Business Case: Target SQL

Context: Target is a globally renowned brand and a prominent retailer in the United States. Target makes itself a preferred shopping destination by offering outstanding value, inspiration, innovation and an exceptional guest experience that no other retailer can deliver.

Problem statement: To analyze data collected between 2016 and 2018 for the Brazil region, extract meaningful insights, and provide actionable recommendations to support data-driven decision-making and strategic business growth.

Analysis:

Q1: Import the dataset and do usual exploratory analysis steps like checking the structure & characteristics of the dataset:

1. **Data type of all columns in the "customers" table.**

Query:

```
select column_name,data_type
from sunm-442402.target.INFORMATION_SCHEMA.COLUMNS
where table_name = 'customers';
```

Query result screenshot:

Query results

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DI

Row	column_name	data_type
1	customer_id	STRING
2	customer_unique_id	STRING
3	customer_zip_code_prefix	INT64
4	customer_city	STRING
5	customer_state	STRING

Insights:

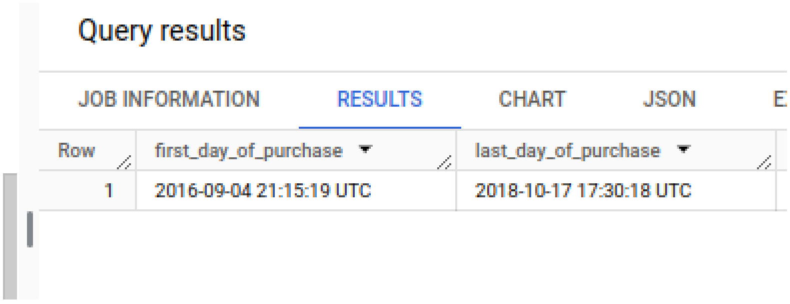
From the above query result, we can see that all the columns in customers table are of type string. Even though zip-code is stored as an integer, it is more suited for a categorical data type and not a numerical data type as numerical operations on zip code does not make sense

2. **Get the time range between which the orders were placed.**

<u>Query:</u>

```
select min(order_purchase_timestamp) as first_day_of_purchase,
max(order_purchase_timestamp) as last_day_of_purchase
from `target.orders`;
```

<u>Query result screenshot:</u>


Query results

JOB INFORMATION RESULTS CHART JSON E

Row	first_day_of_purchase	last_day_of_purchase
1	2016-09-04 21:15:19 UTC	2018-10-17 17:30:18 UTC

<u>Insights:</u>

The data given captures order transactions from the first recorded purchase on 2016-09-04 21:15:19 UTC to the most recent on 2018-10-17 17:30:18 UTC, providing a comprehensive timeline for analyzing customer behavior and sales trends during this period.

3. **Count the Cities & States of customers who ordered during the given period.**

<u>Query:</u>

```
select count(distinct c.customer_city) as city_count,
count(distinct c.customer_state) as state_count
from `target.orders` o
inner join `target.customers` c
```

```
using (customer_id);
```

Query result screenshot:

Query results

JOB INFORMATION RESULTS CHART JSON EXE

Row	city_count	state_count
1	4119	27

Insights:

From the above we can see that there are **27 distinct states** and **4119 distinct cities** where customers have placed orders, highlighting the geographical diversity of our customer base which can be used to identify regions with the highest engagement or market coverage.

Q2: In-depth Exploration:

1. **Is there a growing trend in the no. of orders placed over the past years?**
   Query:

```
select extract(year from o.order_purchase_timestamp) as order_year,
count(o.order_id) as order_count
from `target.orders` o
group by order_year
order by order_year;
```

Query result screenshot:

Query results

JOB INFORMATION RESULTS CHART

Row	order_year	order_count
1	2016	329
2	2017	45101
3	2018	54011

Insights:

From the above we can see that initially the orders were very less in the first year (order count 329 in 2016), but this increased to a significantly higher level (45101 in 2017 and 54011 in 2018). Even by considering that we have the data for only 3 months in 2016 we can see that there is an upward trend in the orders received by Target.

2. **Can we see some kind of monthly seasonality in terms of the no. of orders being placed?**

Query:

```sql
select
extract(year from o.order_purchase_timestamp) as order_year,
extract(month from o.order_purchase_timestamp) as order_month,
count(o.order_id) as order_count
from `target.orders` o
group by order_year, order_month
order by order_count desc;
```

Query result screenshot:

Query results

JOB INFORMATION RESULTS CHART JSON EXECUTION D

Row	order_year	order_month	order_count
1	2017	11	7544
2	2018	1	7269
3	2018	3	7211
4	2018	4	6939
5	2018	5	6873
6	2018	2	6728
7	2018	8	6512
8	2018	7	6292
9	2018	6	6167
10	2017	12	5673
11	2017	10	4631
12	2017	8	4331

Insights:

From the above results we can see that the order follows a certain pattern. The number of orders rises up just before December and dips down during December and again rises in the beginning of the new year till almost the mid of the year post which it starts to dip down again.

3. **During what time of the day, do the Brazilian customers mostly place their orders? (Dawn, Morning, Afternoon or Night)**
    a. **0-6 hrs : Dawn**
    b. **7-12 hrs : Mornings**
    c. **13-18 hrs : Afternoon**
    d. **19-23 hrs : Night**

Query:

```
select
case
when extract(hour from o.order_purchase_timestamp) between 0 and 6 then
'Dawn'
when extract(hour from o.order_purchase_timestamp) between 7 and 12 then
'Morning'
when extract(hour from o.order_purchase_timestamp) between 13 and 18 then
'Afternoon'
when extract(hour from o.order_purchase_timestamp) between 19 and 23 then
'Night'
end as time_of_day,
count(o.order_id) as order_count
from `target.orders` o
inner join `target.customers` c
using (customer_id)
group by time_of_day
order by order_count desc;
```

Query result screenshot:

Query results

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTIC

Row	time_of_day	order_count
1	Afternoon	38135
2	Night	28331
3	Morning	27733
4	Dawn	5242

Insights:

From the above query results we can see that the Brazilians place their orders mostly during the Afternoon.

Q3: Evolution of E-commerce orders in the Brazil region:

1. **Get the month on month no. of orders placed in each state.**

Query:

```
select
c.customer_state,
extract(year from o.order_purchase_timestamp) as order_year,
extract(month from o.order_purchase_timestamp) as order_month,
count(o.order_id) as order_count
from `target.orders` o
inner join `target.customers` c
using (customer_id)
group by c.customer_state, order_year, order_month
order by c.customer_state, order_year, order_month;
```

Query result screenshot:

Query results

JOB INFORMATION RESULTS CHART JSON EXECUTION DETAILS EXECUTION GR

Row	customer_state	order_year	order_month	order_count
1	AC	2017	1	2
2	AC	2017	2	3
3	AC	2017	3	2
4	AC	2017	4	5
5	AC	2017	5	8
6	AC	2017	6	4
7	AC	2017	7	5
8	AC	2017	8	4
9	AC	2017	9	5
10	AC	2017	10	6
11	AC	2017	11	5

Insights:

We can derive multiple insights from the above results and below are a few of them.

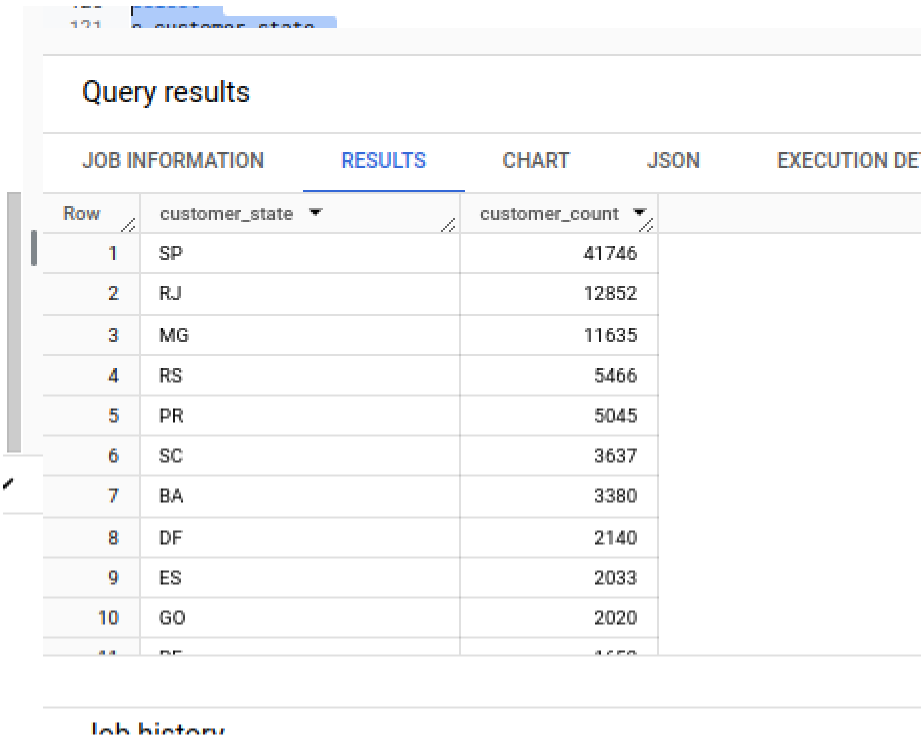
- State SP is the state with the highest number of orders and most of the orders were in the year 2018
- Orders in state AC is more in the beginning of 2018 and after it has a few downs and ups.

2. **How are the customers distributed across all the states?**

Query:

```
select
c.customer_state,
count(distinct c.customer_id) as customer_count
from `target.customers` c
group by c.customer_state
order by customer_count desc;
```

Query result screenshot:



Row	customer_state	customer_count
1	SP	41746
2	RJ	12852
3	MG	11635
4	RS	5466
5	PR	5045
6	SC	3637
7	BA	3380
8	DF	2140
9	ES	2033
10	GO	2020

<u>Insights:</u>

From the results we can see that the majority of the customers are located in SP, i.e. more than thrice the customers present in the state RJ, which has the second highest customer count.

Q4: Impact on Economy: Analyze the money movement by e-commerce by looking at order prices, freight and others.

1. **Get the % increase in the cost of orders from year 2017 to 2018 (include months between Jan to Aug only).**

   <u>Query:</u>

```
with yearly_payment as (
select
extract(year from o.order_purchase_timestamp) as order_year,
sum(p.payment_value) as total_payment
```

```sql
from `target.orders` o
inner join `target.payments` p
using (order_id)
where extract(month from o.order_purchase_timestamp) between 1 and 8
group by order_year
having order_year in (2017, 2018)
)
select y2017,y2018,
(y2018.total_payment - y2017.total_payment) / y2017.total_payment * 100 as
percentage_increase
from
(select total_payment from yearly_payment where order_year = 2017) y2017,
(select total_payment from yearly_payment where order_year = 2018) y2018;
```

Query result screenshot:

Query results

JOB INFORMATION | RESULTS | CHART | JSON

Row	total_payment	total_payment	percentage_increase
1	3669022.120000...	8694733.839999...	136.9768716466...

Insights:

The results show that the payment value has increased by almost 137% from the year 2017 to 2018 for the months between January to August.

2. **Calculate the Total & Average value of the order price for each state.**

Query:

```sql
select
c.customer_state,
round(sum(p.payment_value),2) as total_order_value,
round(avg(p.payment_value),2) as average_order_value
from `target.orders` o
inner join `target.payments` p
using (order_id)
```

```sql
inner join `target.customers` c
using (customer_id)
group by c.customer_state
order by c.customer_state;
```

Query result screenshot:

Query results

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS

Row	customer_state	total_order_value	average_order_value
1	AC	19680.62	234.29
2	AL	96962.06	227.08
3	AM	27966.93	181.6
4	AP	16262.8	232.33
5	BA	616645.82	170.82
6	CE	279464.03	199.9
7	DF	355141.08	161.13
8	ES	325967.55	154.71
9	GO	350092.31	165.76
10	MA	152523.02	198.86
11	MG	1872257.26	154.71
12	MS	137534.84	186.87
13	MT	187029.29	195.23
14	PA	218295.85	215.92

Insights:

Multiple insights can be derived from the above results. Below are a few of them

- State SP shows a significant contribution to the total order value, but its average order value is comparatively lower, suggesting that a high volume of smaller orders drives the revenue here. This state can be targeted with campaigns promoting bundled deals or bulk discounts to increase the average order size.
- State PB has a moderate total order value but a relatively high average order value, indicating a preference for premium or higher-priced products. This state can be targeted with exclusive product launches or premium membership programs to further capitalize on customer spending habits.

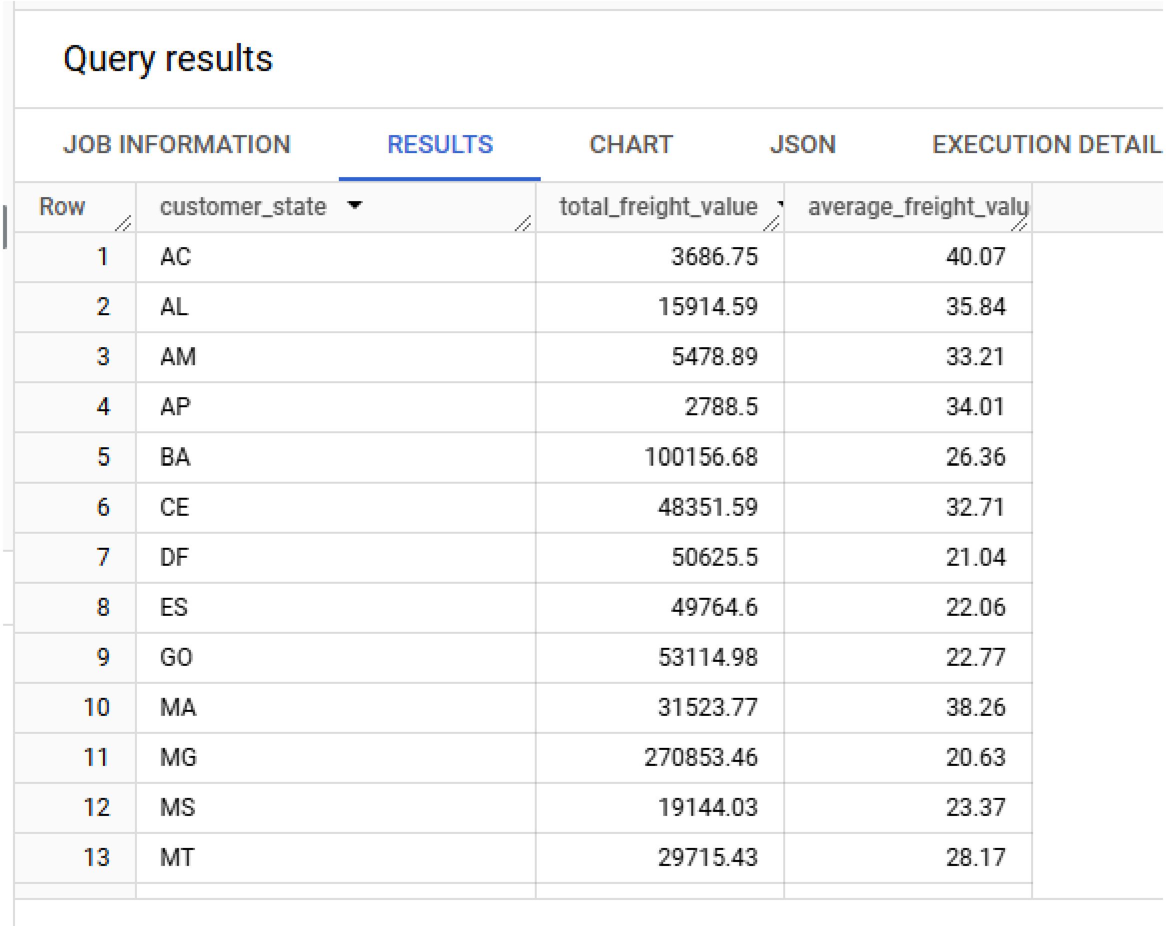
- States with lower total order values, such as RR, AP, and AC may represent untapped potential. Focused marketing campaigns or region-specific discounts could help boost sales in these regions.

3. **Calculate the Total & Average value of order freight for each state.**

Query:

```
select
c.customer_state,
round(sum(oi.freight_value), 2) as total_freight_value,
round(avg(oi.freight_value), 2) as average_freight_value
from `target.orders` o
inner join `target.order_items` oi
using (order_id)
inner join `target.customers` c
using (customer_id)
group by c.customer_state
order by c.customer_state;
```

Query result screenshot:


Query results

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAIL

Row	customer_state	total_freight_value	average_freight_valu
1	AC	3686.75	40.07
2	AL	15914.59	35.84
3	AM	5478.89	33.21
4	AP	2788.5	34.01
5	BA	100156.68	26.36
6	CE	48351.59	32.71
7	DF	50625.5	21.04
8	ES	49764.6	22.06
9	GO	53114.98	22.77
10	MA	31523.77	38.26
11	MG	270853.46	20.63
12	MS	19144.03	23.37
13	MT	29715.43	28.17

<u>Insights:</u>

Below are few insights based on the above results

- State SP has a high total transport cost and a low average transport cost, indicating that this state has a huge order count with a low order value for each order.
- State RR has a high average transport cost and a comparatively low total transport cost, indicating that this has fewer orders, but each order is of high value.

Q5: Analysis based on sales, freight, and delivery time

1. **Find the no. of days taken to deliver each order from the order's purchase date as delivery time.**
   Also, calculate the difference (in days) between the estimated & actual delivery date of an order.
   Do this in a single query.

Query:

```
select
o.order_id,
date_diff(o.order_delivered_customer_date, o.order_purchase_timestamp, day)
as time_to_deliver,
date_diff(o.order_delivered_customer_date, o.order_estimated_delivery_date,
day) as diff_estimated_delivery
from `target.orders` o;
```

Query result screenshot:

Query results

JOB INFORMATION RESULTS CHART JSON EXECUTION

Row	order_id	time_to_deliver	diff_estimated_delive
1	1950d777989f6a877539f5379...	30	12
2	2c45c33d2f9cb8ff8b1c86cc28...	30	-28
3	65d1e226dfaeb8cdc42f66542...	35	-16
4	635c894d068ac37e6e03dc54e...	30	-1
5	3b97562c3aee8bdedcb5c2e45...	32	0
6	68f47f50f04c4cb6774570cfde...	29	-1
7	276e9ec344d3bf029ff83a161c...	43	4
8	54e1a3c2b97fb0809da548a59...	40	4
9	fd04fa4105ee8045f6a0139ca5...	37	1
10	302bb8109d097a9fc6e9cefc5...	33	5
11	66057d37308e787052a32828...	38	6
12	19135c945c554eebfd7576c73...	36	2
13	4493e45e7ca1084efcd38ddeb...	34	0
14	70c77e51e0f179d75a64a6141...	42	11

Insights:

The time to deliver column represents the time taken for the order to reach the customer from the date of ordering. This can be improved wherever the time is too high.

The difference in estimated delivery is the difference between the delivered date and the estimated delivery date. If this is low or negative, it means the order was delivered very fast and in case of negative value it means the order got delivered faster than expected. Such deliveries can be analyzed to find out what helped with reducing the time and the same can be implemented to other orders wherever possible.

2. **Find out the top 5 states with the highest & lowest average freight value.**

Query:

```
with highest_avg_freight as (
select
c.customer_state,
round(avg(oi.freight_value), 2) as average_freight_value,
row_number() over (order by avg(oi.freight_value) desc) as row_num
from `target.order_items` oi
inner join `target.orders` o
using (order_id)
inner join `target.customers` c
using (customer_id)
group by c.customer_state
),
lowest_avg_freight as (
select
c.customer_state,
round(avg(oi.freight_value), 2) as average_freight_value,
row_number() over (order by avg(oi.freight_value) asc) as row_num
from `target.order_items` oi
inner join `target.orders` o
using (order_id)
inner join `target.customers` c
using (customer_id)
group by c.customer_state
)
select
hf.customer_state as high_freight_state,
hf.average_freight_value as high_freight_value,
lf.customer_state as low_freight_state,
lf.average_freight_value as low_freight_value
from highest_avg_freight hf
join lowest_avg_freight lf
on hf.row_num = lf.row_num
where hf.row_num <= 5 and lf.row_num <= 5;
```

Query result screenshot:

Query results

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	high_freight_state	high_freight_value	low_freight_state	low_freight_value
1	RR	42.98	SP	15.15
2	PB	42.72	PR	20.53
3	RO	41.07	MG	20.63
4	AC	40.07	RJ	20.96
5	PI	39.15	DF	21.04

Insights:

The first two columns display the top 5 states with the highest average freight values, while the next two columns show the top 5 states with the lowest average freight values. These averages can help estimate the likely transport costs for each order. Additionally, this data suggests that states with lower freight values may be ordering goods that are easier to transport, while states with higher freight values are likely to order goods that are more challenging to transport.

3. **Find out the top 5 states with the highest & lowest average delivery time.**

Query:

```
with delivery_times as (
select
c.customer_state,
date_diff(o.order_delivered_customer_date, o.order_purchase_timestamp, day)
as delivery_time
from `target.orders` o
inner join `target.customers` c
on o.customer_id = c.customer_id
where o.order_status = 'delivered'
),
ranked_delivery_times as (
select
customer_state as state,
avg(delivery_time) as avg_delivery_time,
```

```sql
rank() over (order by avg(delivery_time) desc) as high_rank,
rank() over (order by avg(delivery_time) asc) as low_rank
from delivery_times
group by customer_state
)
select
high_states.state as high_state,
round(high_states.avg_delivery_time,2) as high_avg_delivery_time,
low_states.state as low_state,
round(low_states.avg_delivery_time,2) as low_avg_delivery_time
from ranked_delivery_times high_states
join ranked_delivery_times low_states
on high_states.high_rank = low_states.low_rank
where high_states.high_rank <= 5 and low_states.low_rank <= 5
order by high_states.high_rank;
```

Query result screenshot:

Query results

JOB INFORMATION RESULTS CHART JSON EXECUTION DETAILS EXECUTION GRAPH

Row	high_state	high_avg_delivery_tir	low_state	low_avg_delivery_tim
1	RR	28.98	SP	8.3
2	AP	26.73	PR	11.53
3	AM	25.99	MG	11.54
4	AL	24.04	DF	12.51
5	PA	23.32	SC	14.48

Insights:

The results show the top 5 states with the highest average delivery times and the top 5 states with the lowest average delivery times. Analyzing the fast delivery states can provide valuable insights into the factors contributing to quicker deliveries. These factors can potentially be adopted or replicated by the states with slower deliveries to improve their performance.

4. **Find out the top 5 states where the order delivery is really fast as compared to the estimated date of delivery.**

Query:

```
select customer_state as state,
round(avg(date_diff(o.order_delivered_customer_date,
o.order_estimated_delivery_date, day)),2) as avg_speed_delivery
from `target.customers` as c
join `target.orders` as o on c.customer_id = o.customer_id
where o.order_status = 'delivered'
group by state
order by avg_speed_delivery
limit 5;
```

Query result screenshot:

Query results

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	state	avg_speed_delivery
1	AC	-19.76
2	RO	-19.13
3	AP	-18.73
4	AM	-18.61
5	RR	-16.41

Insights:

The results highlight the top 5 states with the fastest deliveries, where the negative values in the average indicate that, on average, orders were delivered well ahead of the estimated delivery date. By analyzing the factors contributing to this efficiency in these top states, we can explore opportunities to replicate these practices in other states to enhance overall delivery performance.

Q6: Analysis based on the payments

1. **Find the month on month no. of orders placed using different payment types.**

Query:

```sql
select
extract(year from o.order_purchase_timestamp) as year,
extract(month from o.order_purchase_timestamp) as month,
p.payment_type,
count(o.order_id) as number_of_orders
from `target.orders` o
join `target.payments` p
on o.order_id = p.order_id
group by year, month, p.payment_type
order by year, month, p.payment_type;
```

Query result screenshot:

Query results

JOB INFORMATION RESULTS CHART JSON EXECUTION DETAILS EXECUTION GRAPH

Row	year	month	payment_type	number_of_orders
1	2016	9	credit_card	3
2	2016	10	UPI	63
3	2016	10	credit_card	254
4	2016	10	debit_card	2
5	2016	10	voucher	23
6	2016	12	credit_card	1
7	2017	1	UPI	197
8	2017	1	credit_card	583
9	2017	1	debit_card	9
10	2017	1	voucher	61
11	2017	2	UPI	398
12	2017	2	credit_card	1356

Insights:

The number_of_orders column shows the count of orders placed for each payment method. This data allows us to identify which payment type is most preferred by customers, providing insights into customer behavior and helping tailor payment options to improve user experience.

2. **Find the no. of orders placed on the basis of the payment installments that have been paid.**

Query:

```sql
select
payment_installments,
count(distinct p.order_id) as number_of_orders
from `target.payments` p
group by payment_installments
order by payment_installments;
```

Query result screenshot:

Query results

JOB INFORMATION RESULTS CHART JS

Row	payment_installment	number_of_orders
1	0	2
2	1	49060
3	2	12389
4	3	10443
5	4	7088
6	5	5234
7	6	3916
8	7	1623
9	8	4253
10	9	644
11	10	5315
12	11	23

Insights:

The results provide the number of orders for each installment option. This data helps us analyze customer preferences, such as how many customers prefer to pay in full (no installments) versus opting for payment plans like 3-month, 6-month, or longer installment options. By understanding these preferences, we can gain valuable insights into how customers value payment flexibility.

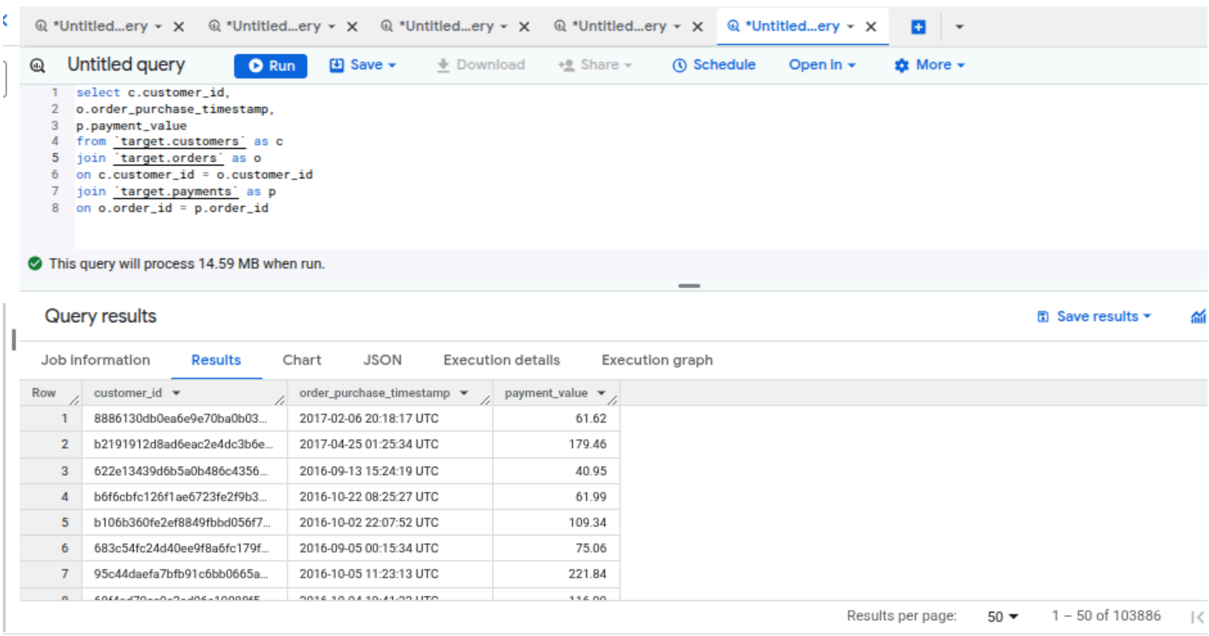
For example, if a significant number of customers choose installment plans, it might indicate a demand for more flexible payment options. On the other hand, if most customers prefer paying in full, it could suggest they prioritize simplicity or may be influenced by discounts for upfront payments.

These insights can be used to tailor offers and promotions. For instance, you could incentivize installment plans by offering low or zero interest rates, or encourage full payments by providing discounts for upfront transactions. Ultimately, this analysis helps businesses align their payment strategies with customer preferences, improving satisfaction and driving sales.

Segmenting customers using RFM

First I am extracting the columns that are needed for this.

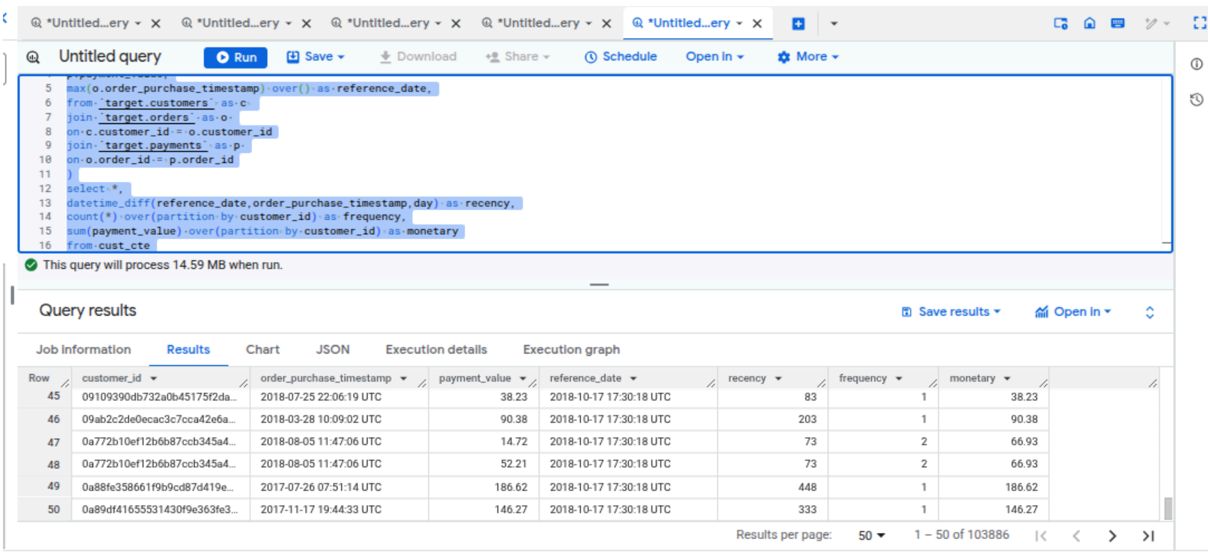
```sql
select c.customer_id,
o.order_purchase_timestamp,
p.payment_value
from `target.customers` as c
join `target.orders` as o
on c.customer_id = o.customer_id
join `target.payments` as p
on o.order_id = p.order_id
```



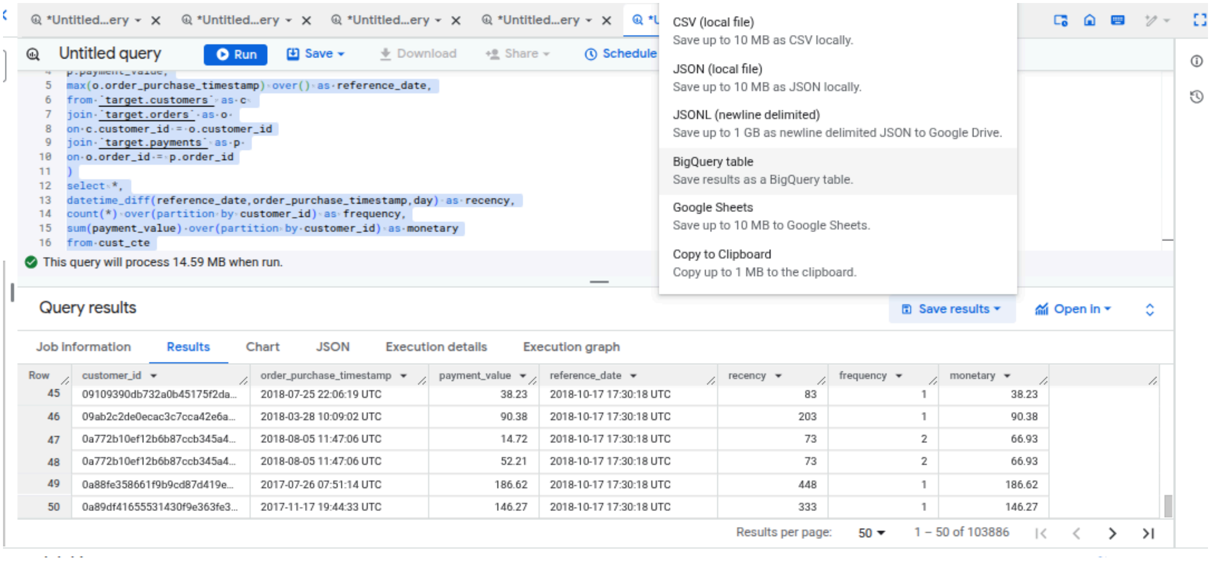
I am also exporting this data as a csv file so that I can use that later for comparison with other clustering algos.

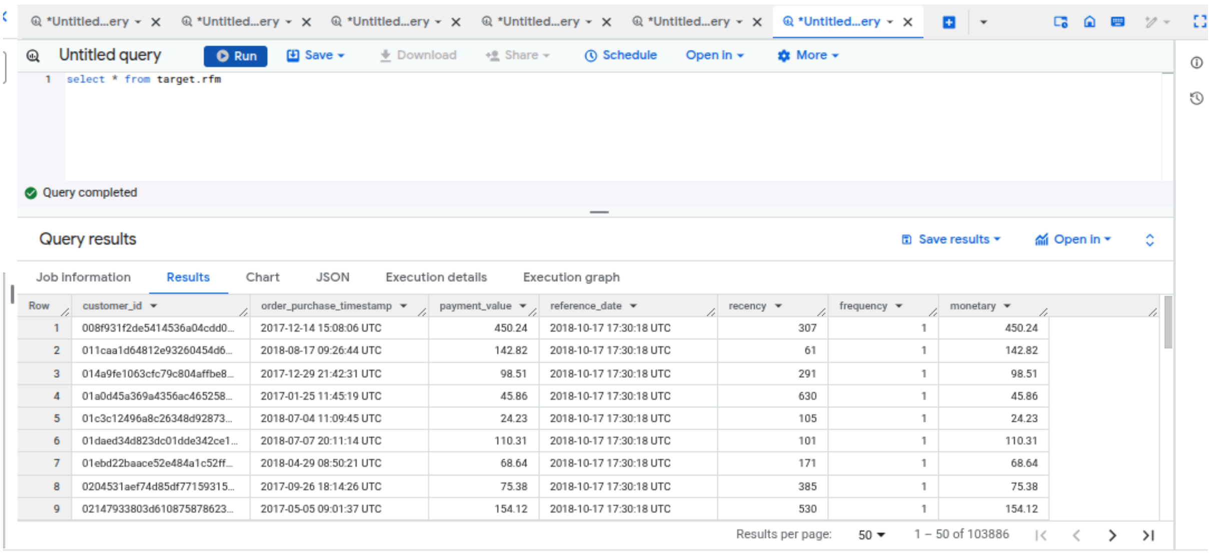
Now, below I have the draft columns with RFM values:

```
with cust_cte as (
select c.customer_id,
o.order_purchase_timestamp,
p.payment_value,
max(o.order_purchase_timestamp) over() as reference_date,
from `target.customers` as c
join `target.orders` as o
on c.customer_id = o.customer_id
join `target.payments` as p
on o.order_id = p.order_id
)
select *,
datetime_diff(reference_date,order_purchase_timestamp,day) as recency,
count(*) over(partition by customer_id) as frequency,
sum(payment_value) over(partition by customer_id) as monetary
from cust_cte
```



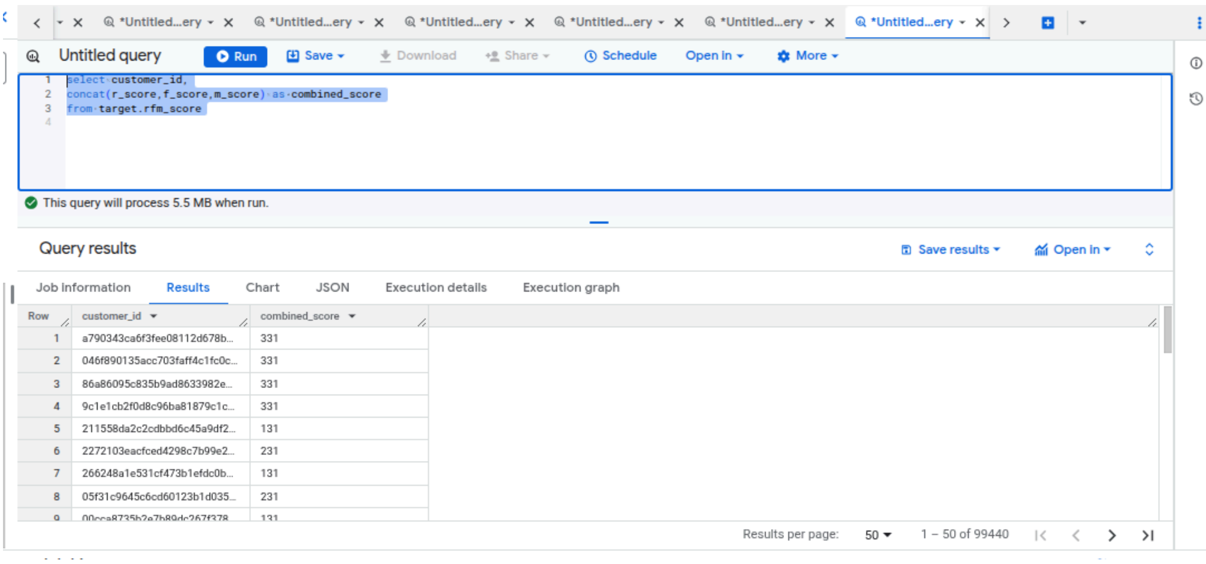
I am saving this table so that I can query using a simple query instead of this complex one.





Final segregation of customers based on rfm

```
select customer_id,
concat(r_score,f_score,m_score) as combined_score
from target.rfm_score
```




```sql
select
concat(r_score,f_score,m_score) as combined_score,
count(customer_id) as count_of_customers
from target.rfm_score
group by 1
```

Job information | Results | Chart | JSON

Row	combined_score	count_of_customers
1	311	10389
2	111	11082
3	211	10953
4	331	52
5	131	83
6	231	80
7	212	10458
8	112	10935
9	312	10629
10	232	225
11	132	285
12	332	186
13	113	10593
14	213	10575
15	313	10865
16	233	620
17	333	585
18	133	845

With the above we are able to see the number of customers segregated based on their RFM scores.

Now for curiosity sake I am checking the clustering using k-means clustering. Just to see how that clusters the customers based on the same data.

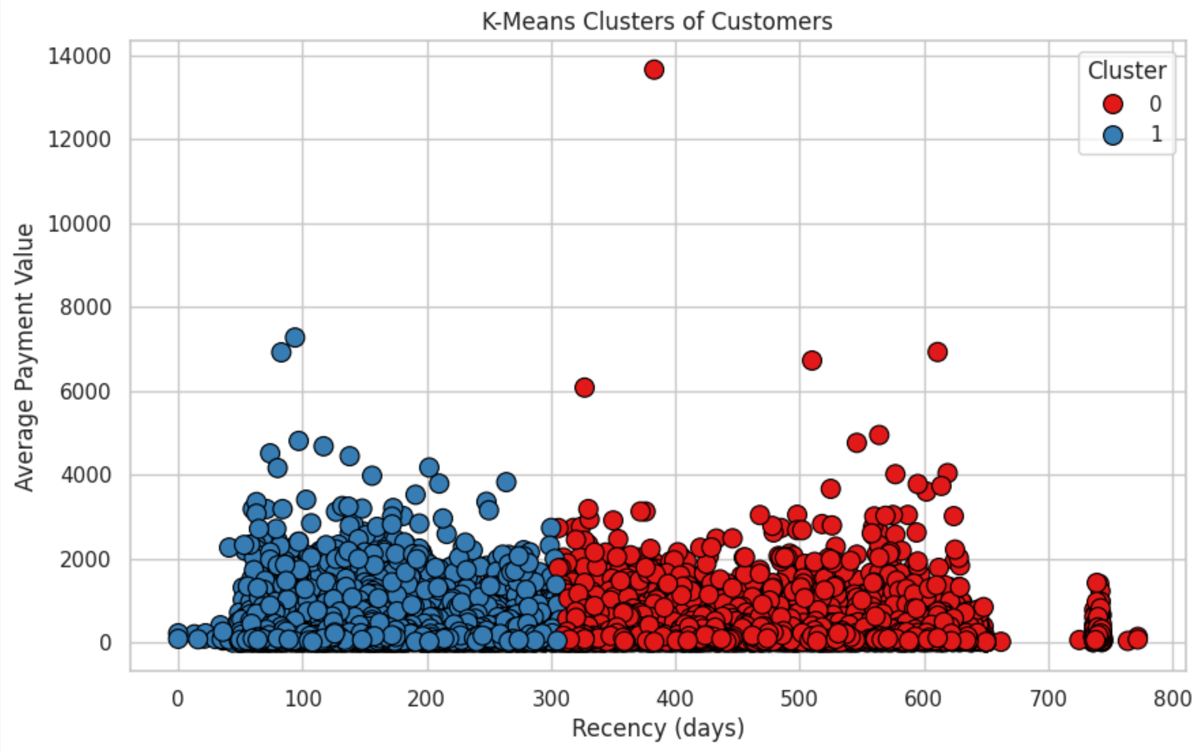
```
import pandas as pd
from datetime import datetime
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

df = pd.read_csv('cust.csv')
df['order_purchase_timestamp'] =
pd.to_datetime(df['order_purchase_timestamp'])
max_date = df['order_purchase_timestamp'].max()
df['recency_days'] = (max_date -
df['order_purchase_timestamp']).dt.days
df = df[['customer_id', 'recency_days', 'payment_value']]
df = df.groupby('customer_id').agg({'recency_days': 'mean',
'payment_value': 'mean'}).reset_index()
scaler = StandardScaler()
scaled = scaler.fit_transform(df[['recency_days', 'payment_value']])
kmeans = KMeans(n_clusters=4)
df['cluster'] = kmeans.fit_predict(scaled)
print(df)
```

```
import matplotlib.pyplot as plt
import seaborn as sns


sns.set(style="whitegrid")


plt.figure(figsize=(10, 6))
sns.scatterplot(
   x='recency_days',
   y='payment_value',
   hue='cluster',
   palette='Set1',
   data=df,
   s=100,
   edgecolor='black'
)

plt.title('K-Means Clusters of Customers')
```

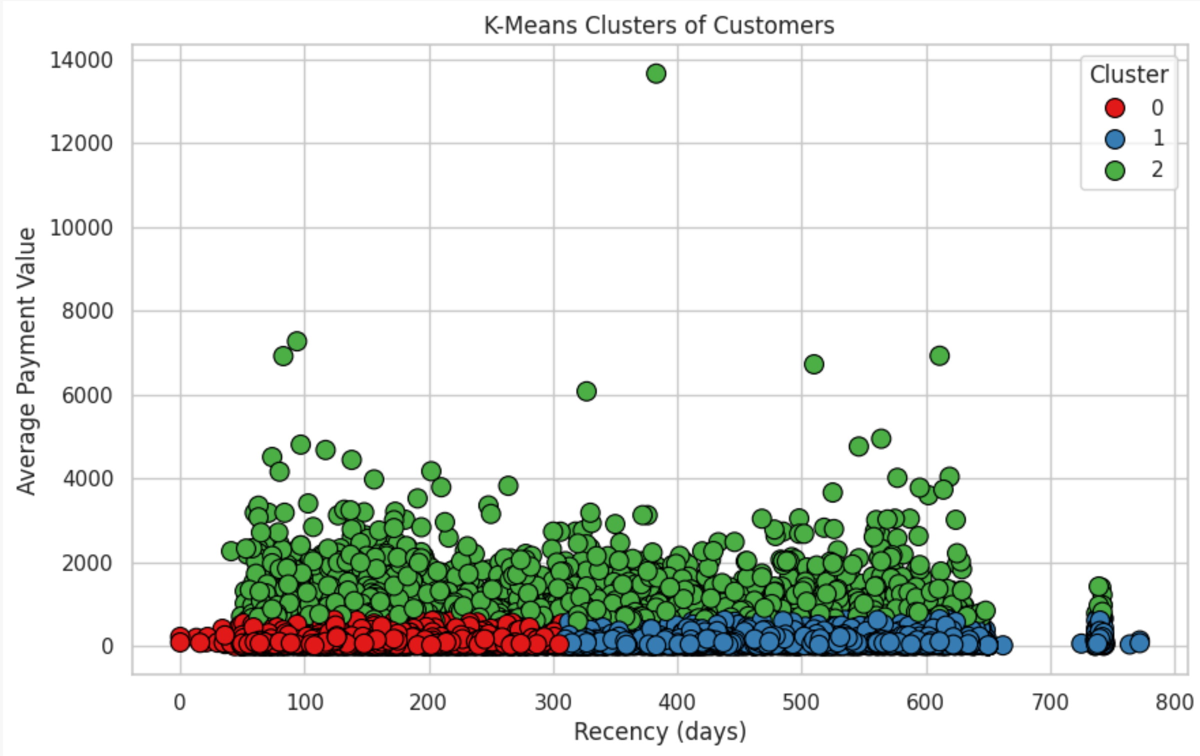
```
plt.xlabel('Recency (days)')
plt.ylabel('Average Payment Value')
plt.legend(title='Cluster')
plt.show()
```

With the above code, and by tweaking the `n_clusters`, we get the below.

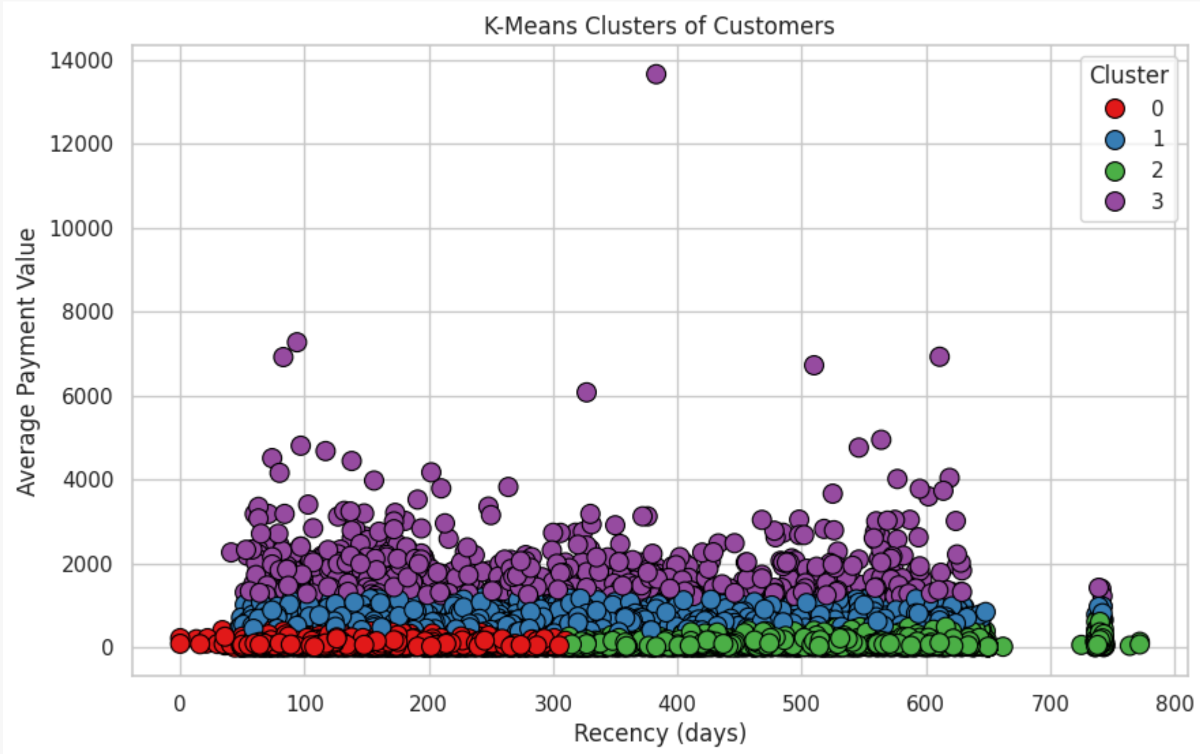
```
n_clusters=2
```



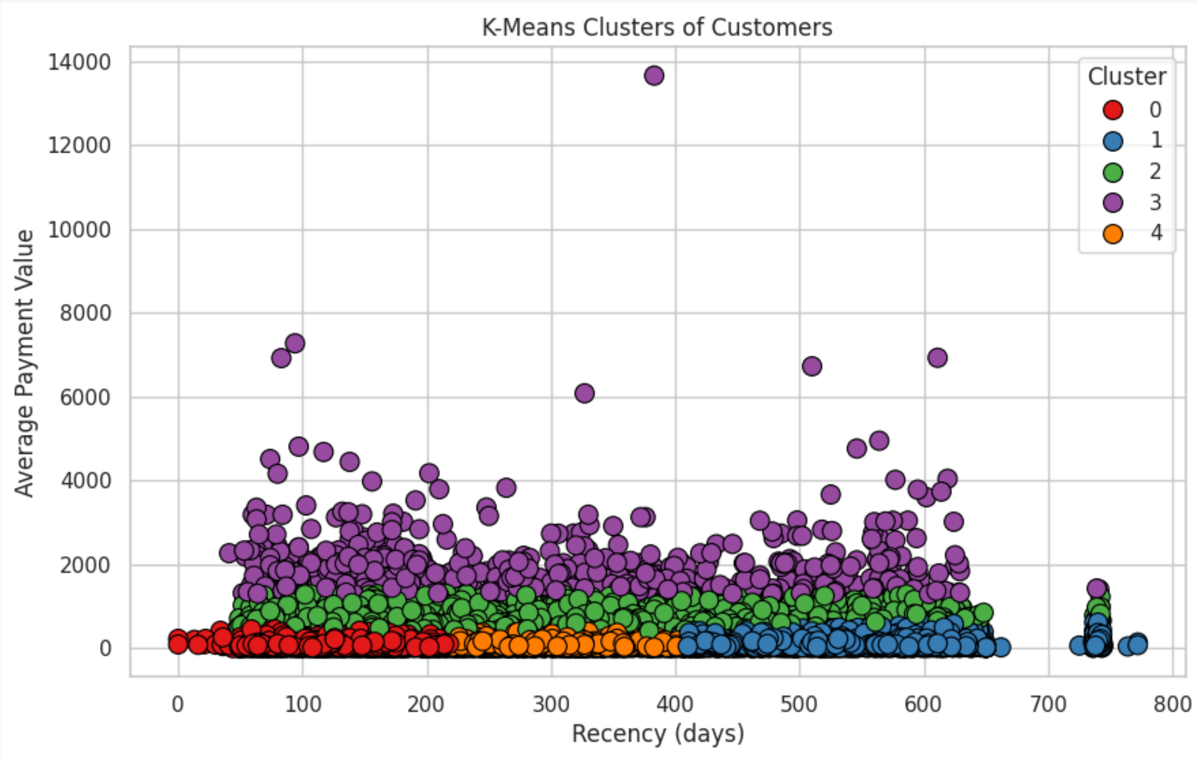

```
n_clusters=3
```




```
n_clusters=4
```




```
n_clusters=5
```



We can see here that with increase in clusters we see splits in the closely packed data. But we can see the main split between high spenders and low spending customers even across recency.

Here we see that the clusters have a different count of customers compared to RFM.

```
cluster_counts = df.groupby('cluster')['customer_id'].count().reset_index()
cluster_counts.columns = ['cluster', 'customer_count']
cluster_counts
```

	cluster	customer_count
0	0	8166
1	1	12909
2	2	603
3	3	12658
4	4	1339
5	5	2499
6	6	52
7	7	12863
8	8	3259
9	9	187
10	10	6
11	11	10219
12	12	6216
13	13	940
14	14	14623
15	15	7288
16	16	5520
17	17	93

However, when I am checking for the customer_ids that belong to the same cluster in RFM, I see that the IDs are grouped within 2 or 3 clusters.

Below all these IDs belong to the same cluster in RFM, and K-means has them in either cluster 7 or 1.

```python
customer_ids_to_check = [
    'c276d4ce04e35d109575d3282cd2bd20',
    '0f6ae9992fe5367634017e013374aee4',
    'f07aa6675ef85cbacaafe5ffc8e17f8c',
    '0b94e6ad540003cdc27b09854bd6f76f',
    '848a418025af5ffbbadfdedd85081da7',
    'd870167097156772b8af23d558b4faa7',
    '9026f16b4a80f2efd9dbc7be05f61cfd',
    '8a8476a80099f2b63c573d23679532ec',
    '0e681bde5fba98aac2a57c904d4401c9'
]

filtered = df[df['customer_id'].isin(customer_ids_to_check)]

filtered
```

	customer_id	recency_days	payment_value	cluster
4465	0b94e6ad540003cdc27b09854bd6f76f	74.0	37.585	7
5538	0e681bde5fba98aac2a57c904d4401c9	191.0	72.875	1
5929	0f6ae9992fe5367634017e013374aee4	106.0	37.255	7
51407	848a418025af5ffbbadfdedd85081da7	184.0	37.610	1
53690	8a8476a80099f2b63c573d23679532ec	112.0	47.870	7
55890	9026f16b4a80f2efd9dbc7be05f61cfd	91.0	18.990	7
75823	c276d4ce04e35d109575d3282cd2bd20	163.0	37.205	1
84315	d870167097156772b8af23d558b4faa7	69.0	37.755	7
93568	f07aa6675ef85cbacaafe5ffc8e17f8c	181.0	37.570	1

customer_ids below are in the same group in RFM, and k-means has divided them into 3 clusters.

```python
customer_ids_to_check = [
    'a743936f44e2d520e1a437c30011d5bd',
    '52701c28ce8192ff074de5576a1b9288',
    'e35070616c6bd238074e2ee5131113fd',
    '9d9bba3706f6e6f719bafea5532c3b83',
    '3af9ceb92649140512cebf7ae5c3b8f2',
    'cc64309188751727fe403d8570630495',
    '1a6485e76c0643994cedafa270691991',
    'f4b6532c70b2307896e02a2213906314'
]

filtered = df[df['customer_id'].isin(customer_ids_to_check)]

filtered
```

	customer_id	recency_days	payment_value	cluster
10254	1a6485e76c0643994cedafa270691991	578.0	24.86	16
22946	3af9ceb92649140512cebf7ae5c3b8f2	346.0	23.71	3
31952	52701c28ce8192ff074de5576a1b9288	365.0	21.95	3
61213	9d9bba3706f6e6f719bafea5532c3b83	459.0	22.78	0
64979	a743936f44e2d520e1a437c30011d5bd	568.0	21.86	16
79668	cc64309188751727fe403d8570630495	498.0	24.28	0
88424	e35070616c6bd238074e2ee5131113fd	358.0	22.03	3
95173	f4b6532c70b2307896e02a2213906314	352.0	25.33	3

Now I am checking for all the ids that belong to a single RFM cluster by exporting the data into a csv file and importing as a pandas dataFrame.

```
[33] df_same_cluster = pd.read_csv('same_cluster_ids.csv')
```

```
df_same_cluster
```

	customer_id	combined_score
0	03b753419e88796de2c7127ff13dce58	133
1	84f8d3841c9595ec254d45d1e103b056	133
2	24847010ba7f47ae100a5e27e6150c54	133
3	6995f12b21fdf8e6a245b17888e38a46	133
4	855defd5536f14c713b89056b7bae001	133
...	...	...
840	d3e82ccec3cb5f956a38d96c057ceaae	133
841	f959b7bc834045511217e6410985963f	133
842	926b6a6fb8b6081e00b335edaf578d35	133
843	de832e8dbb1f588a47013e53feaa67cc	133
844	9af2372a1e49340278e7c1ef8d749f34	133

845 rows × 2 columns

All of these belong to the same cluster as per RFM. Below is how k-means has segregated this.

```
[35] target_customer_ids = df_same_cluster['customer_id'].tolist()
```

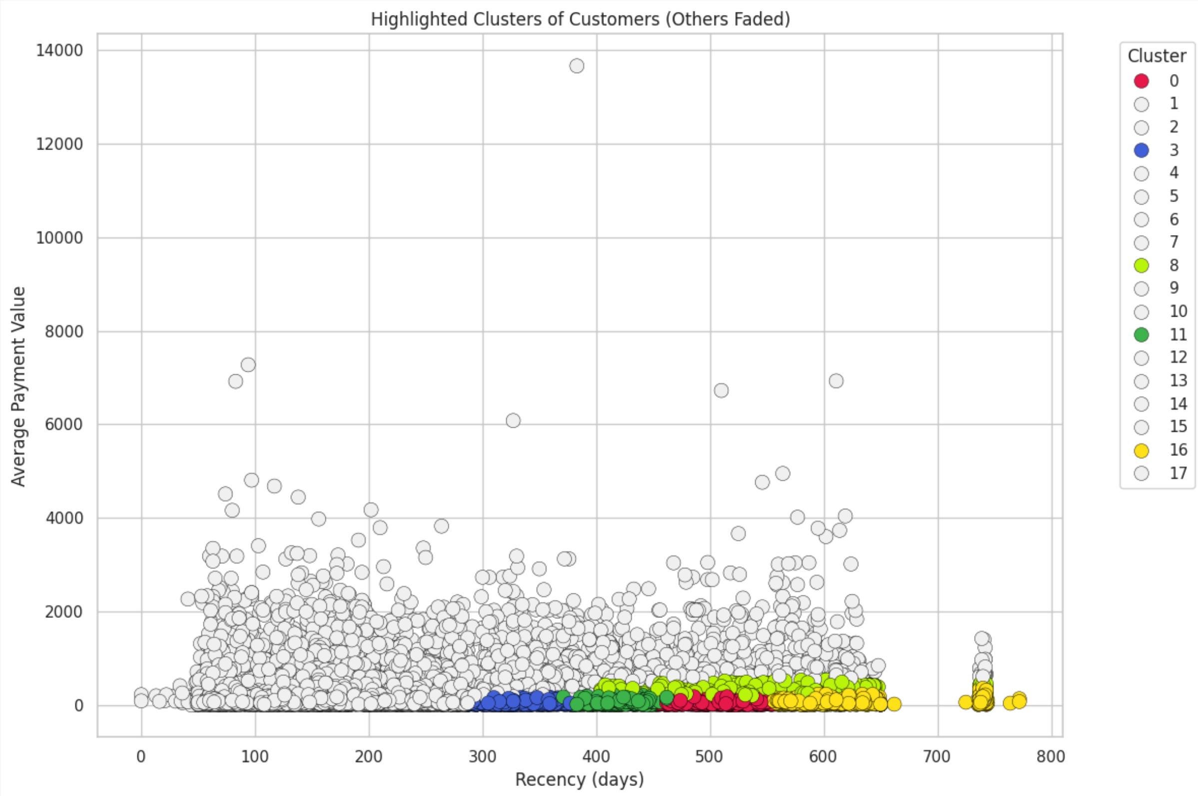
```
[36] matched_df = df[df['customer_id'].isin(target_customer_ids)]
```

```
cluster_distribution = matched_df['cluster'].value_counts().reset_index()
cluster_distribution.columns = ['cluster', 'customer_count']
print(cluster_distribution)
```

```
   cluster  customer_count
0        0             264
1       11             249
2       16             154
3        3             120
4        8              25
5        4              16
6       12              13
7        2               3
8       17               1
```

It seems like too many clusters. But cluster 8,4,12,2,17 have too few datapoints

Let me plot this visually and see if the clusters are nearby.



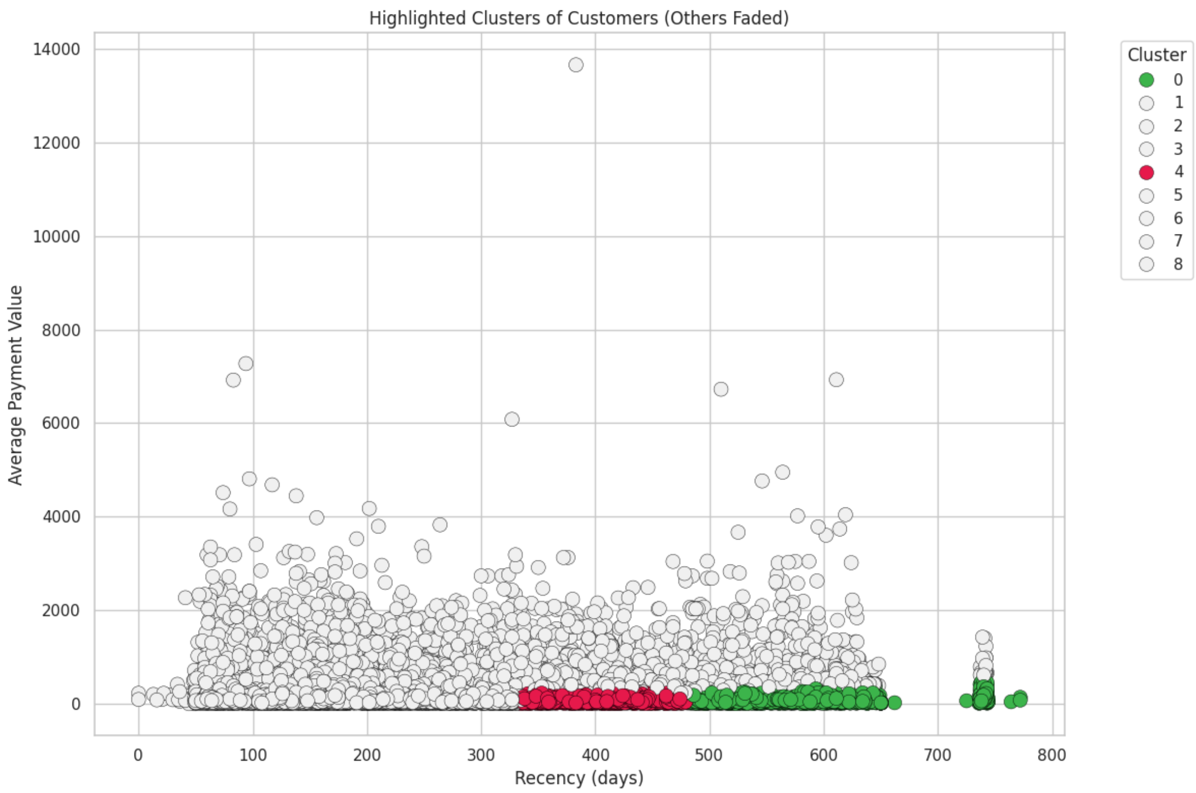
As we can see from the plot, all these are closer to each other. Also, the recency days are stretched. In RFM we have grouped into 3 parts, first 33%, next 33% and the remaining. But here it is more granular and shows the stretch of days for different groups.

Let me reduce the cluster count and see if that makes a difference. With reduced cluster count below is what we get

```
cluster_distribution = matched_df['cluster'].value_counts().reset_index()
cluster_distribution.columns = ['cluster', 'customer_count']
print(cluster_distribution)
```

```
   cluster  customer_count
0        4             441
1        0             358
2        8              32
3        6              12
4        2               2
```

Ignoring cluster 8,6,2, as these have few records, we get the below plot.



Here we see that these IDs are more together.

This demonstrates the power of machine learning. For RFM, I had to invest a considerable amount of time, but using the k-means clustering, I was able to get the clusters together more easily and in a way more accurately considering the granular way in which the clusters get formed.