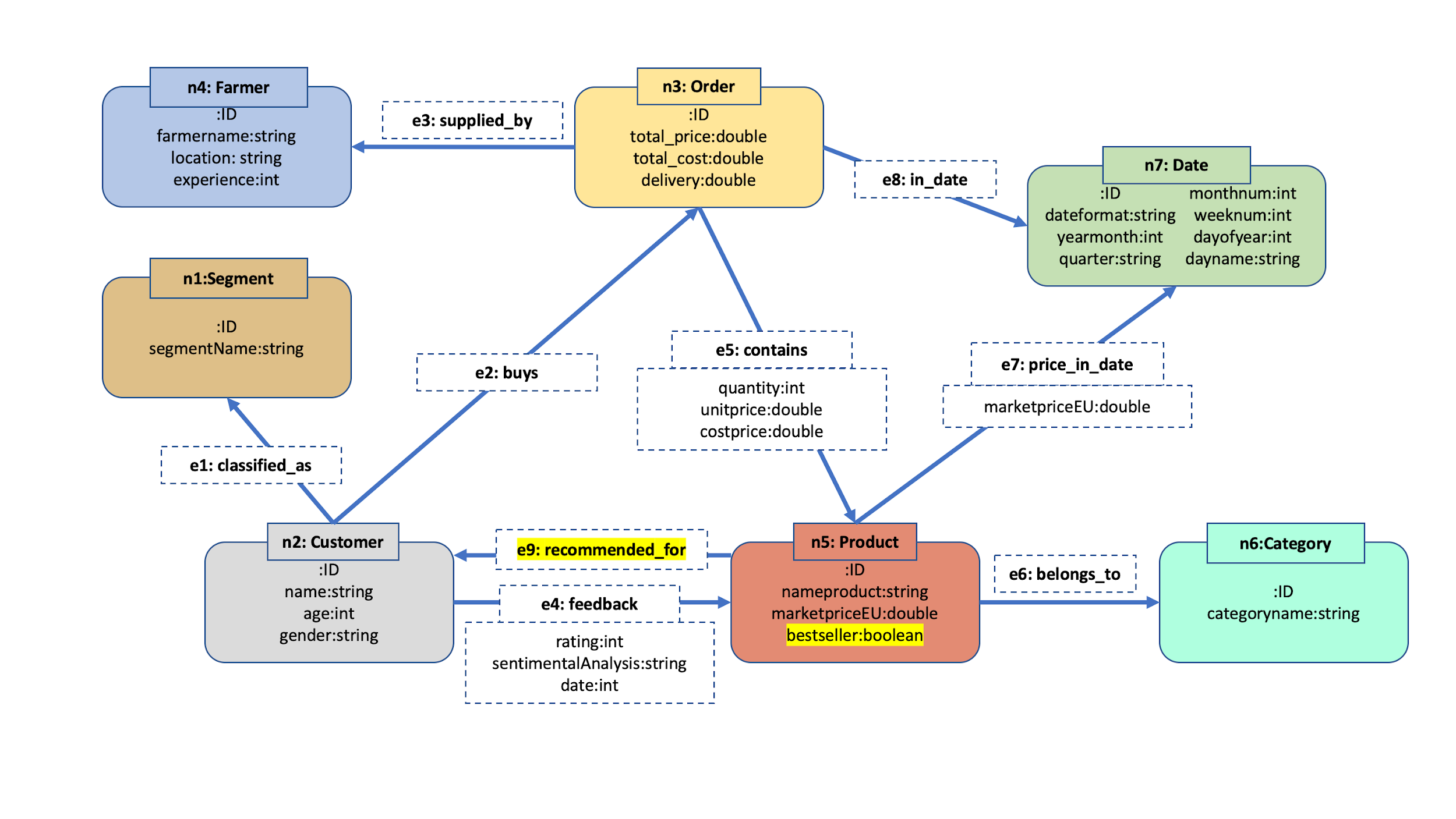
## 5.2. Sentiment Analysis with Real-time feedbacks

The data streams are coming in real time from our web application. We are going to get all the streams and analyse them based on the nature of the words the people are using to describe their experience with us. The stream will be evaluated and at the output of this process will contain the date when the review was given, the client ID, the product ID, the rating and the feedback category which we are calling sentAnalysis (negative, neutral or positive). This is a way for us to understand if the clients are happy with us and if they are getting what they were expecting. For the PoC, we have simulated real time comments with the *“Execute\_streams.bat”* which will be processed by Spark Streaming. The whole implementation of the stream processing can be found in the folder ***DataIntegration\SparkStreaming***. The image below shows examples of our output data from SparkStreaming.

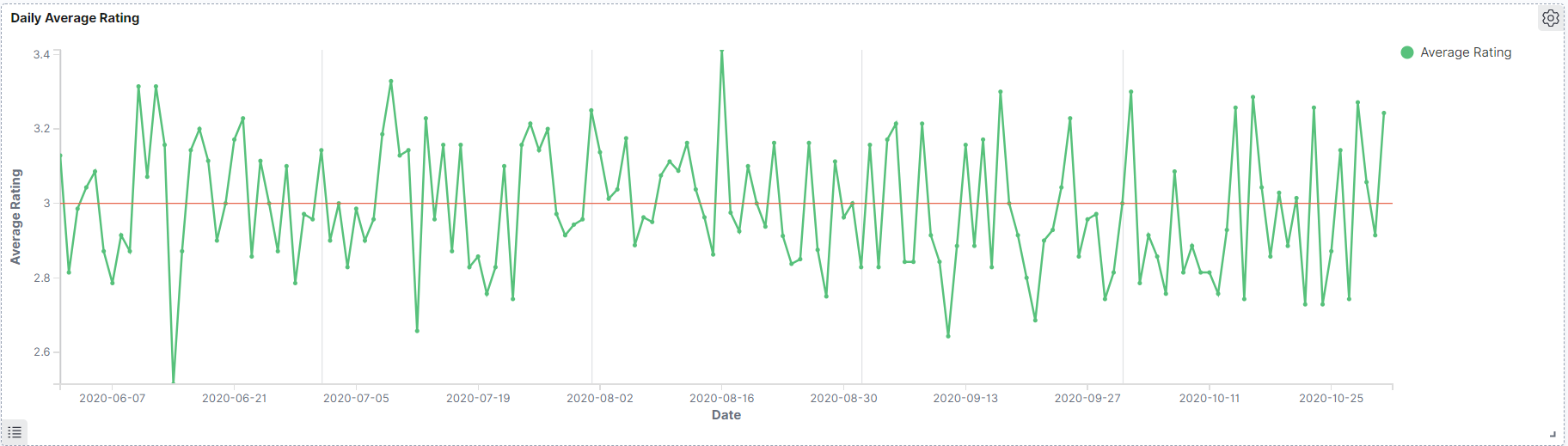


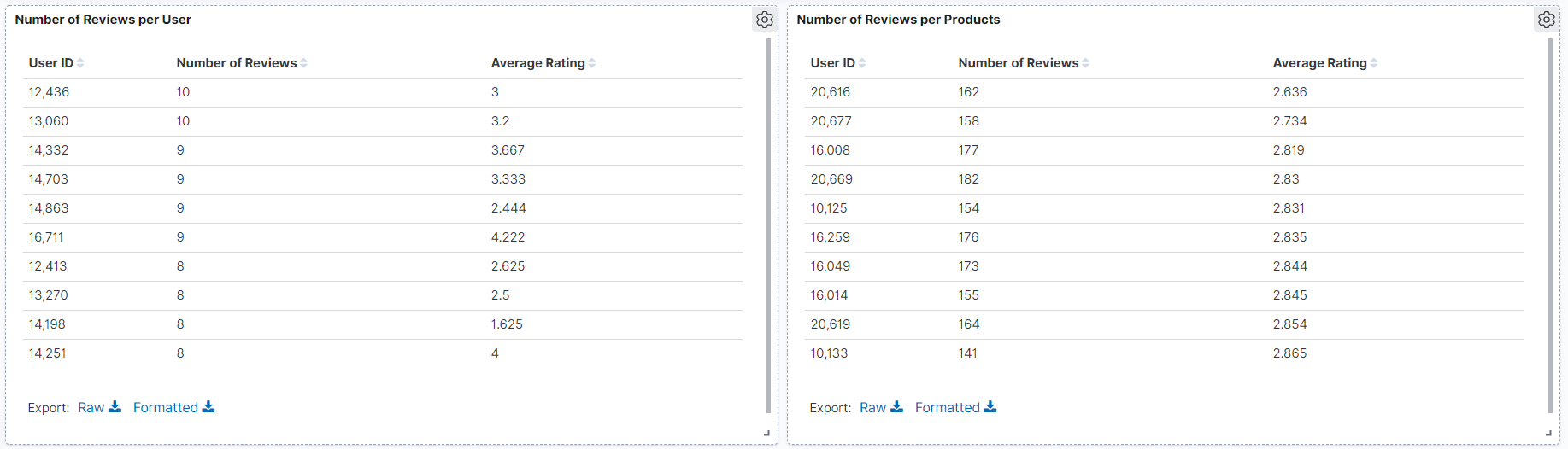
## 5.3. DataView - Graph Modelling with Neo4j

The diagram below shows our graph model implemented in Neo4j.



## 5.4. Data Visualization with ElasticSearch





The diagrams above show some examples of the visualisations we created. The first graph shows the average rating daily. This is useful to see the trend over time. The second diagram shows the average ratings by users and products. From this we can get an idea of which users/products that have bad ratings and we can try to fix any issue.

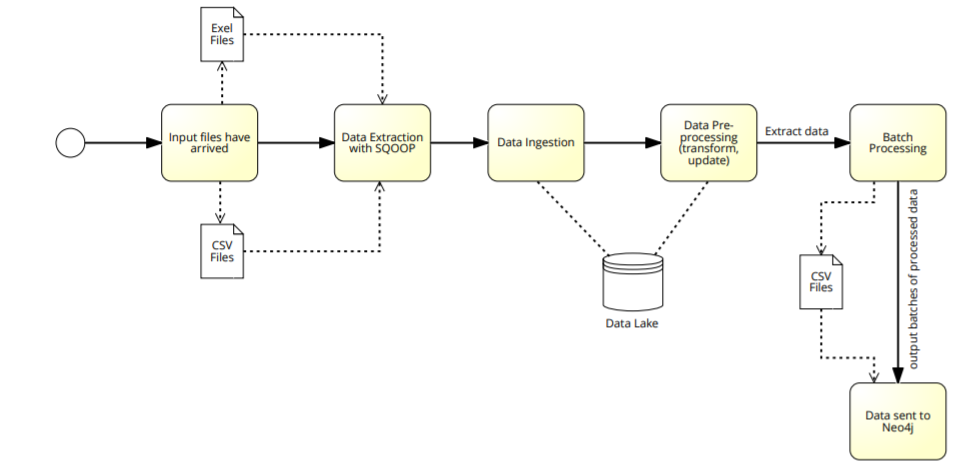
## 5.5. Product Recommendation with Neo4j and ElasticSearch

We are using graph analytics to build a products recommender which uses Neo4j to perform collaborative filtering to output recommended products. The result will be filtered with real-time feedback retrieved from ElasticSearch before showing to our customers. This recommender will give recommendations of products to a specific user based on two strategies: based on orders and based on user feedback. The detailed process is implemented in the SDM project.

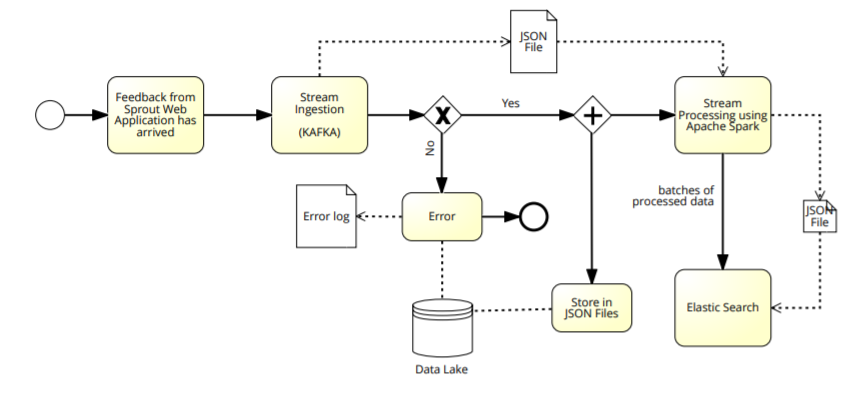
# Appendix 1 - Data Flow Figures

**Data Ingestion**

Sales & Price Data

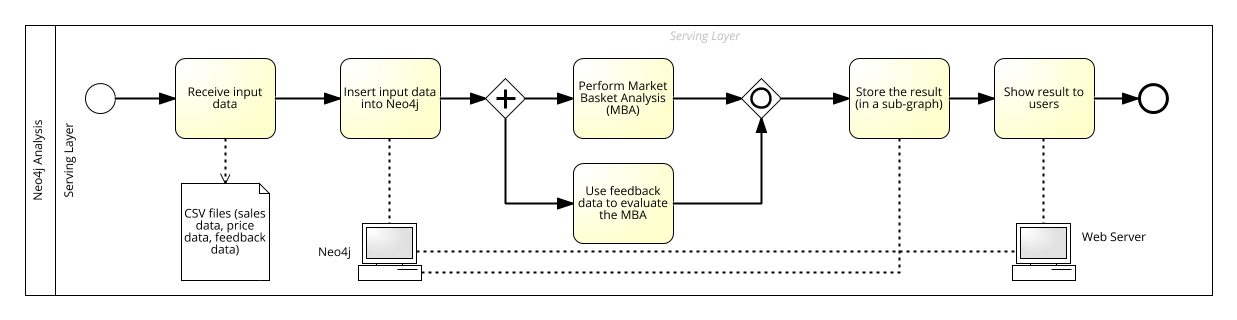


Feedback Data

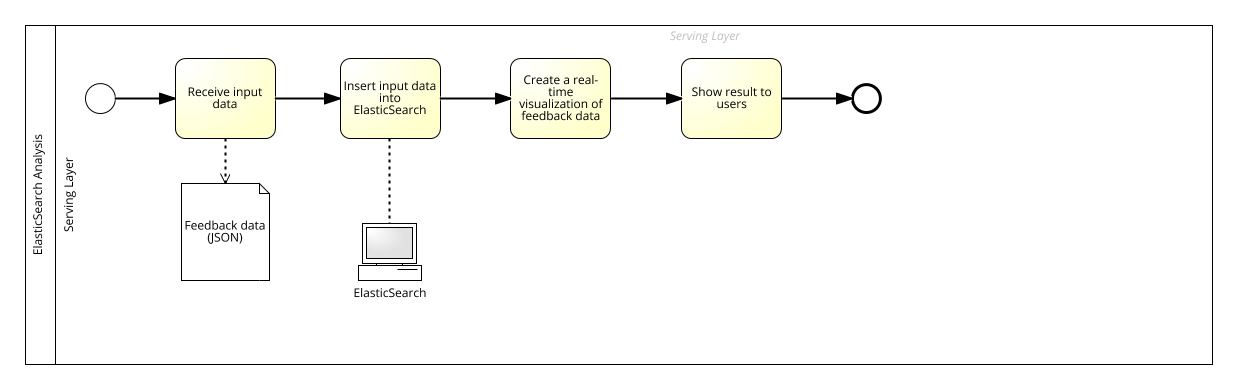


**Data Processing**

Neo4j Analysis



ElasticSearch Analysis



# References

1. (2020). *Open Data Portal Home. EU Open Data Portal.* Retrieved from <https://data.europa.eu/euodp/en/home>
2. (2020). *EU prices for selected animal products*. EU Open Data Portal. Retrieved from: <https://data.europa.eu/euodp/data/dataset/ef78194c-e94c-47dd-85f7-44ba98b6746c>
3. (2020). *EU prices for selected fruit products*. EU Open Data Portal. Retrieved from: <https://data.europa.eu/euodp/data/dataset/eu-prices-for-selected-fruit-products>
4. (2020). *EU prices for selected dairy products*. EU Open Data Portal. Retrieved from: <https://data.europa.eu/euodp/data/dataset/eu-prices-for-selected-dairy-products>
5. (2020). *EU prices for selected vegetal products*. EU Open Data Portal. Retrieved from: <https://data.europa.eu/euodp/data/dataset/eu-prices-for-selected-vegetal-products>
6. J. Ben Schafer. Dan Frankowski. Jon Herlocker. Shilad Sen. (2007). *Collaborative Filtering Recommender Systems.* LNCS. Retrieved from: https://link.springer.com/chapter/10.1007/978-3-540-72079-9\_9
7. (2017). Online Retail. Kaggle. Retrieved from: <https://www.kaggle.com/puneetbhaya/online-retail>
8. (2020). *EU prices for selected vegetable products*. EU Open Data Portal. Retrieved from: <https://data.europa.eu/euodp/data/dataset/b8c13971-e315-470e-a125-40d67fe7067b>