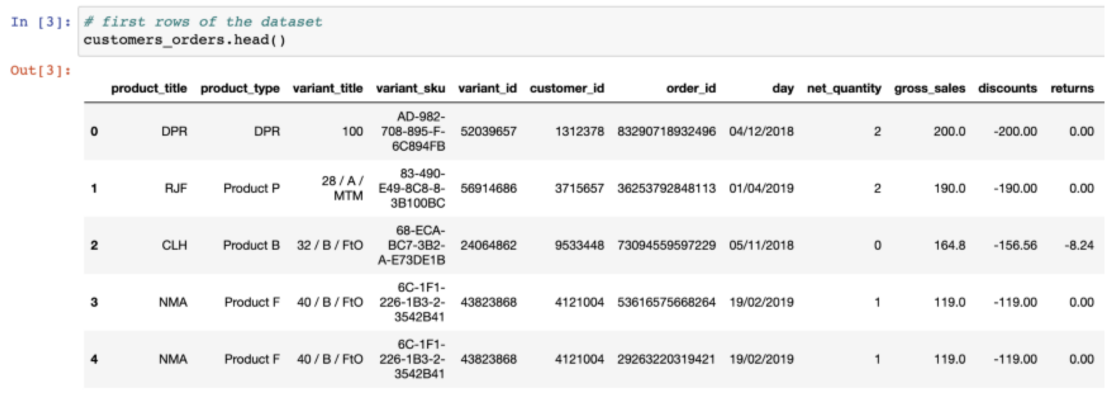
1. Business Case

In the case study, I visualized the customer behaviour and characteristics from diverse aspects. Taking it one step further, I will form the business case around the question: **Can the customer base be grouped to develop customized relationships?**

I will approach this question from a behavioural aspect (alternatives can be geographical or demographical perspectives) to better understand customers' spending and ordering habits with the following features: **Number of products ordered, average return rate and total spending.**

2. Data Preparation

There are approximately 25000 unique customers combined with their order information in the raw dataset:



In [3]:
```
# first rows of the dataset
customers_orders.head()
```

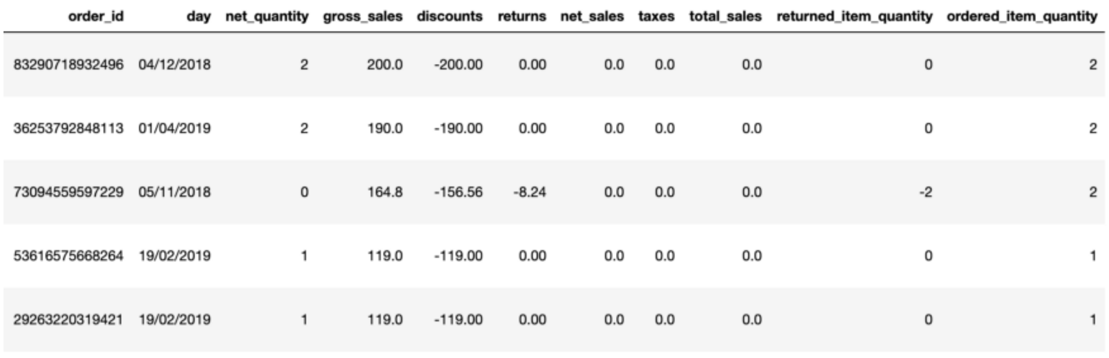
Out[3]:

	product_title	product_type	variant_title	variant_sku	variant_id	customer_id	order_id	day	net_quantity	gross_sales	discounts	returns
0	DPR	DPR	100	AD-982-708-895-F-6C894FB	52039657	1312378	83290718932496	04/12/2018	2	200.0	-200.00	0.00
1	RJF	Product P	28 / A / MTM	83-490-E49-8C8-8-3B100BC	56914686	3715657	36253792848113	01/04/2019	2	190.0	-190.00	0.00
2	CLH	Product B	32 / B / FtO	68-ECA-BC7-3B2-A-E73DE1B	24064862	9533448	73094559597229	05/11/2018	0	164.8	-156.56	-8.24
3	NMA	Product F	40 / B / FtO	6C-1F1-226-1B3-2-3542B41	43823868	4121004	53616575668264	19/02/2019	1	119.0	-119.00	0.00
4	NMA	Product F	40 / B / FtO	6C-1F1-226-1B3-2-3542B41	43823868	4121004	29263220319421	19/02/2019	1	119.0	-119.00	0.00

Dataset is well-formatted and had no NA values. So, we can start by forming the features. 3 features will be calculated per customer_id and they will help us with the visualization (using Plotly library) and algorithm explainability in the latter steps. Data preparation will be done with pandas and numpy.

- **Number of products ordered:** It is calculated by counting the product_type ordered by a customer with the below function:

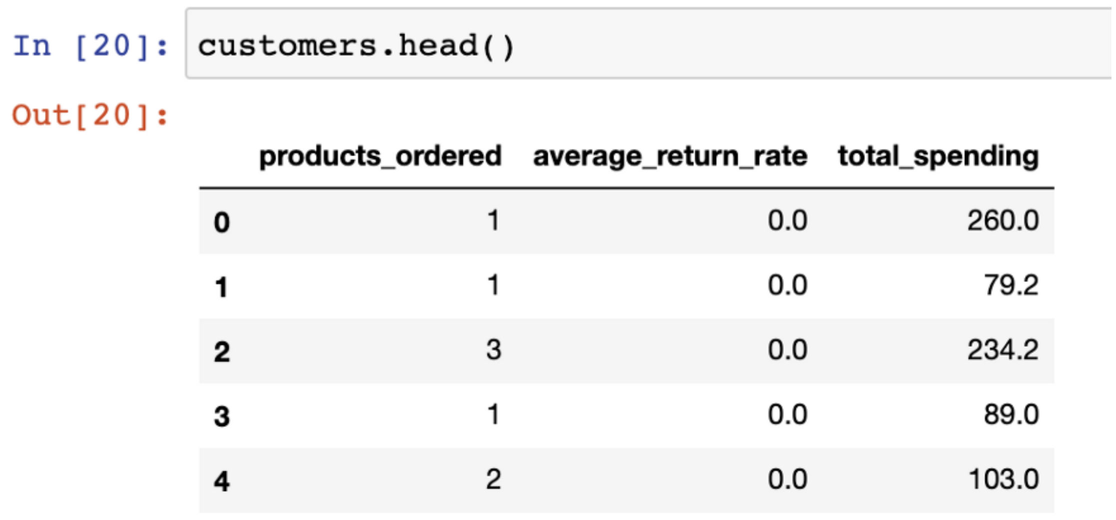
- **Average return rate:** It is the ratio of returned_item_quantity to the ordered_item_quantity averaged for all orders of a customer.



order_id	day	net_quantity	gross_sales	discounts	returns	net_sales	taxes	total_sales	returned_item_quantity	ordered_item_quantity
83290718932496	04/12/2018	2	200.0	-200.00	0.00	0.0	0.0	0.0	0	2
36253792848113	01/04/2019	2	190.0	-190.00	0.00	0.0	0.0	0.0	0	2
73094559597229	05/11/2018	0	164.8	-156.56	-8.24	0.0	0.0	0.0	-2	2
53616575668264	19/02/2019	1	119.0	-119.00	0.00	0.0	0.0	0.0	0	1
29263220319421	19/02/2019	1	119.0	-119.00	0.00	0.0	0.0	0.0	0	1

- **Total spending:** It is the aggregated sum of total sales, which is the final amount after taxes and returns.

After the calculations, 3 features merged in the customers data frame:

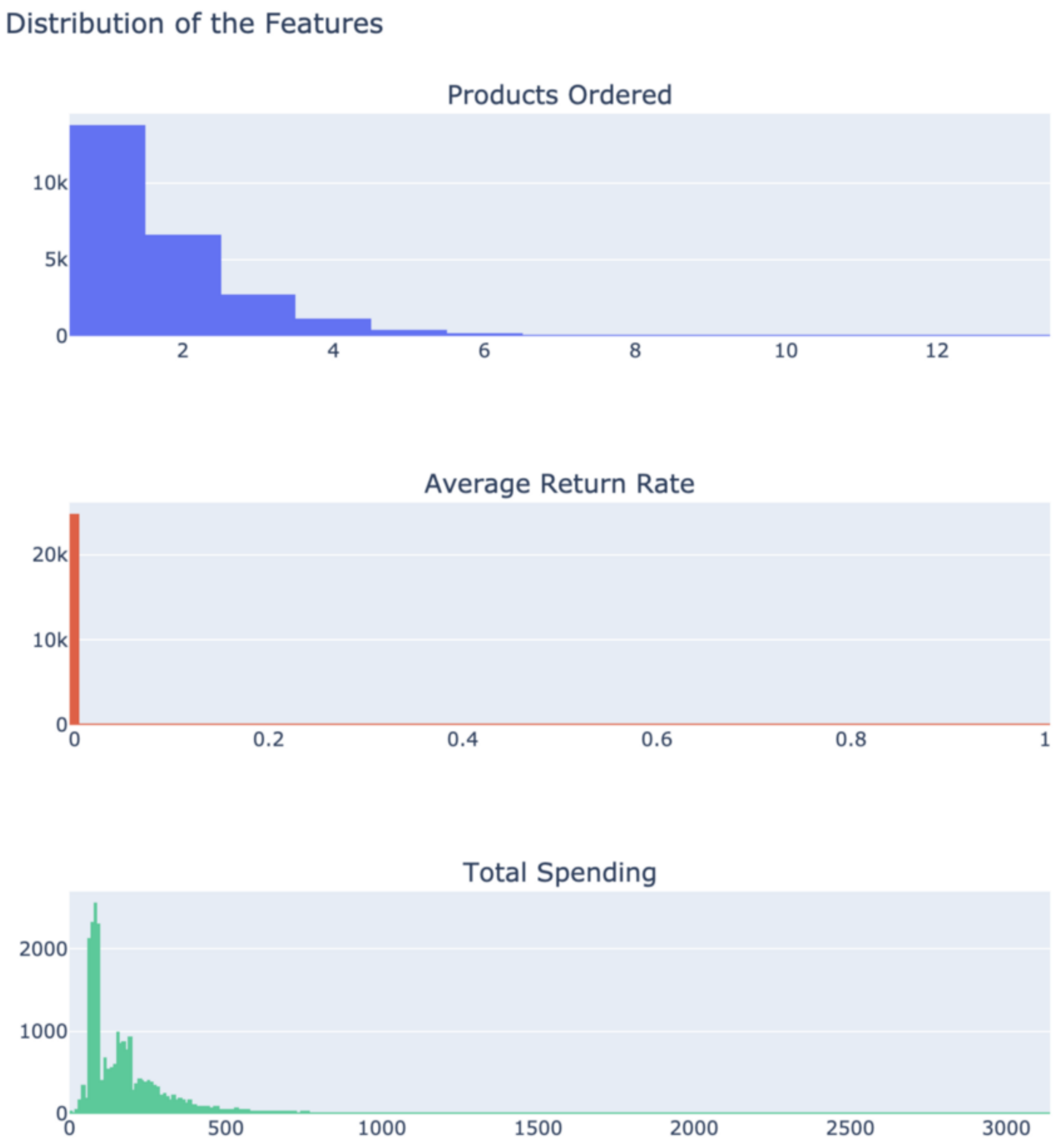


```
In [20]: customers.head()
```

```
Out[20]:
```

	products_ordered	average_return_rate	total_spending
0	1	0.0	260.0
1	1	0.0	79.2
2	3	0.0	234.2
3	1	0.0	89.0
4	2	0.0	103.0

Let’s have a look at the individual distribution of the features:



All 3 distributions are positively. Products ordered shows a power-law distribution and average return rate of 99% of the customers are 0.

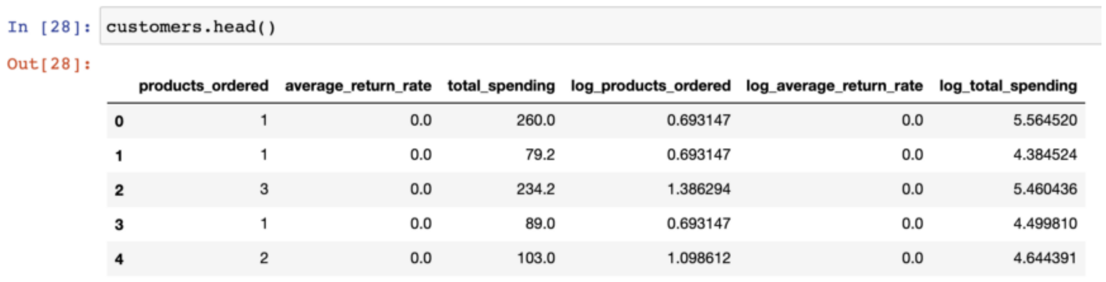
3 features have different ranges varying between [1, 13], [0, 1] and [0, 1000] which is an important observation showing that features need scaling!

Scaling:

K-means algorithm interprets each row in the customers data frame as a point in a 3-dimensional space.

For K-means to perform effectively, we are going to scale the data using logarithmic transformation which is a suitable transformation for skewed data. This will scale down proportionally the 3D space which our data is spread, yet preserving the proximity between the points.

After applying the above function, `customers` **data frame is ready to be fed into K-means clustering:**



```
In [28]: customers.head()
```

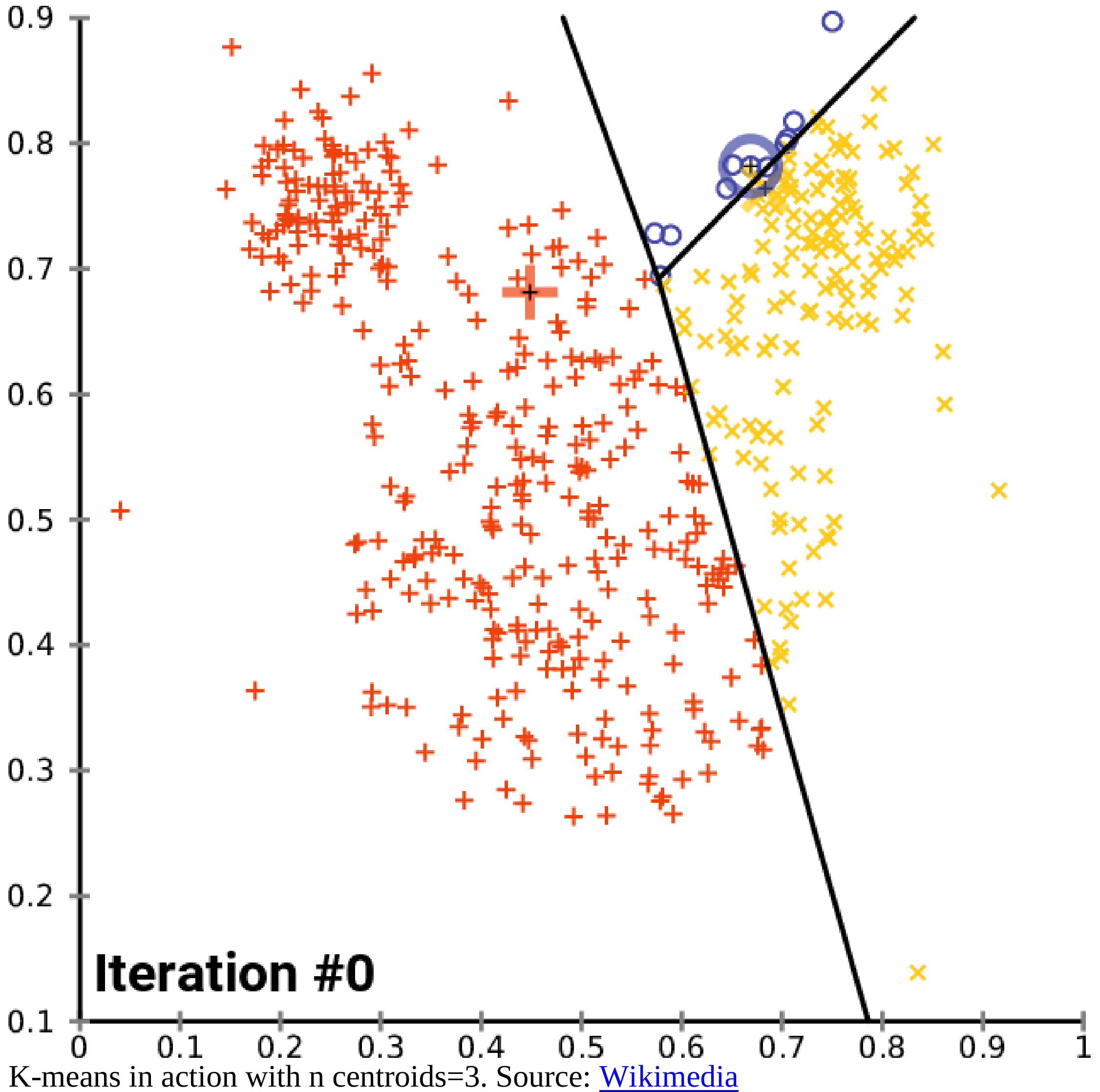
Out[28]:

	products_ordered	average_return_rate	total_spending	log_products_ordered	log_average_return_rate	log_total_spending
0	1	0.0	260.0	0.693147	0.0	5.564520
1	1	0.0	79.2	0.693147	0.0	4.384524
2	3	0.0	234.2	1.386294	0.0	5.460436
3	1	0.0	89.0	0.693147	0.0	4.499810
4	2	0.0	103.0	1.098612	0.0	4.644391

3. Segmentation with K-means Clustering

We are going to use K-means algorithm from scikit-learn. Let's first understand how the algorithm will form customer groups:

1. Initialize *k=n centroids=number-of-clusters* randomly or smartly
2. Assign each data point to the closest centroid based on euclidian distance, thus forming the groups
3. Move centers to the average of all points in the cluster



K-means in action with n centroids=3. Source: [Wikimedia](https://commons.wikimedia.org)

While running the steps through, the algorithm checks the sum of squared distances between clustered-point and center for each cluster. Mathematically speaking, it tries to minimize — optimize the ***within-cluster sum-of-squared-distances*** or ***inertia*** of each cluster**.**

$$\sum_{i=0}^{n} \min_{\mu_j \in C} (||x_i - \mu_j||^2)$$

Mathematical expression of within-cluster sum-of-squared-distances or inertia where *X is the points in the cluster and μ is the current centroid*

When ***inertia*** value does not minimize further, algorithm converges. Thus, iteration stops.

```
from sklearn.cluster import Kmeans
kmeans_model = KMeans(init='k-means++',
                    max_iter=500,
                    random_state=42)
```

- `init` parameter with the `k-means++` allows the algorithm to place initial centers smartly, rather than random.
- `max_iter` is the maximum number of iterations of the algorithm in a single run, default value is 300.
- `random_state` guarantees the reproducibility of the model results.

This algorithm is easy to understand, fits well to large datasets in terms of computing times and guarantees convergence. However, when centroids are initialized randomly, algorithm may not assign the points to the groups in the most optimal way.

One important consideration is the selection of *k*. In other words, how many groups should be formed? For example, K-means applied above uses *k=8* as a default value.

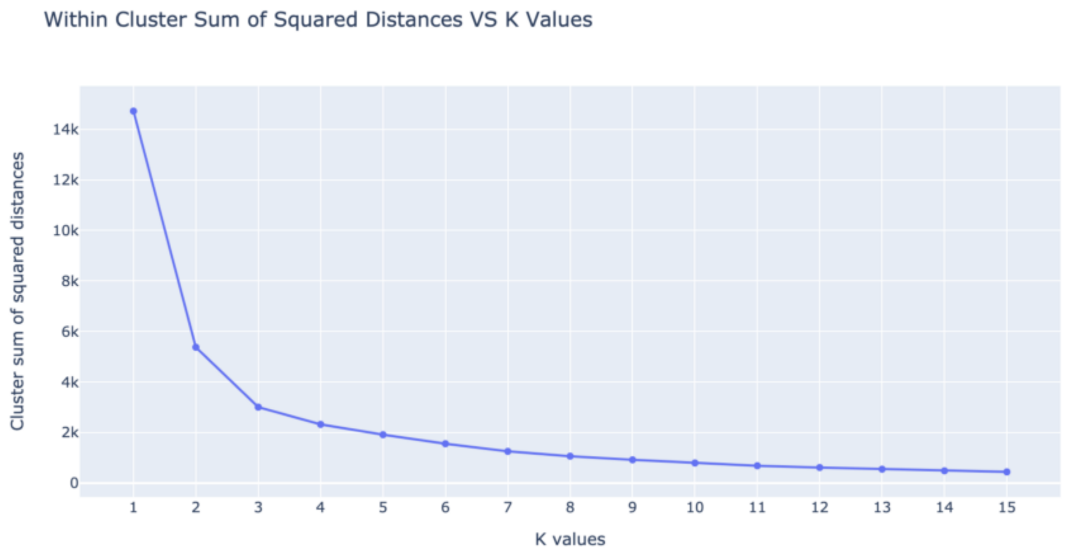
In the next step, we are going to choose *k* which is the most important hyperparameter of K-means.

4. Hyperparameter Tuning

We are going to build different K-means models with *k* values 1 to 15, and save the corresponding *inertia* values.

```
results = make_list_of_K(15, customers.iloc[:,3:])
k_values_distances = pd.DataFrame({"clusters": clusters,
                              "within cluster sum of squared distances": results})
```

When we plot *inertia* against the *k* values:



With the elbow method, we are going to select the *k* value where the decrease in the inertia stabilizes.

When *k=1* inertia is at the highest, meaning data is not grouped yet. Inertia decreases steeply until *k=2*. Between *k=2* and *4*, the curve continues to decrease fast.

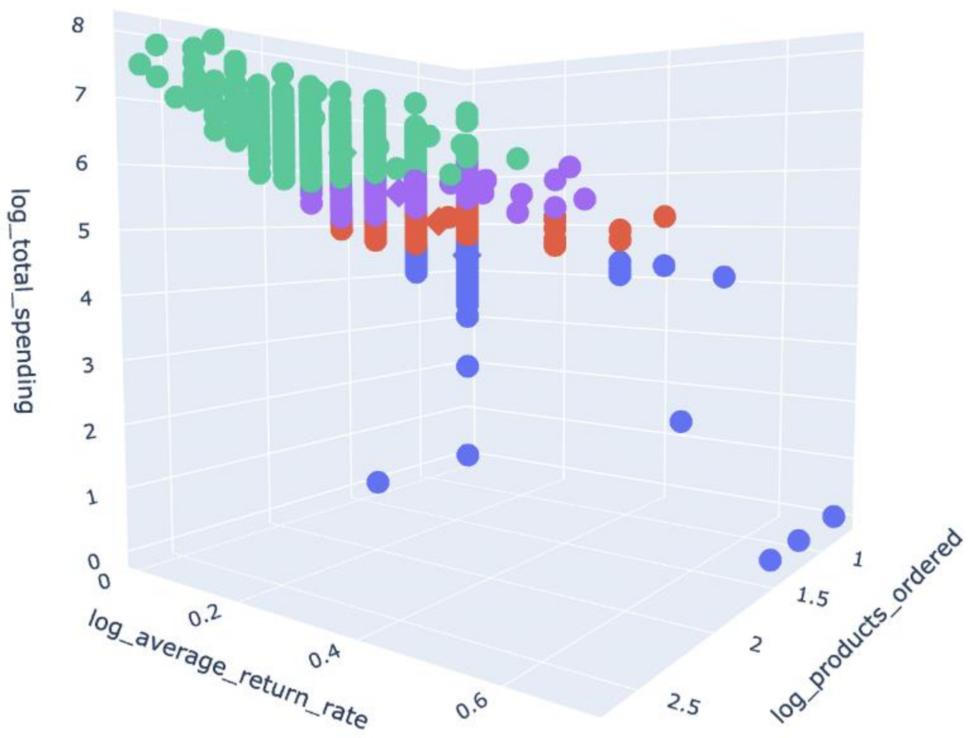
At *k=4*, the descent stabilizes and continues linearly afterwards, forming an elbow at *k=4*. This points out the optimal number of customer group is *4*.

5. Visualization and Interpretation of the Results

Let's plug in the *k=4* to K-means and visualize how customer groups are created:

```
# create clustering model with optimal k=4
updated_kmeans_model = KMeans(n_clusters = 4,
                        init='k-means++',
                        max_iter=500,

random_state=42)updated_kmeans_model.fit_predict(customers.iloc[:,3:])
```



Data points are shown in spheres and centroids of each group are shown with cubes. 4 customer groups are as follows:

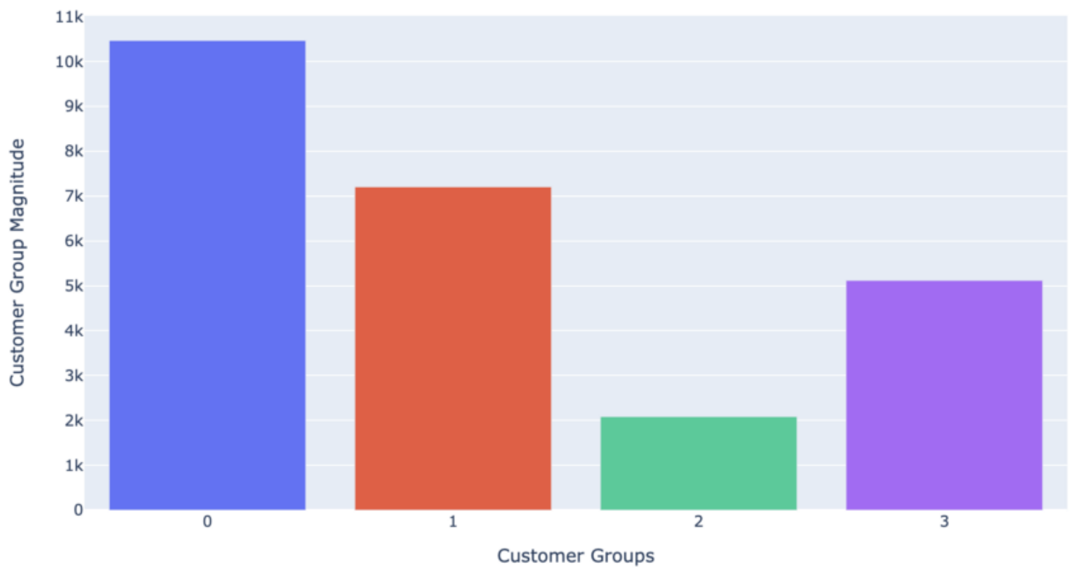
Blue: Customers who ordered at least one product, with maximum total spending of 100 and having the highest average return rate. They might be the newcomers of the e-commerce website.

Red: Customers who ordered 1 to 4 products, with average total spending of 150 and a maximum return rate of 0.5.

Purple: Customers who ordered 1 to 4 products, with average total spending of 300 and a maximum return rate of 0.5.

Green: Customers who ordered 1 to 13 products, with average total spending of 600 and average return rate as 0. It makes the most favourable customer group for the company.

Let’s look at how many customers are there in each group — known as cluster magnitudes:



The overall strategy would be preserving the most favourable customer group — the green one — while moving the blue customer group to the red and purple areas.

Blue group is 42% of all customers, any improvements achieved in this customer group will dramatically increase the revenue. Eliminating high return rates and offering gift cards can move this customer group to low-average-return-rate and high-total-spending area. If we assume that they are newcomers, gift cards can expedite their come-back.

Red and purple group together consists of 50% of all customers. They are showing the same characteristics from the average return rate and products ordered perspectives but differ from total spending. These groups can be defined as who already know the brand and orders multiple products. Those customers can be kept up-to-date with the brand with some specialized communications and discounts.

Green customer group consists of 8% of all customers, forming the most favourable customer group for the brand. They order multiple products and they are highly likely to keep them. To maintain and possibly expand this group, special deals and pre-product launches might help. Moreover, they can be magnets for new customers impacting the expansion of the customer base.

Conclusion

We approached customer segmentation problem from a behavioural aspect with the number of products ordered, average return rate and total spending for each customer. Use of 3 features helped us with the understandability and visualization of the model.

All in all, the dataset was apt to perform an unsupervised machine learning problem. At first, we only had customers data with order information and did not know if they belonged to any group. With the K-means clustering, patterns in the data were found and extended further into groups. We carved out strategies for the formed groups, making meaning out of a dataset that is a dust cloud initially.