

A report on

**CUSTOMER SEGMENTATION ML PROJECT**

**Under the guidance of mentor:**

**AISHWARYA SINGH**

* **Name of Internee:** Rajveer Singh
* **Technology:** Python with ml
* **College:** Lucknow University F.O.E

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **N0.** | | | **TOPIC** | | **PAGE NO.** |
| **1** | | | **INTRODUCTION** | | **3** |
| **2** | | | **OBJECTIVE** | | **4** |
| **3** | | | **BACKGROUND** | | **5** |
| **4** | | | **HARDWARE AND SOFTWARE REQUIREMENTS** | | **9** |
| **5** | | | **CODING** | | **10** |
| **6** | | | **OUTPUT SCREEN SHOTS** | | **14** |
| **7** | | | **FUTURE SCOPE** | | **16** |
| **8** | | **CONCLUSION** | | | **17** |
| **9** | **BIBLIOGRAPHY AND REFRENCES** | | | **17** | |

**INTRODUCTION**

**Customer segmentation**

Customer segmentation is the practice of dividing a customer base into groups of individuals that are similar in specific ways relevant to marketing, such as age, gender, interests and spending habits**.**

When you perform customer segmentation, you find similar characteristics in each customer’s behavior and needs. Then, those are generalized into groups to satisfy demands with various strategies. Moreover, those strategies can be an input of the

* Targeted marketing activities to specific groups
* Launch of features aligning with the customer demand
* Development of the product roadmap

Photo source: Costumer purchasing groceries



**OBJECTIVE**

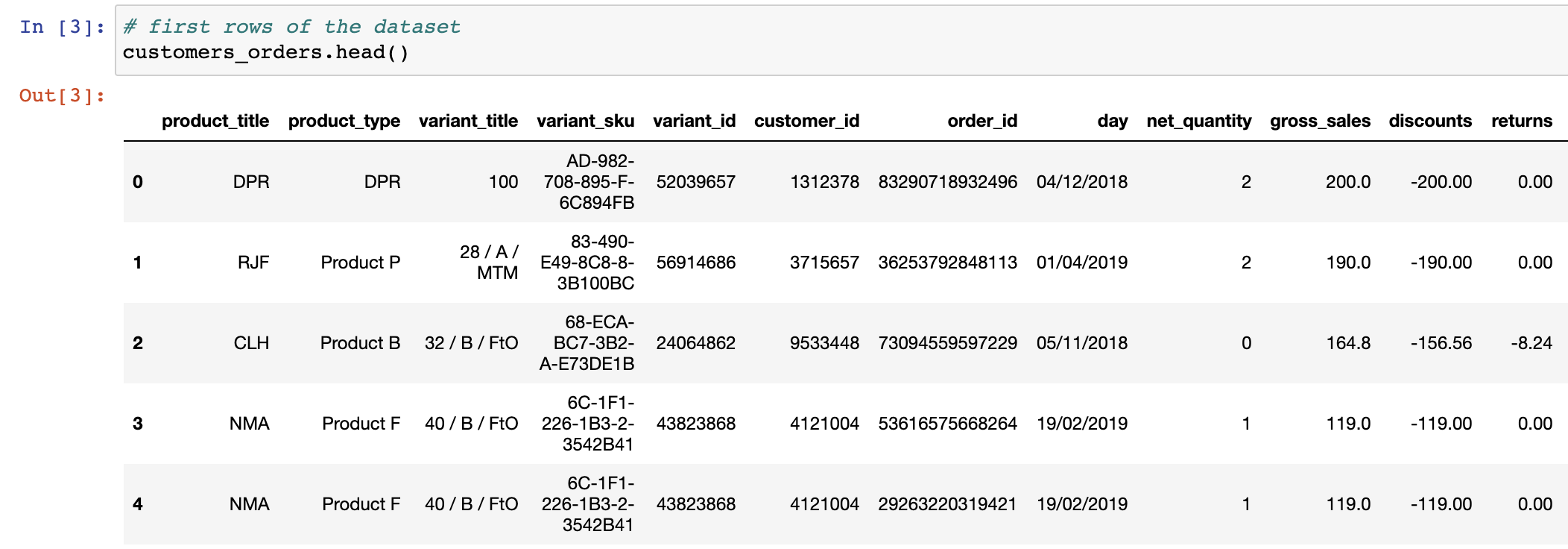
* I will apply an [unsupervised machine learning algorithm](https://en.wikipedia.org/wiki/Unsupervised_learning) with Python.
* Load dataset from November 2018 — April 2019 is actual sales data courtesy of an e-commerce company for case study.
* I will apply [K-Means clustering](https://en.wikipedia.org/wiki/K-means_clustering) to the dataset with the following steps.
* Business Case
* Data Preparation
* Segmentation with K-means Clustering
* Hyper parameter Tuning
* Visualization and Interpretation of the Results
* I will explain how K-means clustering works. Eventually, I will provide specific strategies for the segments formed.

**BACKGROUND**

**PROCEDURE:**

* I visualized the customer behavior 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 behavioral 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.**
* DATA PREPARATION:

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



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](https://plot.ly/python/) library) and algorithm explainability in the latter steps. Data preparation will be done with [pandas](https://pandas.pydata.org/) and [numpy](https://numpy.org/).

* **Number of products ordered:** It is calculated by counting the product\_type ordered by a customer with the below function:

|  |
| --- |
| def encode\_column(column): |
|  | if column > 0: |
|  | return 1 |
|  | if column <= 0: |
|  | return 0 |
|  |  |
|  |  |
|  | def aggregate\_by\_ordered\_quantity(dataframe, column\_list): |
|  | '''this function: |
|  | 1. aggregates a given dataframe by column list, |
|  | as a result creates a aggregated dataframe by counting the ordered item quantities |
|  |  |
|  | 2. adds number\_of\_X ordered where X is the second element in the column\_list |
|  | to the aggregated dataframe by encoding ordered items into 1 |
|  |  |
|  | 3. creates final dataframe containing information about |
|  | how many of X are ordered, based on the first element passed in the column list''' |
|  |  |
|  | aggregated\_dataframe = (dataframe |
|  | .groupby(column\_list) |
|  | .ordered\_item\_quantity.count() |
|  | .reset\_index()) |
|  |  |
|  | aggregated\_dataframe["products\_ordered"] = (aggregated\_dataframe |
|  | .ordered\_item\_quantity |
|  | .apply(encode\_column)) |
|  |  |
|  | final\_dataframe = (aggregated\_dataframe |
|  | .groupby(column\_list[0]) |
|  | .products\_ordered.sum() # aligned with the added column name |
|  | .reset\_index()) |
|  |  |
|  | return final\_dataframe |

* **Average return rate:**It is the ratio of returned\_item\_quantity to the ordered\_item\_quantity averaged for all orders of a customer.
* **Total spending:**It istheaggregated sum of total sales, which is the final amount after taxes and returns.
* **Let’s have a look at the individual distribution of the features:**



All 3 distributions are positively [skewed distributions](https://en.wikipedia.org/wiki/Skewness). 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. When grouping them, it uses the [euclidian distance](https://en.wikipedia.org/wiki/Euclidean_distance) between the data points and the center of the group. With highly varying ranges, algorithm may perform poorly and not be able to form the groups as expected.

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.

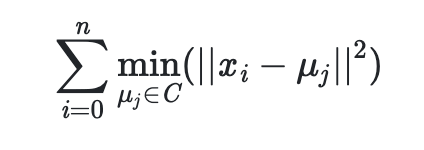
* **After applying the above function,**customers**data frame is ready to be fed into K-means clustering.**
* **Segmentation with K-means Clustering**
* We are going to use [K-means algorithm from scikit-learn](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html). Let’s first understand how the algorithm will form customer groups:
* Initialize *k*=*n centroids*=*number-of-clusters* randomly or smartly

Assign each data point to the closest centroid based on euclidian distance, thus forming the groups

Move centers to the average of all points in the cluster

Repeat steps 2 and 3 until [convergence](https://www.mathsisfun.com/definitions/converge.html).

* 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***.***

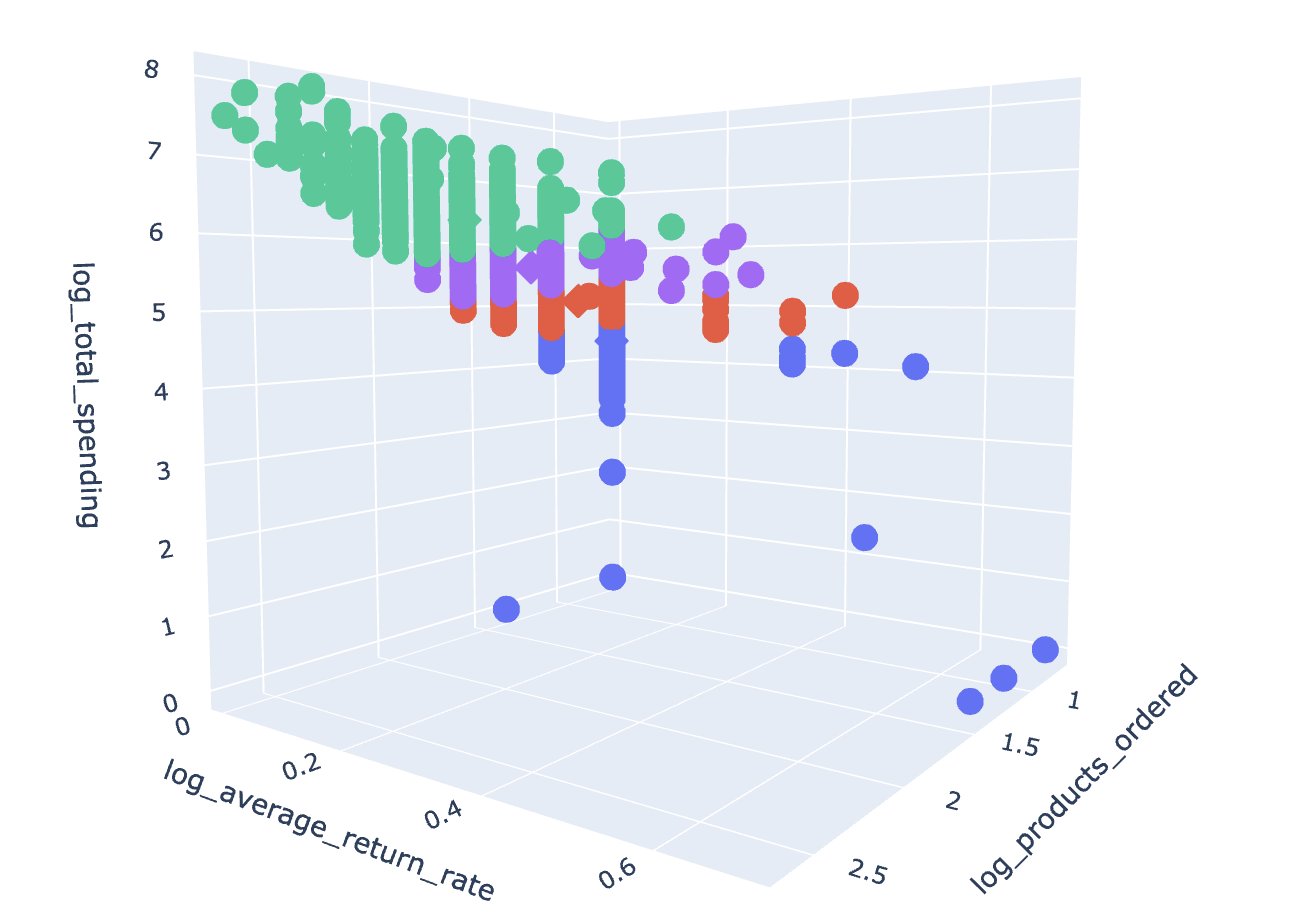


When ***inertia*** value does not minimize further, algorithm converges. Thus, iteration stops.

* **Hyperparameter Tuning**

While selecting *k,*we are going to decide against the optimization criteria of the K-means, inertia, using [elbow method](https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/). We are going to build different K-means models with *k* values 1 to 15, and save the corresponding *inertia* values.

* **Visualizing and interpretation of results:**



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

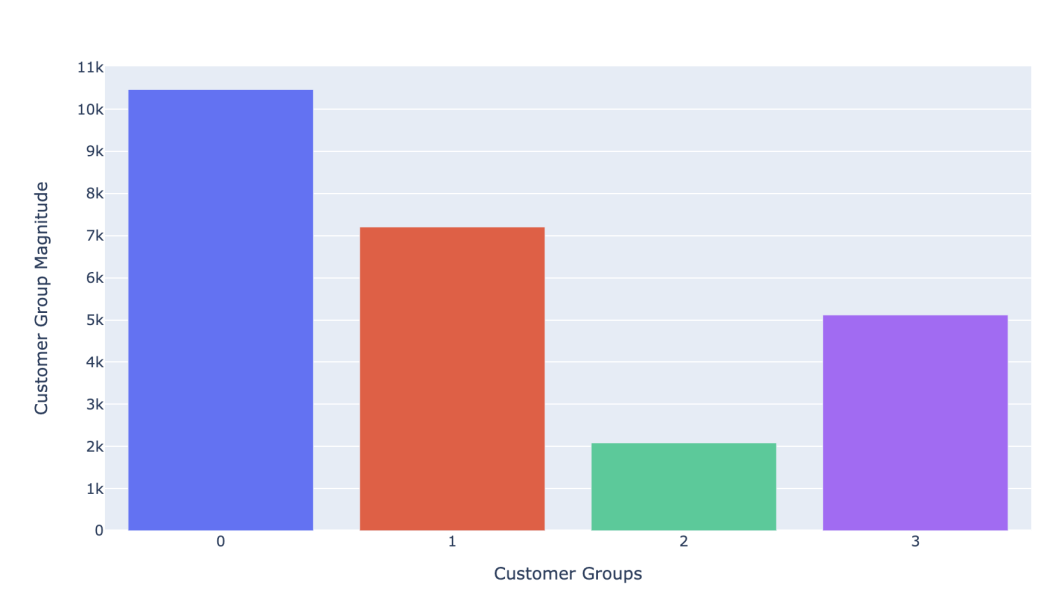
**Blue:** Customers who ordered at least one product, with maximum total spending of 100 and having the highest average return rate. They might be the newcomers of the e-commerce website.

**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:**



**HARDWARE AND SOFTWARE REQUIREMENTS**

**HARDWARE REQUIREMENTS:**

|  |  |
| --- | --- |
| Hardware tools | Minimum requirements |
| Processor | i5 or above |
| RAM | 4gb |
| Monitor | 17”colored |
| Mouse | Optical |
| Keyboard | 122keys |

**SOFTWARE REQUIREMENTS:**

|  |  |
| --- | --- |
| Software tools | Minimum requirements |
| Platform | Windows, linux or macos |
| Operating System | Windows, linux or macos |
| Technology | Machine learning-Python |
| Scripting Language | Python |
| IDE | Pycharm and Jupyter Notebook |

**Coding**

# data wrangling

import pandas as pd

import numpy as np

# visualization

import matplotlib.pyplot as plt

import plotly.express as px

import plotly.graph\_objects as go

from plotly.subplots import make\_subplots

# for data preprocessing and clustering

from sklearn.cluster import KMeans

%matplotlib inline

# to include graphs inline within the frontends next to code

%config InlineBackend.figure\_format='retina'

#to enable retina (high resolution) plots

pd.options.mode.chained\_assignment = None # to bypass warnings in various dataframe assignments

# load data into a dataframe

customers\_orders = pd.read\_csv("Orders - Analysis Task.csv")

# first rows of the dataset

customers\_orders.head()

# first glance of customers\_orders data

customers\_orders.info()

# descriptive statistics of the non-object columns

customers\_orders.describe()

print("Number of rows that net quantity is negative:",

customers\_orders[customers\_orders.net\_quantity < 0].shape[0])

*# exclude not sold/ordered SKUs from the dataset*

customers\_orders = customers\_orders[

customers\_orders["ordered\_item\_quantity"] > 0]

**1. Products ordered**

It is the count of the products ordered in product\_type column by a customer.

**Create functions to identify customers who order multiple products**

In [8]:

**def** encode\_column(column):

**if** column > 0:

**return** 1

**if** column <= 0:

**return** 0

**def** aggregate\_by\_ordered\_quantity(dataframe, column\_list):

*'''this function:*

*1. aggregates a given dataframe by column list,*

*as a result creates a aggregated dataframe by counting the ordered item quantities*

*2. adds number\_of\_X ordered where X is the second element in the column\_list*

*to the aggregated dataframe by encoding ordered items into 1*

*3. creates final dataframe containing information about*

*how many of X are ordered, based on the first element passed in the column list'''*

aggregated\_dataframe = (dataframe

.groupby(column\_list)

.ordered\_item\_quantity.count()

.reset\_index())

aggregated\_dataframe["products\_ordered"] = (aggregated\_dataframe

.ordered\_item\_quantity

.apply(encode\_column))

final\_dataframe = (aggregated\_dataframe

.groupby(column\_list[0])

.products\_ordered.sum() *# aligned with the added column name*

.reset\_index())

**return** final\_dataframe

In [9]:

*# apply functions to customers\_orders*

customers = aggregate\_by\_ordered\_quantity(customers\_orders, ["customer\_id", "product\_type"])

print(customers.head())

**2. Average Return Rate**

It is the ratio of returned item quantity and ordered item quantity. This ratio is first calculated per order and then averaged for all orders of a customer.

In [10]:

*# aggregate data per customer\_id and order\_id,*

*# to see ordered item sum and returned item sum*

ordered\_sum\_by\_customer\_order = (customers\_orders

.groupby(["customer\_id", "order\_id"])

.ordered\_item\_quantity.sum()

.reset\_index())

returned\_sum\_by\_customer\_order = (customers\_orders

.groupby(["customer\_id", "order\_id"])

.returned\_item\_quantity.sum()

.reset\_index())

*# merge two dataframes to be able to calculate unit return rate*

ordered\_returned\_sums = pd.merge(ordered\_sum\_by\_customer\_order, returned\_sum\_by\_customer\_order)

In [11]:

*# calculate unit return rate per order and customer*

ordered\_returned\_sums["average\_return\_rate"] = (-1 \*

ordered\_returned\_sums["returned\_item\_quantity"] /

ordered\_returned\_sums["ordered\_item\_quantity"])

In [12]:

ordered\_returned\_sums.head()

Out[12]:

|  | **customer\_id** | **order\_id** | **ordered\_item\_quantity** | **returned\_item\_quantity** | **average\_return\_rate** |
| --- | --- | --- | --- | --- | --- |
| **0** | 1000661 | 99119989117212 | 3 | 0 | 0.0 |
| **1** | 1001914 | 79758569034715 | 1 | 0 | 0.0 |
| **2** | 1002167 | 38156088848638 | 1 | 0 | 0.0 |
| **3** | 1002167 | 57440147820257 | 1 | 0 | 0.0 |
| **4** | 1002167 | 58825523953710 | 1 | 0 | 0.0 |

In [13]:

*# take average of the unit return rate for all orders of a customer*

customer\_return\_rate = (ordered\_returned\_sums

.groupby("customer\_id")

.average\_return\_rate

.mean()

.reset\_index())

In [14]:

return\_rates = pd.DataFrame(customer\_return\_rate["average\_return\_rate"]

.value\_counts()

.reset\_index())

return\_rates.rename(columns=

{"index": "average return rate",

"average\_return\_rate": "count of unit return rate"},

inplace=**True**)

return\_rates.sort\_values(by="average return rate")

Out[14]:

|  | **average return rate** | **count of unit return rate** |
| --- | --- | --- |
| **0** | 0.000000 | 24823 |
| **9** | 0.013889 | 1 |
| **10** | 0.066667 | 1 |
| **8** | 0.083333 | 1 |
| **12** | 0.125000 | 1 |
| **5** | 0.166667 | 2 |
| **6** | 0.200000 | 2 |
| **4** | 0.250000 | 5 |
| **1** | 0.333333 | 13 |
| **11** | 0.400000 | 1 |
| **3** | 0.500000 | 9 |
| **7** | 0.666667 | 2 |
| **2** | 1.000000 | 13 |

In [15]:

*# add average\_return\_rate to customers dataframe*

customers = pd.merge(customers,

customer\_return\_rate,

on="customer\_id")

**3. Total spending**

Total spending is the aggregated sum of total sales value which is the amount after the taxes and returns.

In [16]:

*# aggreagate total sales per customer id*

customer\_total\_spending = (customers\_orders

.groupby("customer\_id")

.total\_sales

.sum()

.reset\_index())

customer\_total\_spending.rename(columns = {"total\_sales" : "total\_spending"},

inplace = **True**)

**Create features data frame**

In [17]:

*# add total sales to customers dataframe*

customers = customers.merge(customer\_total\_spending,

on="customer\_id")

In [18]:

print("The number of customers from the existing customer base:", customers.shape[0])

The number of customers from the existing customer base: 24874

In [19]:

*# drop id column since it is not a feature*

customers.drop(columns="customer\_id",

inplace=**True**)

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 |

**Visualize features**

In [21]:

fig=make\_subplots(rows=3, cols=1,

subplot\_titles=("Products Ordered",

"Average Return Rate",

"Total Spending"))

fig.append\_trace(go.Histogram(x=customers.products\_ordered),

row=1, col=1)

fig.append\_trace(go.Histogram(x=customers.average\_return\_rate),

row=2, col=1)

fig.append\_trace(go.Histogram(x=customers.total\_spending),

row=3, col=1)

fig.update\_layout(height=800, width=800,

title\_text="Distribution of the Features")

fig.show()

**Scale Features: Log Transformation**

In [22]:

**def** apply\_log1p\_transformation(dataframe, column):

*'''This function takes a dataframe and a column in the string format*

*then applies numpy log1p transformation to the column*

*as a result returns log1p applied pandas series'''*

dataframe["log\_" + column] = np.log1p(dataframe[column])

**return** dataframe["log\_" + column]

**1. Products ordered**

In [23]:

apply\_log1p\_transformation(customers, "products\_ordered")

Out[23]:

0 0.693147

1 0.693147

2 1.386294

3 0.693147

4 1.098612

...

24869 1.098612

24870 1.098612

24871 0.693147

24872 1.098612

24873 0.693147

Name: log\_products\_ordered, Length: 24874, dtype: float64

**2. Average return rate**

In [24]:

apply\_log1p\_transformation(customers, "average\_return\_rate")

Out[24]:

0 0.0

1 0.0

2 0.0

3 0.0

4 0.0

...

24869 0.0

24870 0.0

24871 0.0

24872 0.0

24873 0.0

Name: log\_average\_return\_rate, Length: 24874, dtype: float64

**3. Total spending**

In [25]:

apply\_log1p\_transformation(customers, "total\_spending")

Out[25]:

0 5.564520

1 4.384524

2 5.460436

3 4.499810

4 4.644391

...

24869 5.560682

24870 5.495117

24871 4.499810

24872 5.590987

24873 4.174387

Name: log\_total\_spending, Length: 24874, dtype: float64

**Visualize log transformation applied features**

In [26]:

fig = make\_subplots(rows=3, cols=1,

subplot\_titles=("Products Ordered",

"Average Return Rate",

"Total Spending"))

fig.append\_trace(go.Histogram(x=customers.log\_products\_ordered),

row=1, col=1)

fig.append\_trace(go.Histogram(x=customers.log\_average\_return\_rate),

row=2, col=1)

fig.append\_trace(go.Histogram(x=customers.log\_total\_spending),

row=3, col=1)

fig.update\_layout(height=800, width=800,

title\_text="Distribution of the Features after Logarithm Transformation")

fig.show()

customers.head()

Out[27]:

|  | **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 |

In [28]:

*# features we are going to use as an input to the model*

customers.iloc[:,3:]

Out[28]:

|  | **log\_products\_ordered** | **log\_average\_return\_rate** | **log\_total\_spending** |
| --- | --- | --- | --- |
| **0** | 0.693147 | 0.0 | 5.564520 |
| **1** | 0.693147 | 0.0 | 4.384524 |
| **2** | 1.386294 | 0.0 | 5.460436 |
| **3** | 0.693147 | 0.0 | 4.499810 |
| **4** | 1.098612 | 0.0 | 4.644391 |
| **...** | ... | ... | ... |
| **24869** | 1.098612 | 0.0 | 5.560682 |
| **24870** | 1.098612 | 0.0 | 5.495117 |
| **24871** | 0.693147 | 0.0 | 4.499810 |
| **24872** | 1.098612 | 0.0 | 5.590987 |
| **24873** | 0.693147 | 0.0 | 4.174387 |

24874 rows × 3 columns

**Create K-means model**

In [29]:

*# create initial K-means model*

kmeans\_model = KMeans(init='k-means++',

max\_iter=500,

random\_state=42)

In [30]:

kmeans\_model.fit(customers.iloc[:,3:])

*# print the sum of distances from all examples to the center of the cluster*

print("within-cluster sum-of-squares (inertia) of the model is:", kmeans\_model.inertia\_)

within-cluster sum-of-squares (inertia) of the model is: 1066.6314745569075

**Hyperparameter tuning: Find optimal number of clusters**

In [31]:

**def** make\_list\_of\_K(K, dataframe):

*'''inputs: K as integer and dataframe*

*apply k-means clustering to dataframe*

*and make a list of inertia values against 1 to K (inclusive)*

*return the inertia values list*

*'''*

cluster\_values = list(range(1, K+1))

inertia\_values=[]

**for** c **in** cluster\_values:

model = KMeans(

n\_clusters = c,

init='k-means++',

max\_iter=500,

random\_state=42)

model.fit(dataframe)

inertia\_values.append(model.inertia\_)

**return** inertia\_values

**Visualize different K and models**

In [32]:

*# save inertia values in a dataframe for k values between 1 to 15*

results = make\_list\_of\_K(15, customers.iloc[:, 3:])

k\_values\_distances = pd.DataFrame({"clusters": list(range(1, 16)),

"within cluster sum of squared distances": results})

In [33]:

*# visualization for the selection of number of segments*

fig = go.Figure()

fig.add\_trace(go.Scatter(x=k\_values\_distances["clusters"],

y=k\_values\_distances["within cluster sum of squared distances"],

mode='lines+markers'))

fig.update\_layout(xaxis = dict(

tickmode = 'linear',

tick0 = 1,

dtick = 1),

title\_text="Within Cluster Sum of Squared Distances VS K Values",

xaxis\_title="K values",

yaxis\_title="Cluster sum of squared distances")

fig.show()

**Update K-Means Clustering**

In [34]:

*# 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:])

Out[34]:

array([2, 0, 1, ..., 0, 1, 0])

**Add cluster centers to the visualization**

In [35]:

*# create cluster centers and actual data arrays*

cluster\_centers = updated\_kmeans\_model.cluster\_centers\_

actual\_data = np.expm1(cluster\_centers)

add\_points = np.append(actual\_data, cluster\_centers, axis=1)

add\_points

Out[35]:

array([[1.01496335e+00, 1.15284613e-03, 7.65510085e+01, 7.00601007e-01,

1.15218211e-03, 4.35093589e+00],

[2.40005740e+00, 5.19364987e-04, 2.84126659e+02, 1.22379231e+00,

5.19230164e-04, 5.65293350e+00],

[1.52730191e+00, 5.47067357e-04, 1.59911940e+02, 9.27152295e-01,

5.46917770e-04, 5.08085726e+00],

[3.95129984e+00, 5.59198198e-04, 5.81869314e+02, 1.59965014e+00,

5.59041905e-04, 6.36796300e+00]])

In [36]:

*# add labels to customers dataframe and add\_points array*

add\_points = np.append(add\_points, [[0], [1], [2], [3]], axis=1)

customers["clusters"] = updated\_kmeans\_model.labels\_

In [37]:

*# create centers dataframe from add\_points*

centers\_df = pd.DataFrame(data=add\_points, columns=["products\_ordered",

"average\_return\_rate",

"total\_spending",

"log\_products\_ordered",

"log\_average\_return\_rate",

"log\_total\_spending",

"clusters"])

centers\_df.head()

Out[37]:

|  | **products\_ordered** | **average\_return\_rate** | **total\_spending** | **log\_products\_ordered** | **log\_average\_return\_rate** | **log\_total\_spending** | **clusters** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1.014963 | 0.001153 | 76.551009 | 0.700601 | 0.001152 | 4.350936 | 0.0 |
| **1** | 2.400057 | 0.000519 | 284.126659 | 1.223792 | 0.000519 | 5.652933 | 1.0 |
| **2** | 1.527302 | 0.000547 | 159.911940 | 0.927152 | 0.000547 | 5.080857 | 2.0 |
| **3** | 3.951300 | 0.000559 | 581.869314 | 1.599650 | 0.000559 | 6.367963 | 3.0 |

In [38]:

*# align cluster centers of centers\_df and customers*

centers\_df["clusters"] = centers\_df["clusters"].astype("int")

In [39]:

centers\_df.head()

Out[39]:

|  | **products\_ordered** | **average\_return\_rate** | **total\_spending** | **log\_products\_ordered** | **log\_average\_return\_rate** | **log\_total\_spending** | **clusters** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1.014963 | 0.001153 | 76.551009 | 0.700601 | 0.001152 | 4.350936 | 0 |
| **1** | 2.400057 | 0.000519 | 284.126659 | 1.223792 | 0.000519 | 5.652933 | 1 |
| **2** | 1.527302 | 0.000547 | 159.911940 | 0.927152 | 0.000547 | 5.080857 | 2 |
| **3** | 3.951300 | 0.000559 | 581.869314 | 1.599650 | 0.000559 | 6.367963 | 3 |

In [40]:

customers.head()

Out[40]:

|  | **products\_ordered** | **average\_return\_rate** | **total\_spending** | **log\_products\_ordered** | **log\_average\_return\_rate** | **log\_total\_spending** | **clusters** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 0.0 | 260.0 | 0.693147 | 0.0 | 5.564520 | 2 |
| **1** | 1 | 0.0 | 79.2 | 0.693147 | 0.0 | 4.384524 | 0 |
| **2** | 3 | 0.0 | 234.2 | 1.386294 | 0.0 | 5.460436 | 1 |
| **3** | 1 | 0.0 | 89.0 | 0.693147 | 0.0 | 4.499810 | 0 |
| **4** | 2 | 0.0 | 103.0 | 1.098612 | 0.0 | 4.644391 | 2 |

In [41]:

*# differentiate between data points and cluster centers*

customers["is\_center"] = 0

centers\_df["is\_center"] = 1

*# add dataframes together*

customers = customers.append(centers\_df, ignore\_index=**True**)

In [42]:

customers.tail()

Out[42]:

|  | **products\_ordered** | **average\_return\_rate** | **total\_spending** | **log\_products\_ordered** | **log\_average\_return\_rate** | **log\_total\_spending** | **clusters** | **is\_center** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **24873** | 1.000000 | 0.000000 | 64.000000 | 0.693147 | 0.000000 | 4.174387 | 0 | 0 |
| **24874** | 1.014963 | 0.001153 | 76.551009 | 0.700601 | 0.001152 | 4.350936 | 0 | 1 |
| **24875** | 2.400057 | 0.000519 | 284.126659 | 1.223792 | 0.000519 | 5.652933 | 1 | 1 |
| **24876** | 1.527302 | 0.000547 | 159.911940 | 0.927152 | 0.000547 | 5.080857 | 2 | 1 |
| **24877** | 3.951300 | 0.000559 | 581.869314 | 1.599650 | 0.000559 | 6.367963 | 3 | 1 |

**Visualize Customer Segmentation**

In [43]:

*# add clusters to the dataframe*

customers["cluster\_name"] = customers["clusters"].astype(str)

In [44]:

*# visualize log\_transformation customer segments with a 3D plot*

fig = px.scatter\_3d(customers,

x="log\_products\_ordered",

y="log\_average\_return\_rate",

z="log\_total\_spending",

color='cluster\_name',

hover\_data=["products\_ordered",

"average\_return\_rate",

"total\_spending"],

category\_orders = {"cluster\_name":

["0", "1", "2", "3"]},

symbol = "is\_center"

)

fig.update\_layout(margin=dict(l=0, r=0, b=0, t=0))

fig.show()

**Check for Cluster Magnitude**

In [45]:

*# values for log\_transformation*

cardinality\_df = pd.DataFrame(

customers.cluster\_name.value\_counts().reset\_index())

cardinality\_df.rename(columns={"index": "Customer Groups",

"cluster\_name": "Customer Group Magnitude"},

inplace=**True**)

In [46]:

cardinality\_df

Out[46]:

|  | **Customer Groups** | **Customer Group Magnitude** |
| --- | --- | --- |
| **0** | 0 | 10468 |
| **1** | 2 | 7236 |
| **2** | 1 | 5100 |
| **3** | 3 | 2074 |

In [47]:

fig = px.bar(cardinality\_df, x="Customer Groups",

y="Customer Group Magnitude",

color = "Customer Groups",

category\_orders = {"Customer Groups": ["0", "1", "2", "3"]})

fig.update\_layout(xaxis = dict(

tickmode = 'linear',

tick0 = 1,

dtick = 1),

yaxis = dict(

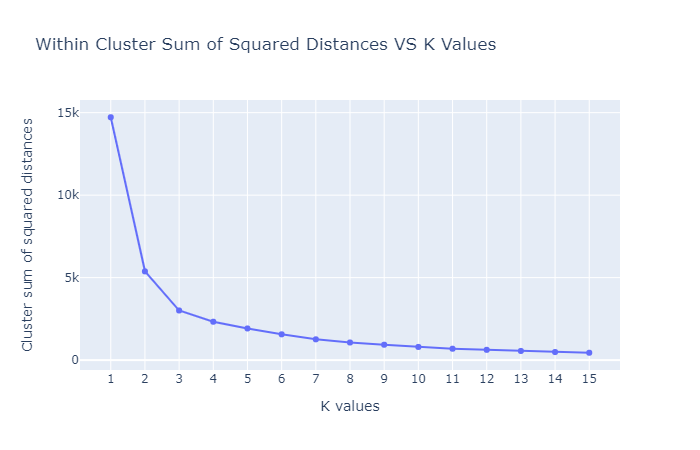
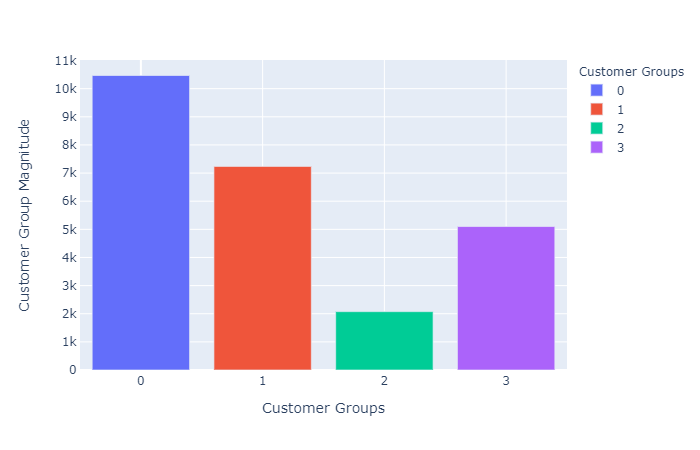
tickmode = 'linear',

tick0 = 1000,

dtick = 1000))

fig.show()

**Output screenshots**



**FUTURE SCOPE**

Companies can use [marketing automation](https://searchcustomerexperience.techtarget.com/definition/marketing-automation) software to define and create customer segments. The customer segments can be based on demographic data, psychographic data and activity-based data such as actions that users took on a website. Companies use marketing automation software to configure, schedule and execute campaigns for particular customer segments.

**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.

**REFRENCES AND BIBLIOGRAPHY**

* Datasets(<https://www.kaggle.com/datasets>)
* Sklearn libraries(<https://scikitlearn.org/stable/modules/generated/sklearn.cluster.KMeans.html>)
* Algorithm refrences(<https://nbviewer.jupyter.org/github/cereniyim/Customer-Segmentation-Unsupervised-ML-Model/blob/3c4374dd16861ea365cdf468bd9b2c28a964f4e3/Customer_Segmentation_Kmeans_Clustering.ipynb>)