MIT 805 - Project Assignment

Part 2

Vimal Ranchhod

u21794023

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Faculty of Engineering, Built Environment & IT

University of Pretoria

Date: 14 November 2021

1 Changes from Assignment-Part1

In part 1 the dataset which was 5.67Gb which was significantly larger to process, and therefore a smaller size dataset was the way forward for part 2. All information is still relevant for part one in terms of looking at ecommerce data, however we will now consider a cosmetic store[2] where we will analyse data over a 2-month period from January 2020 to February 2020 and this approximates to 1Gb in size. There was no change in the data layout with regard to column titles and data types, so that will remain the same. A missing component from part 1 was data pre-processing, so this was done within python and will be described below.

1.1 Data Pre-processing

Initially, the files for 2 months were two separate files and therefore this had to be appended to create one file for processing within the map-reduce algorithm.

Since this is an online store, a number of different event types could occur. Executing the "unique()" action on the dataframe yielded several event types and these include "view","cart","purchase", and "remove from cart". Purchase was the field with actual sales, and this will be considered for our analysis.

The aim was to segment the customers based on their consumption, how often they purchased and when was their last purchase. Three fields from the dataset become more relevant, mainly, "price","user_id" and "event_time". Null values can change the outcome of the analysis, and therefore these would typically be removed or replaced with a value appropriate to the analysis. Using the "isnull" method, the three relevant fields were reviewed and none of these had any null values. We are expecting a unique customer count of 49473 over a period of two months. Detail of the pre-processing can be found on the following Github link.

2 MapReduce

2.1 Introduction

Large datasets that require processing for analysis are often constrained by a single computer, as these have limited resources for processing gigabytes or terabytes of data. One option is to increase the performance of a single computer, however this is quite a costly exercise and is often referred to as vertical scaling. Vertical scaling requires adding more cores or RAM to the machine. Horizontal scaling has the primary architecture of adding more computers to a cluster, hence instead of scaling up in size, work is distributed among several computers for processing. In this way, data is partitioned, and parallel processing occurs across several servers. The setup is perfect for batch processing, which usually occurs offline. Although the cost of horizontal scaling is significantly cheaper, software is required to allow the distribution of data and prevent any faults. Typical examples of this software would include Hadoop and Spark.

At the core of these functions lies the Map-reduce algorithm, which is the intuitive title description for first creating a mapping of the data and then reduce the mapper, so one can summarize the data and reduce the effective size for processing. The code for the processing can be found at this Github link.

2.2 The MapReduce algorithm

MapReduce functions with firstly partitioning a large dataset across a number of cluster servers, which are called nodes. The nodes create an apportioned sized dataset and create key, value pairs of the data. Creation of key and value pairs is the mapper part of the algorithm. In the case of sales data, you may want to count the number of times a customer has made sales. The customer ID in this case is the key and this will be mapped to a value of one. The data is then sorted amongst the clusters, each containing a list of key, value pairs. The reducer algorithm then iterates through the list and sums the number of values per key, and the output is a total count per customer ID.

2.3 Apache Spark and Python

In this specific scenario of customer sales data, Apache spark together with python was used as the processing platform for a large dataset. Initially, with some work, Hadoop was installed on the local computer using the Hortonworks cloud platform, but some difficulty arose with its python dependency. I then opted for the Microsoft Azure free trial which allows you to set up a cluster of machines, but this request had an incurred cost, and it was decided not to go down this route. However, azure platform has a number of key features to allow MapReduce and perform machine learning algorithms with an output of great visualisations, all within the Microsoft Azure hub.

Apaches Spark is great tool to initialise on your local computer. There was some difficulty in the initial installation, but once up and running it was much less prone to errors and quickly carries out mapreduce functionality. Since we are developing Spark code with Python, the anaconda distribution was used, since this was already installed. It does not matter what python environment is used, as the spark code will be run from the command line. The next requirement was a java runtime environment installation, since Spark runs on top of Scala and Scala runs on the java environment. The Java development Kit 8 was installed under the local C drive with a path modification for compatibility with this spark version. This will also install the java runtime environment. Finally, a prebuilt version of Apache Spark 3.2.0 was downloaded and extracted to a folder on the C drive. A configuration file under Spark was then modified to prevent the occurrence of error logs. The user environmental and path variables were created so that all the relevant software can be located. Spark is considered to be faster than Hadoop's MapReduce by using a directed acyclic graph engine. It uses this tool to find the optimal way to distribute and process the task at hand[1].

Spark can run on different cluster managers besides its own built-in cluster managers. Cluster manager can distribute work to executors or different machines from whatever is received from the driver programs. It also allows fault tolerance if one of the executors goes down, rather than terminating the entire job. Spark in this way is scalable to large datasets, primarily in a horizontal fashion. Scala is the most popular language with Spark, but python is a little easier to startup and run Mapreduce operations. Python and Scala have similar syntax appearance in Spark, so it is fairly easy to migrate to scala once familiarity is attained.

2.4 RDD and Spark Dataframes

Spark uses an RDD or Resilient Distributed Dataset to process the data. The spark shell creates a spark context object which allows you to create an RDD. The RDD has several methods that allow for transformations and these include map, flatmap, filter, distinct. The RDD methods accept function parameters such as python's lambda function to create key and value pairs. Once the transformation is done such as mapping you can then perform actions on this using action methods such as reduce, count, take, top etc.

Spark extends the RDD to another structured format called a dataframe which is more applicable to structured data formats. A spark session is imported and using the getOrCreate method where the database view is created. This spark sql session allows you to perform sql commands on your data across the cluster. You can also perform methods directly on the dataframe. These include filter, groupBy, select etc

A spark dataframe was used for the cosmetics dataset, as the time for data processing was five times less than an RDD object. Once a Spark session was created, Spark then infers a schema of the csv file that has been parsed to it. We then filter the dataframe for only purchases, as that is the relevant piece of information required. Only three specific columns were selected from the dataframe, and these include "event_time, "user_id" and "price". The date was reformatted to a datetime object and the time was stripped from the date object. The mapper and reducer was then performed using a sql method of groupBy where the reducing was done by three operations. The sum was to determine the total amount spent by a user, the count is the number of single purchases made by the user and finally the max function determine the most recent purchase made by the user. The columns were renamed and the dataframe was saved as a csv to be processed for visualisation. All scripts are submitted via spark through the command line interface.

2.5 Conclusion

The final dataset was reduced from 1Gb to approximately 2Mb in size. MapReduce has great functionality, especially for machine learning where large datasets need to be processed. The ideal solution would still be cloud solutions such as Microsoft's Azure or Amazon's Elastic MapReduce. Given the appropriate resources, these will definitely be explored in the future. Frank Kane has great content[1] for an extension of Apache Spark, using Azure's Elastic MapReduce as a cloud-based solution.

3 Visualisation

3.1 Introduction

The mapper and reducer have now outputted key information required for the next phase of analysis and visualisation. There are some basic plots that one can achieve through the information received that highlight very high-level dashboard results. This could give some limitation in making data-driven decisions for the business as a whole. Another option is to scale the relevant fields of Monetary, Frequency and Recency to highlight key segmented groups and then have several approaches to retain or expand customer growth. Finally, an unsupervised algorithm could provide a lot more insight into key customers, and this may be the optimal approach when targeting consumers. All programming code can be found on this Github page.

3.2 Bar Plots

Bar plots render some basic and important information. The specific visual in figure 1 was created using python where the the original output dataframe was sorted according to the highest to lowest monetary spend by all customers. In the context of the business it does share relevant information of what key customers to focus on, however if one delves into the depth of all customer activity, monetary value alone will not justify key focus areas for the business.

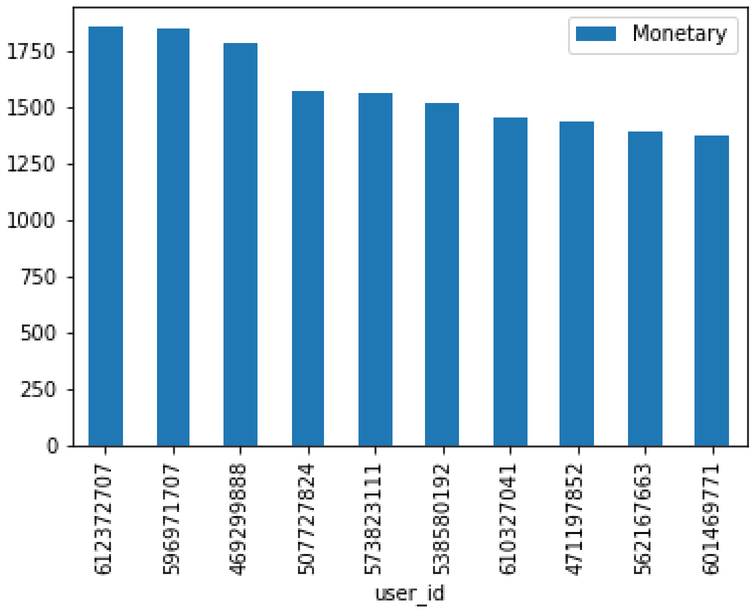

Figure 1: Basic bar plot highlighting the top 10 customers with regard to purchase value

In figure 2 the frequency or count of purchase is then considered as another factor a business could consider as important. Very similar to the layout in figure 1, frequent customers may require some

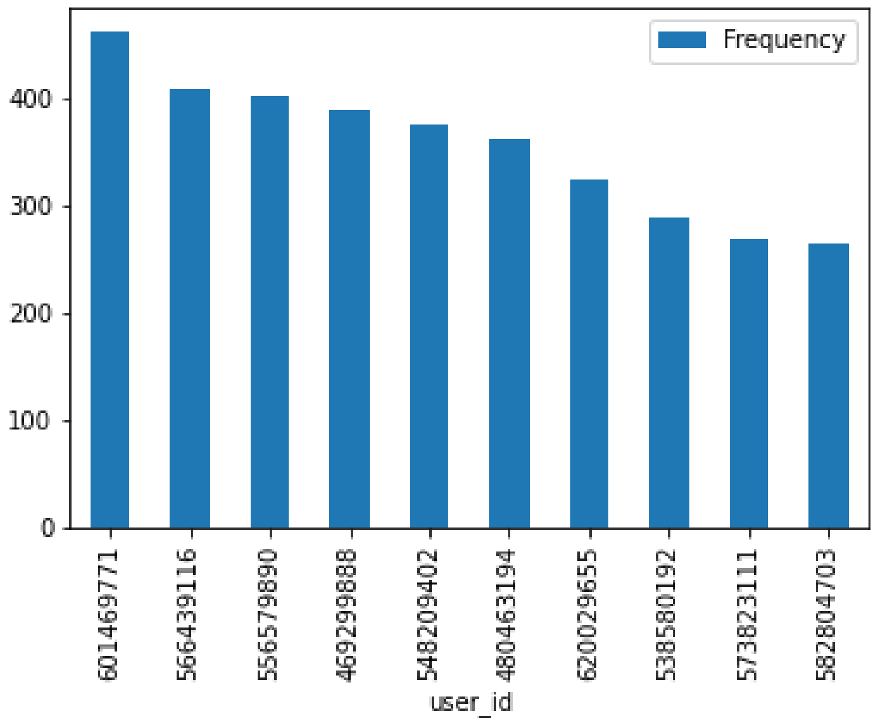

Figure 2: Basic bar plot highlighting the top 10 customers with regard to purchase count

sort of additional product information to possibly expand or increase their purchase on higher value items. This insight cannot be determined on frequency or monetary alone but requires a combination of a few variables to make a conclusive decision on retention or promotion. We are also limited with the number of only 10 customers which is a fraction of the total customer base. A combination of these variables may offer some better insight.

3.3 Customer Tree Map

A summary of features received from the MapReduce algorithm can be combined to perform customer segmentation. Each variable such as recency, monetary and frequency can be split into quantiles using pandas qcut method and then assigned a label range from 1 to 4. In the case of monetary of frequency, a higher value will yield a higher category label, whereas recency with a lower value meaning that a purchase was made more recently will receive a higher category value. The product of these ratings will give an overall score to the customer user_id. The higher the score is indicative that these customers are critical to the business and might not require constant attention but perhaps a bulk discount, or membership to some loyalty programs. Those that are on the lower end of the score range might require more attention or being reminded of the brand by perhaps targeted advertising or a reminder of items in their cart or similar cheaper alternatives available.

The categories listed in the tree map visual of figure 3 were an equal split between the minimum and maximum range of the RFM score, where a classification group was assigned. The only drawback of this approach is that it might not be an optimal division of customers but rather a discretionary decision. A more appropriate avenue would be an unsupervised clustering algorithm that can dictate key customer groups based on the optimal segregation of data points.

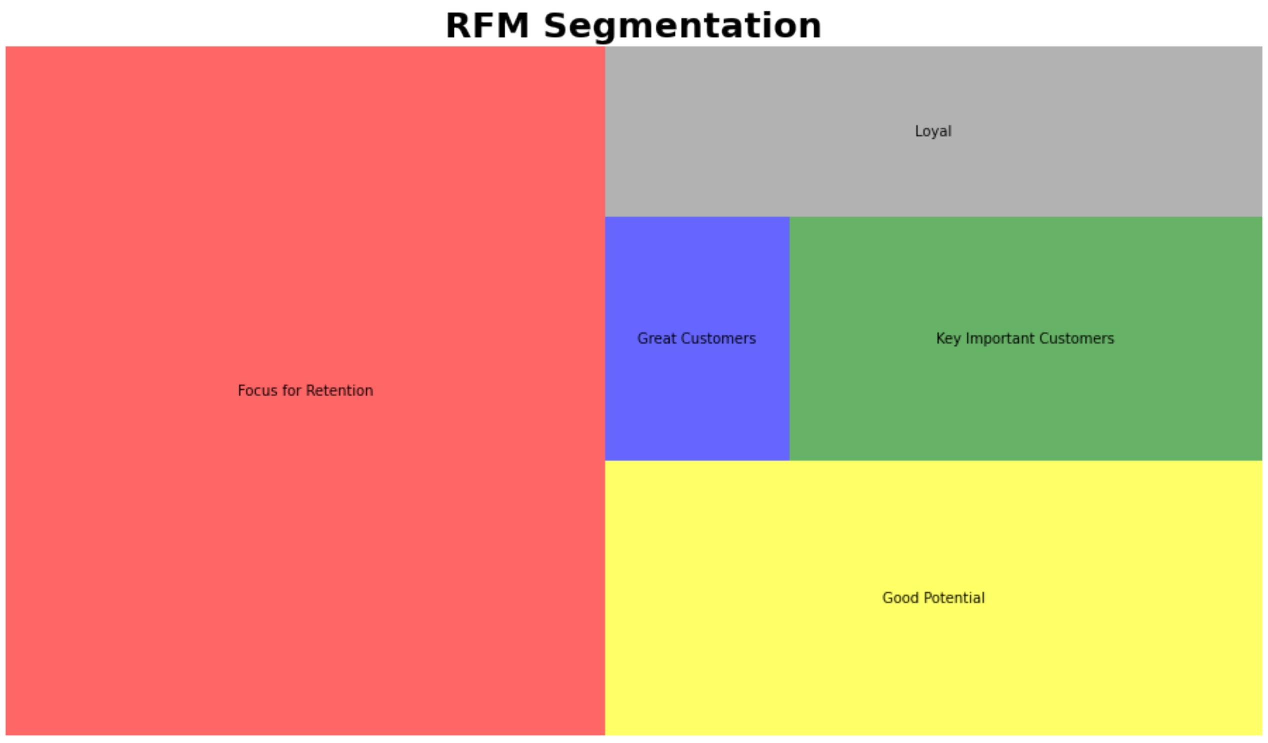

Figure 3: Tree map of five customer segmented groups based on the RFM score

3.4 Kmeans Clustering

In brief, the kmeans clustering algorithm is an unsupervised learning algorithm to determine the optimum number of clusters of a dataset. The within cluster sum of squares was used as a measurement. Individual cluster scores are determined by squaring the distance between each data point and the centroid within a cluster, then summing this up. Once the individual score is determined, all scores are summed up to determine the final WCSS score. This is done iteratively from a cluster number of sizes 1 to 10.

As the cluster number increases, the distance between centroids and each point within the cluster also decreases, effectively reducing the score. The WCSS will finally equate to zero if each individual point is a cluster on its own. The graph in figure 4 is a chart of the cluster number versus the WCSS score. A reasonable cluster number is when there is a significant drop in the score, but any reasonable

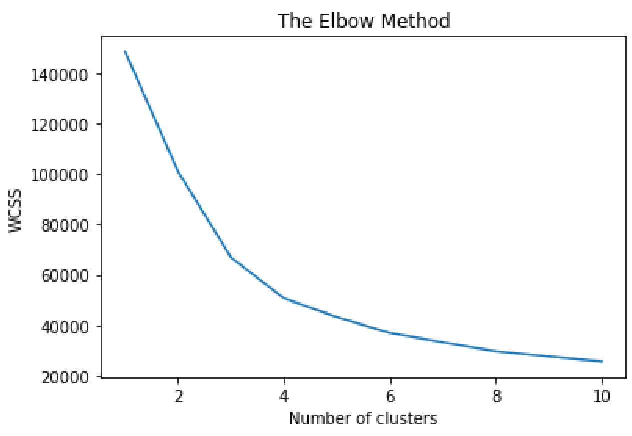

Figure 4: Elbow plot in choosing an optimal number of clusters

score reduction past this point will give a sub-optimal clustering solution. The distinct point where the cluster number selection is made is called the elbow. The elbow in figure 4 is at approximately 4 clusters. Four clusters were chosen for the solution when optimially segregating the three cosmetics sales data features.

The display of clustering can be done using 2 variables at a time using matplot library's scatter function as depicted in figure 5, however a comparison should be made with all three variables.

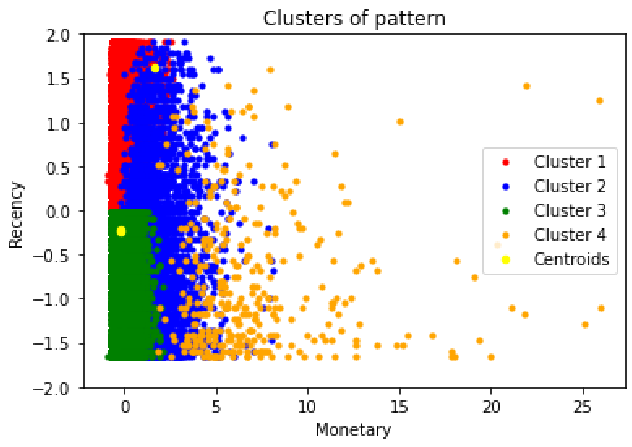

Figure 5: Two dimensional scatter plot comparing 2 features

In figure 6 plotly's 'go' library allows a 3d plot which can be animated in jupyter notebooks for a clear zoomed in visualisation where each datapoint label can be looked at for a reference. The three variables however still require comparison using a few sub-plots. This is where the seaborn library serves as powerful tool for pair-wise plots and comparison of dimensions with 2 dimensional subplots.

In figure 7 cluster "0" which makes up a larger proper of the samples have a high recency, indicating purchases were made a while back. They are also less frequent customers, spending marginally less

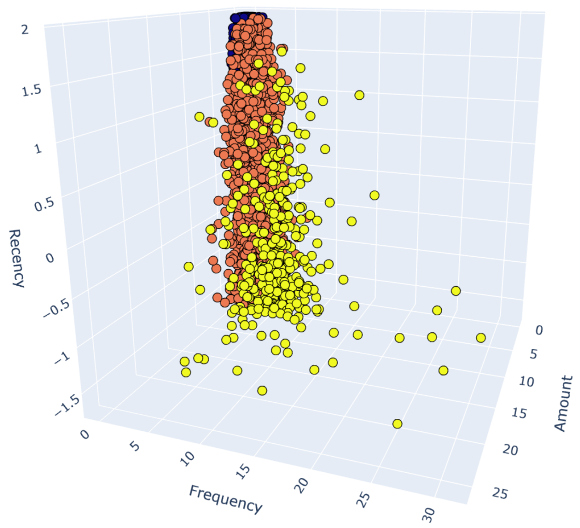

Figure 6: Three dimensional cluster plot that allows rotation and animated viewing of all clusters

than the other groups. These can be classified as a focus group that may require special attention to prevent the quick churn rate.

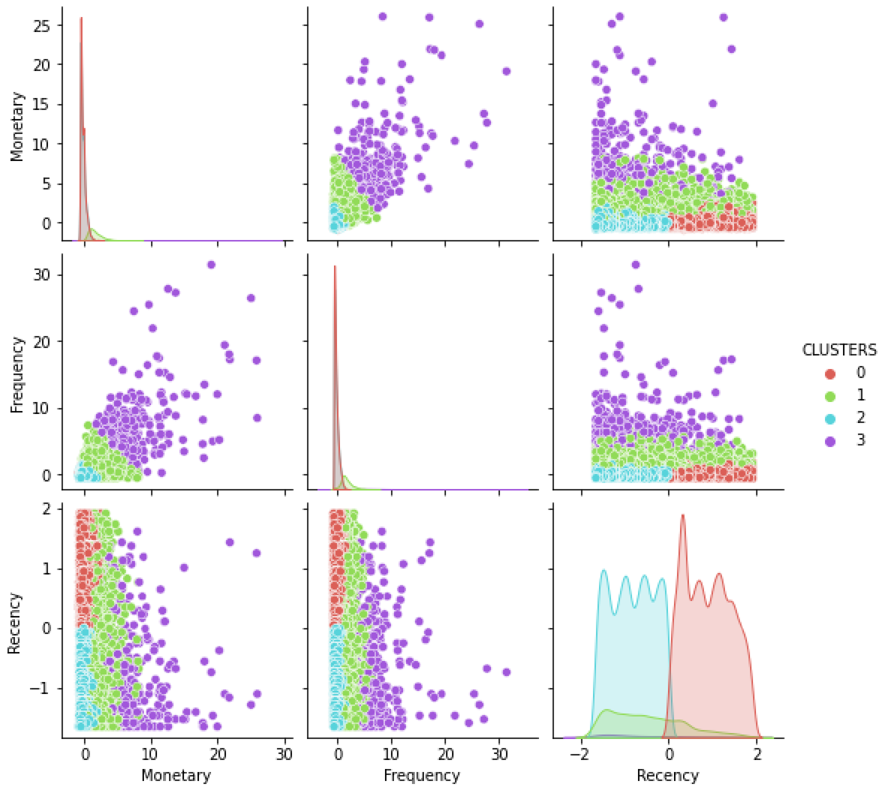

Figure 7: Python's Seaborn library pairwise plot segmenting the four clusters

Cluster "2" which also contributes to a fairly large sample of 21366 customers has a low recency, low frequency and as well as a low monetary expenditure. These could be new customers, and promotional advertising may interest them with a similar product idea to what they have already purchased. Reminders of repetitive items can be a form of targeted advertising via messaging or emails. Cluster "1" and "3" have a low recency and high monetary and frequency values, and these are loyal top-tier customers who should benefit from loyalty programmes and bulk discounts due to frequent purchases.

The effective grouping of customers can then be summarized in the pie chart in figure 8. Given the categorization, the marketing team may design the appropriate marketing tools to retain and attract customers going forward.

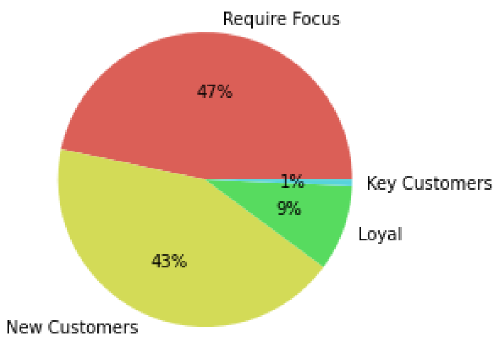

Figure 8: Pie Chart of segmented customers by cluster count

3.5 Conclusion

The simplistic bar plots are useful for a high-level view of the data but lack deep insight on how to promote marketing campaigns and improve decision-making. The tree map was used with categories of mapping a range of values to the individual fields of recency, monetary and frequency. Although this provides useful information, it may not be an optimal approach and this can lead to an unnecessary marketing expense by targeting an incorrectly placed segment. Clustering by kmeans proves to be good machine learning tool for proper market segregation. Further development can be done by looking at the product range purchased within each segment and then promote products based on clustering of the products.

References

[1] Sundog Education by Frank Kane, Frank Kane, and Sundog Education Team. Taming big data with apache spark and python - hands on! `https://www.udemy.com/course/taming-big-data-with-apache-spark-hands-on/`.

[2] Michael Kechinov. ecommerce events history in cosmetics shop, Mar 2020. `https://www.kaggle.com/mkechinov/ecommerce-events-history-in-cosmetics-shop?select=2020-Jan.csv`.