User Segmentation Analysis Using Python

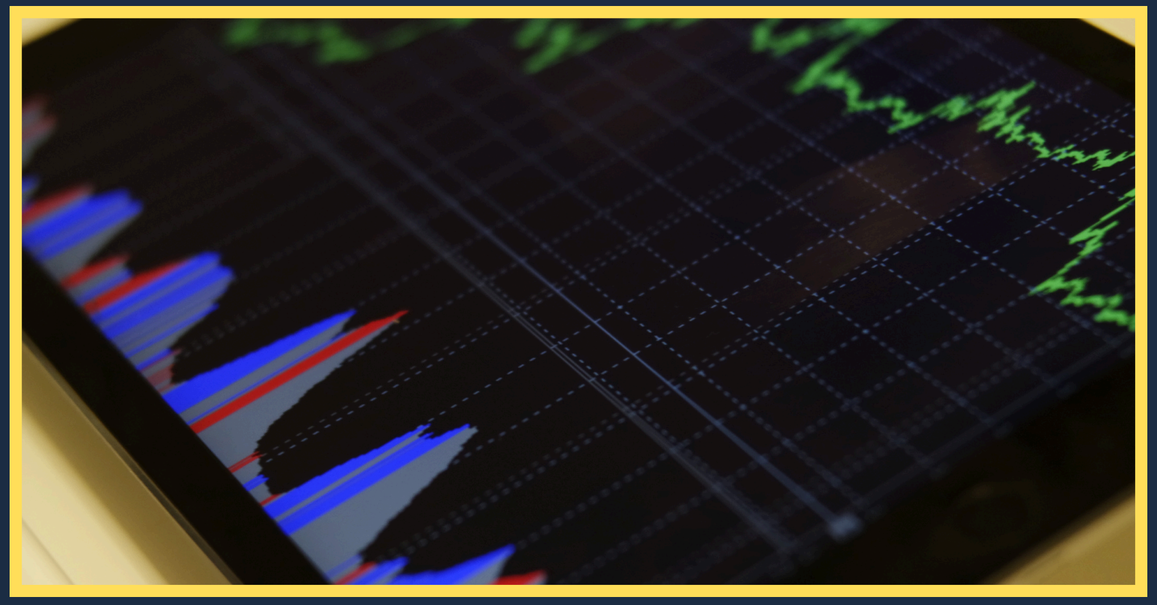
RFM-BASED ONLINE RETAIL SEGMENTATION

Project by Hernandia Zanua Leras

Background

In the realm of online retail, gaining insight into customer behavior is pivotal for boosting sales and nurturing high-value clientele. Transactional datasets capture critical details such as purchase timing, transaction frequency, and customer spending patterns, which serve as valuable inputs for behavioral analysis. A widely adopted approach for this purpose is the RFM framework—Recency, Frequency, and Monetary—used to categorize customers based on how recently they made a purchase (recency), how often they engage in transactions (frequency), and the total monetary value of their purchases (monetary).

Project Overview



1. Objective

- Identify high-value customers by analyzing purchase patterns to determine which customers contribute the most to revenue.
- Segment customers based on behavior using Recency, Frequency, and Monetary (RFM) metrics to tailor marketing strategies effectively.
- Enhance customer retention and engagement by understanding transactional habits and targeting interventions to improve loyalty and repeat purchases.

2. Tools

- Python (A programming language used for data processing, analysis, and automation, commonly applied in data analytics and data driven projects.)
- VSCode (A lightweight code editor used to write, run, and manage Python scripts efficiently with extensions that support data analysis workflows.)

Dataset Description

01

Dataset

Online Retail Data

02

Dataset Size

461774 Rows, 7 Columns

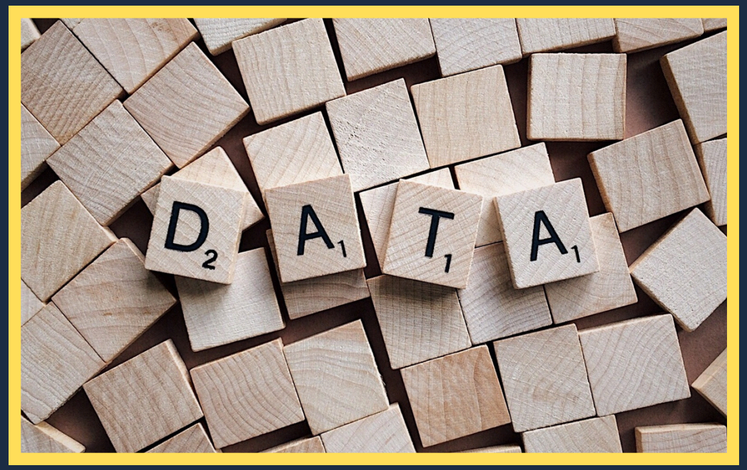
02

Goal

The goal is to understand customer behavior through RFM analysis in order to segment, retain, and maximize value from high-potential customers.”

Dataset Description

- **order_id** : A unique identifier assigned to each customer transaction or purchase.
- **product_code** : A unique code used to identify each product sold.
- **product_name** : The name or description of the product purchased by the customer.
- **quantity** : The number of product units purchased in a single transaction. Negative values indicate returns or order cancellations.
- **order_date** : The date and time when the product order was placed.
- **price** : The unit price of the product at the time of the transaction.
- **customer_id** : A unique identifier for each customer, used to track and analyze customer behavior.



Stages of User Cohort Segmentation

1. Import data from CSV into a DataFrame

```
import pandas as pd
import numpy as np
import datetime as dt
```

✓ 0.0s

```
import os
os.getcwd()
```

✓ 0.0s

- Import Online Retail transaction data from a CSV file into a DataFrame using pandas
- Load pandas, numpy, and datetime libraries for data processing and date handling
- Use the os library to check the current working directory and ensure the correct file path

Stages of User Cohort Segmentation

1. Import data from CSV into a DataFrame

```
df = pd.read_csv('Online Retail Data.csv', header=0)
df
```

✓ 0.9s

	order_id	product_code	product_name	quantity	order_date	price	customer_id
0	493410	TEST001	This is a test product.	5	2010-01-04 09:24:00	4.50	12346.0
1	C493411	21539	RETRO SPOTS BUTTER DISH	-1	2010-01-04 09:43:00	4.25	14590.0
2	493412	TEST001	This is a test product.	5	2010-01-04 09:53:00	4.50	12346.0
3	493413	21724	PANDA AND BUNNIES STICKER SHEET	1	2010-01-04 09:54:00	0.85	NaN
4	493413	84578	ELEPHANT TOY WITH BLUE T-SHIRT	1	2010-01-04 09:54:00	3.75	NaN
...	...	...	...	...	...	...	...
461768	539991	21618	4 WILDFLOWER BOTANICAL CANDLES	1	2010-12-23 16:49:00	1.25	NaN
461769	539991	72741	GRAND CHOCOLATECANDLE	4	2010-12-23 16:49:00	1.45	NaN
461770	539992	21470	FLOWER VINE RAFFIA FOOD COVER	1	2010-12-23 17:41:00	3.75	NaN
461771	539992	22258	FELT FARM ANIMAL RABBIT	1	2010-12-23 17:41:00	1.25	NaN
461772	539992	21155	RED RETROSPOT PEG BAG	1	2010-12-23 17:41:00	2.10	NaN

461773 rows × 7 columns

- Load the CSV file into a pandas DataFrame with the first row defined as column headers
- Inspect the dataset structure, which contains 461,773 rows and 7 transaction-related columns

Stages of User Cohort Segmentation

1. Import data from CSV into a DataFrame

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 461773 entries, 0 to 461772
Data columns (total 7 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   order_id      461773 non-null  object
 1   product_code  461773 non-null  object
 2   product_name  459055 non-null  object
 3   quantity      461773 non-null  int64
 4   order_date    461773 non-null  object
 5   price         461773 non-null  float64
 6   customer_id   360853 non-null  float64
dtypes: float64(2), int64(1), object(4)
memory usage: 24.7+ MB
```

- The dataset contains 461,773 transaction records with 7 columns
- Missing values exist in the product_name and customer_id fields
- Some columns such as order_date require data type adjustment before analysis

Stages of User Cohort Segmentation

2. Data Cleansing

```
# Copy the original dataframe
df_clean = df.copy()

# Process order date
df_clean['date'] = pd.to_datetime(df_clean['order_date']).dt.floor('D')

# Remove incomplete data
df_clean = df_clean[~df_clean['customer_id'].isna()]
df_clean = df_clean[~df_clean['product_name'].isna()]

# Standardize product_name and remove 'test' products
df_clean['product_name'] = df_clean['product_name'].str.lower()
df_clean = df_clean[
    (~df_clean['product_code'].str.lower().str.contains('test')) |
    (~df_clean['product_name'].str.contains('test '))
]
```

- Copy the original dataframe to preserve raw data.
- Convert order_date to date only for easier daily analysis.
- Remove rows with missing customer_id or product_name.
- Standardize product names (lowercase) and remove test products.

```
# Create order_status column
df_clean['order_status'] = np.where(df_clean['order_id'].str[:1] == 'C', 'cancelled', 'delivered')

# Correct quantity and price
df_clean['quantity'] = df_clean['quantity'].abs()
df_clean = df_clean[df_clean['price'] > 0]

# Create amount column
df_clean['amount'] = df_clean['quantity'] * df_clean['price']
```

- Determine order_status from order_id prefix.
- Correct negative quantity values and remove non-positive prices.
- Calculate amount as quantity × price.

Stages of User Cohort Segmentation

2. Data Cleansing

```
# Standardize product_name by product_code
most_freq_product_name = (
    df_clean.groupby(['product_code', 'product_name'], as_index=False)
    .agg(order_cnt=('order_id', 'nunique'))
    .sort_values(['product_code', 'order_cnt'], ascending=[True, False])
)
most_freq_product_name['rank'] = (
    most_freq_product_name.groupby('product_code')['order_cnt']
    .rank(method='first', ascending=False)
)
most_freq_product_name = most_freq_product_name[most_freq_product_name['rank'] == 1].drop(
    columns=['order_cnt', 'rank']
)
df_clean = df_clean.merge(
    most_freq_product_name.rename(columns={'product_name': 'most_freq_product_name'}),
    how='left',
    on='product_code'
)
df_clean['product_name'] = df_clean['most_freq_product_name']
df_clean = df_clean.drop(columns='most_freq_product_name')
```

- Some products share the same product_code but have different product_name due to variations like typos or capitalization.
- We group the data by product_code and product_name to count the frequency of each name.
- Then, we replace all variations with the most frequent name to ensure consistency across the dataset.

```
# Convert customer_id to string
df_clean['customer_id'] = df_clean['customer_id'].astype(str)

# Remove outliers
df_clean = df_clean[(np.abs(stats.zscore(df_clean[['quantity', 'amount']])) < 3).all(axis=1)]

# Reset index
df_clean = df_clean.reset_index(drop=True)

df_clean
```

- Convert customer_id to string for consistency.
- Remove outliers in quantity and amount (Z-score < 3) to avoid biased analysis.
- Reset index after filtering rows.

Stages of User Cohort Segmentation

2. Data Cleansing

```
df_clean.info()
✓ 1.1s
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 358469 entries, 0 to 358468
Data columns (total 10 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   order_id      358469 non-null  object
 1   product_code  358469 non-null  object
 2   product_name  358469 non-null  object
 3   quantity      358469 non-null  int64
 4   order_date    358469 non-null  object
 5   price         358469 non-null  float64
 6   customer_id   358469 non-null  object
 7   date          358469 non-null  datetime64[ns]
 8   order_status  358469 non-null  object
 9   amount        358469 non-null  float64
dtypes: datetime64[ns](1), float64(2), int64(1), object(6)
memory usage: 27.3+ MB
```

- Dataset has 358,469 rows and 10 columns with no missing values.
- Key columns include order_id, product_code, product_name, quantity, price, customer_id, date, order_status, and amount.
- All data types are properly set (e.g., date as datetime, quantity as int, price & amount as float) and memory usage is ~27 MB.

Stages of User Cohort Segmentation

3. User-Level Transaction Summary: Total Orders, Total Order Value, and Last Order Date

```
df_user = (
    df_clean
    .groupby('customer_id', as_index=False)
    .agg(
        order_cnt=('order_id', 'nunique'),
        max_order_date=('date', 'max'),
        total_order_value=('amount', 'sum')
    )
)
df_user
```

✓ 0.3s

	customer_id	order_cnt	max_order_date	total_order_value
0	12346.0	5	2010-10-04	602.40
1	12608.0	1	2010-10-31	415.79
2	12745.0	2	2010-08-10	723.85
3	12746.0	2	2010-06-30	266.35
4	12747.0	19	2010-12-13	4094.79
...	...	...	...	...
3884	18283.0	6	2010-11-22	641.77
3885	18284.0	2	2010-10-06	486.68
3886	18285.0	1	2010-02-17	427.00
3887	18286.0	2	2010-08-20	941.48
3888	18287.0	4	2010-11-22	2345.71

3889 rows × 4 columns

- Data is grouped by customer_id, so each row represents a unique customer.
- order_cnt shows the number of unique transactions, max_order_date is the latest transaction date, and total_order_value is the sum of all purchase amounts per customer.
- The output contains 3,889 unique customers with 4 columns, ready for loyalty and customer segmentation analysis.

Stages of User Cohort Segmentation

4. Days Since Last Order

```
today
```
✓ 0.0s

```
Timestamp('2010-12-23 00:00:00')
```

```
today = df_clean['date'].max()
df_user['day_since_last_order'] = (today - df_user['max_order_date']).dt.days
df_user
```
✓ 0.0s

	customer_id	order_cnt	max_order_date	total_order_value	day_since_last_order
0	12346.0	5	2010-10-04	602.40	80
1	12608.0	1	2010-10-31	415.79	53
2	12745.0	2	2010-08-10	723.85	135
3	12746.0	2	2010-06-30	266.35	176
4	12747.0	19	2010-12-13	4094.79	10
...	...	...	...	...	...
3884	18283.0	6	2010-11-22	641.77	31
3885	18284.0	2	2010-10-06	486.68	78
3886	18285.0	1	2010-02-17	427.00	309
3887	18286.0	2	2010-08-20	941.48	125
3888	18287.0	4	2010-11-22	2345.71	31

3889 rows × 5 columns

- The dataset identifies the most recent transaction date as a reference point (“today”) and calculates day_since_last_order for each customer.
- This metric reflects customer recency: smaller values indicate recent activity, while larger values show inactivity over a longer period.
- The resulting output contains 3,889 customers across 5 columns, providing a structured foundation for RFM and churn analysis.

Stages of User Cohort Segmentation

Customer Activity Summary

```
df_user.describe()
```
✓ 0.0s

	order_cnt	max_order_date	total_order_value	day_since_last_order
count	3889.000000	3889	3889.000000	3889.000000
mean	5.128568	2010-09-23 04:46:57.793777664	1544.623084	90.800720
min	1.000000	2010-01-05 00:00:00	1.250000	0.000000
25%	1.000000	2010-08-19 00:00:00	296.360000	25.000000
50%	3.000000	2010-10-26 00:00:00	648.200000	58.000000
75%	6.000000	2010-11-28 00:00:00	1585.940000	126.000000
max	163.000000	2010-12-23 00:00:00	71970.390000	352.000000
std	8.499330	NaN	3434.816315	88.873286

- On average, customers make about 5 transactions (median 3), with a maximum of 163 orders, highlighting the presence of highly loyal customers.
- The average spending per customer is around 1,545, though the high standard deviation indicates significant variation in purchase behavior.
- The median recency is 57 days, with a range from 0 to 352 days, showing that most customers have not purchased in the past 1–2 months and that activity levels vary widely across the base.

Stages of User Cohort Segmentation

5. Customer Recency Binning

```
df_user['recency_score'] = pd.cut(df_user['day_since_last_order'],
                                  bins=[df_user['day_since_last_order'].min(),
                                        np.percentile(df_user['day_since_last_order'], 20),
                                        np.percentile(df_user['day_since_last_order'], 40),
                                        np.percentile(df_user['day_since_last_order'], 60),
                                        np.percentile(df_user['day_since_last_order'], 80),
                                        df_user['day_since_last_order'].max()],
                                  labels=[5, 4, 3, 2, 1],
                                  include_lowest=True).astype(int)
df_user
```

✓ 0.0s

	customer_id	order_cnt	max_order_date	total_order_value	day_since_last_order	recency_score
0	12346.0	5	2010-10-04	602.40	80	2
1	12608.0	1	2010-10-31	415.79	53	3
2	12745.0	2	2010-08-10	723.85	135	2
3	12746.0	2	2010-06-30	266.35	176	1
4	12747.0	19	2010-12-13	4094.79	10	5
...	...	...	...	...	...	...
3884	18283.0	6	2010-11-22	641.77	31	4
3885	18284.0	2	2010-10-06	486.68	78	2
3886	18285.0	1	2010-02-17	427.00	309	1
3887	18286.0	2	2010-08-20	941.48	125	2
3888	18287.0	4	2010-11-22	2345.71	31	4

3889 rows × 6 columns

- The code uses pd.cut to divide the column day_since_last_order into 5 bins, based on the minimum, percentiles (20, 40, 60, 80), and maximum values.
- Each bin is assigned a recency score from 5 (most recent) down to 1 (least recent), reflecting how recently a customer made their last order.
- The result is stored in a new column recency_score, converted to integers for easier analysis.
- This scoring system provides a structured way to measure customer recency for RFM or churn analysis

Stages of User Cohort Segmentation

6. Customer Frequency Binning

```
df_user['frequency_score'] = pd.cut(df_user['order_cnt'],
                                    bins=[0,
                                          np.percentile(df_user['order_cnt'], 20),
                                          np.percentile(df_user['order_cnt'], 40),
                                          np.percentile(df_user['order_cnt'], 60),
                                          np.percentile(df_user['order_cnt'], 80),
                                          df_user['order_cnt'].max()],
                                    labels=[1, 2, 3, 4, 5],
                                    include_lowest=True).astype(int)
df_user
```

✓ 0.0s

	customer_id	order_cnt	max_order_date	total_order_value	day_since_last_order	recency_score	frequency_score
0	12346.0	5	2010-10-04	602.40	80	2	4
1	12608.0	1	2010-10-31	415.79	53	3	1
2	12745.0	2	2010-08-10	723.85	135	2	2
3	12746.0	2	2010-06-30	266.35	176	1	2
4	12747.0	19	2010-12-13	4094.79	10	5	5
...	...	...	...	...	...	...	...
3884	18283.0	6	2010-11-22	641.77	31	4	4
3885	18284.0	2	2010-10-06	486.68	78	2	2
3886	18285.0	1	2010-02-17	427.00	309	1	1
3887	18286.0	2	2010-08-20	941.48	125	2	2
3888	18287.0	4	2010-11-22	2345.71	31	4	3

3889 rows × 7 columns

- The code segments customers into 5 frequency bins based on their total number of orders, using percentile cutoffs (20, 40, 60, 80) plus minimum and maximum values.
- Each bin is assigned a frequency score from 1 to 5, where higher scores represent customers with more transactions.
- The new column frequency_score provides a structured metric to evaluate customer purchasing frequency for RFM or churn analysis.

Stages of User Cohort Segmentation

7. Customer Monetary Binning

```
df_user['monetary_score'] = pd.cut(df_user['total_order_value'],
                                   bins=[df_user['total_order_value'].min(),
                                         np.percentile(df_user['total_order_value'], 20),
                                         np.percentile(df_user['total_order_value'], 40),
                                         np.percentile(df_user['total_order_value'], 60),
                                         np.percentile(df_user['total_order_value'], 80),
                                         df_user['total_order_value'].max()],
                                   labels=[1, 2, 3, 4, 5],
                                   include_lowest=True).astype(int)
df_user
```

✓ 0.3s

	customer_id	order_cnt	max_order_date	total_order_value	day_since_last_order	recency_score	monetary_score
0	12346.0	5	2010-10-04	602.40	80	2	3
1	12608.0	1	2010-10-31	415.79	53	3	2
2	12745.0	2	2010-08-10	723.85	135	2	3
3	12746.0	2	2010-06-30	266.35	176	1	2
4	12747.0	19	2010-12-13	4094.79	10	5	5
...	...	...	...	...	...	...	...
3884	18283.0	6	2010-11-22	641.77	31	4	3
3885	18284.0	2	2010-10-06	486.68	78	2	3
3886	18285.0	1	2010-02-17	427.00	309	1	2
3887	18286.0	2	2010-08-20	941.48	125	2	4
3888	18287.0	4	2010-11-22	2345.71	31	4	5

3889 rows × 7 columns

- The code divides total_order_value into 5 bins using percentile cutoffs (20, 40, 60, 80) along with minimum and maximum values.
- Each bin is assigned a monetary score from 1 to 5, where higher scores represent customers with greater spending.
- The new column monetary_score provides a structured measure of customer value, useful for RFM and churn analysis.

Stages of User Cohort Segmentation

8. Customer Monetary Binning

```
df_user['monetary_score'] = pd.cut(df_user['total_order_value'],
                                   bins=[df_user['total_order_value'].min(),
                                         np.percentile(df_user['total_order_value'], 20),
                                         np.percentile(df_user['total_order_value'], 40),
                                         np.percentile(df_user['total_order_value'], 60),
                                         np.percentile(df_user['total_order_value'], 80),
                                         df_user['total_order_value'].max()],
                                   labels=[1, 2, 3, 4, 5],
                                   include_lowest=True).astype(int)
df_user
```

✓ 0.3s

	customer_id	order_cnt	max_order_date	total_order_value	day_since_last_order	recency_score	monetary_score
0	12346.0	5	2010-10-04	602.40	80	2	3
1	12608.0	1	2010-10-31	415.79	53	3	2
2	12745.0	2	2010-08-10	723.85	135	2	3
3	12746.0	2	2010-06-30	266.35	176	1	2
4	12747.0	19	2010-12-13	4094.79	10	5	5
...	...	...	...	...	...	...	...
3884	18283.0	6	2010-11-22	641.77	31	4	3
3885	18284.0	2	2010-10-06	486.68	78	2	3
3886	18285.0	1	2010-02-17	427.00	309	1	2
3887	18286.0	2	2010-08-20	941.48	125	2	4
3888	18287.0	4	2010-11-22	2345.71	31	4	5

3889 rows × 7 columns

- The code divides total_order_value into 5 bins using percentile cutoffs (20, 40, 60, 80) along with minimum and maximum values.
- Each bin is assigned a monetary score from 1 to 5, where higher scores represent customers with greater spending.
- The new column monetary_score provides a structured measure of customer value, useful for RFM and churn analysis.

Stages of User Cohort Segmentation

9. Customer Segmentation by Recency and Frequency

```
df_user['segment'] = np.select(
    [(df_user['recency_score']==5) & (df_user['frequency_score']>=4),
     (df_user['recency_score'].between(3, 4)) & (df_user['frequency_score']>=4),
     (df_user['recency_score']>=4) & (df_user['frequency_score'].between(2, 3)),
     (df_user['recency_score']<=2) & (df_user['frequency_score']==5),
     (df_user['recency_score']==3) & (df_user['frequency_score']==3),
     (df_user['recency_score']==5) & (df_user['frequency_score']==1),
     (df_user['recency_score']==4) & (df_user['frequency_score']==1),
     (df_user['recency_score']<=2) & (df_user['frequency_score'].between(3, 4)),
     (df_user['recency_score']==3) & (df_user['frequency_score']<=2),
     (df_user['recency_score']<=2) & (df_user['frequency_score']<=2)],
    ['01-Champion', '02-Loyal Customers', '03-Potential Loyalists', "04-Can't Lose Them", '05-Need Attention',
     '06-New Customers', '07-Promising', '08-At Risk', '09-About to Sleep', '10-Hibernating']
)
df_user
```

✓ 0.0s

	customer_id	order_cnt	max_order_date	total_order_value	day_since_last_order	recency_score	frequency_score	monetary_score	segment
0	12346.0	5	2010-10-04	602.40	80	2	4	3	08-At Risk
1	12608.0	1	2010-10-31	415.79	53	3	1	2	09-About to Sleep
2	12745.0	2	2010-08-10	723.85	135	2	2	3	10-Hibernating
3	12746.0	2	2010-06-30	266.35	176	1	2	2	10-Hibernating
4	12747.0	19	2010-12-13	4094.79	10	5	5	5	01-Champion
...	...	...	...	...	...	...	...	...	...
3884	18283.0	6	2010-11-22	641.77	31	4	4	3	02-Loyal Customers
3885	18284.0	2	2010-10-06	486.68	78	2	2	3	10-Hibernating
3886	18285.0	1	2010-02-17	427.00	309	1	1	2	10-Hibernating
3887	18286.0	2	2010-08-20	941.48	125	2	2	4	10-Hibernating
3888	18287.0	4	2010-11-22	2345.71	31	4	3	5	03-Potential Loyalists

3889 rows × 9 columns

- The code uses np.select to assign customers into segments based on combinations of their recency_score and frequency_score, with different logical conditions defining groups like recent and frequent buyers, inactive but frequent buyers, or low recency and low frequency customers.
- Each condition is mapped to a descriptive segment label (e.g., Champion, Loyal Customers, At Risk, Hibernating) and stored in a new column called segment for structured RFM and churn analysis.
- Customer ID 12747.0 is classified as '01-Champion' due to achieving a perfect RFM score with high recency and frequency.
- Customer ID 18285.0 falls into the '10-Hibernating' segment based on 309 days of inactivity and low RFM scores.

Stages of User Cohort Segmentation

10. Summary RFM Segmentation

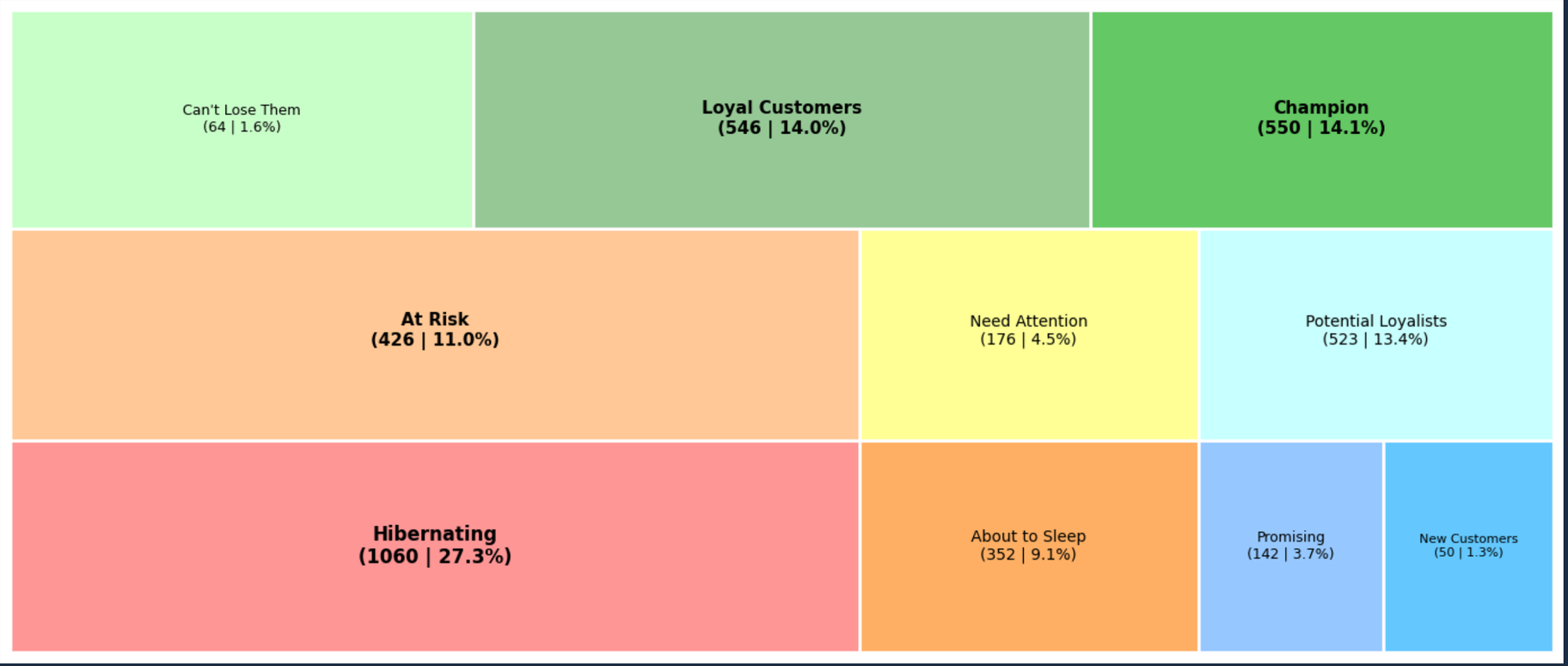
```
summary = pd.pivot_table(df_user, index='segment',
                         values=['customer_id','day_since_last_order','order_cnt','total_order_value'],
                         aggfunc={'customer_id': pd.Series.nunique,
                                  'day_since_last_order': [np.mean, np.median],
                                  'order_cnt': [np.mean, np.median],
                                  'total_order_value': [np.mean, np.median]})
summary['pct_unique'] = (summary['customer_id'] / summary['customer_id'].sum() * 100).round(1)
summary
```

✓ 0.0s

	customer_id	day_since_last_order		order_cnt		total_order_value		pct_unique
	nunique	mean	median	mean	median	mean	median	
segment								
01-Champion	550	10.618182	9.5	15.467273	10.0	5003.674245	2775.525	14.1
02-Loyal Customers	546	40.864469	37.0	8.767399	7.0	2622.817826	1946.850	14.0
03-Potential Loyalists	523	23.573614	24.0	2.829828	3.0	766.769828	622.070	13.4
04-Can't Lose Them	64	121.984375	112.5	11.375000	9.5	2839.948125	2268.405	1.6
05-Need Attention	176	58.613636	59.0	3.397727	3.0	989.232676	826.370	4.5
06-New Customers	50	14.220000	16.0	1.000000	1.0	244.689000	193.675	1.3
07-Promising	142	32.760563	34.0	1.000000	1.0	287.800282	238.440	3.7
08-At Risk	426	140.455399	120.0	4.136150	4.0	1153.825683	875.430	11.0
09-About to Sleep	352	58.735795	58.0	1.417614	1.0	448.229688	334.755	9.1
10-Hibernating	1060	196.837736	199.0	1.313208	1.0	343.083842	257.005	27.3

- Aggregates customer metrics by RFM segment, showing distribution and behavior patterns across 10 customer groups.
- Champions are the most valuable (14.1%, 550 customers) with highest order frequency (15.5) and value ($5,003 avg).
- Hibernating customers dominate by volume (27.3%, 1,060 customers) but show 196 days inactivity and lowest engagement.
- Over 38% fall into problematic segments (At Risk, About to Sleep, Hibernating), indicating urgent need for re-engagement strategies.

RFM Segmentation Visualization



CONCLUSIONS

MAJORITY OF CUSTOMERS ARE INACTIVE

Hibernating segment dominates with 27.3% (1,060 customers) who haven't transacted in an average of 196 days, representing the largest single segment and significant revenue loss potential.

HIGH-VALUE SEGMENTS ARE RELATIVELY BALANCED

Champions (14.1%, 550 customers) and Loyal Customers (14.0%, 546 customers) are nearly equal in size, together representing 28.1% of the customer base with the highest revenue per customer.

47.4% OF CUSTOMERS NEED URGENT INTERVENTION

Combined problematic segments (Hibernating 27.3%, At Risk 11.0%, About to Sleep 9.1%) indicate nearly half the customer base is at risk of permanent churn.

GROWTH OPPORTUNITY EXISTS IN MID-TIER SEGMENTS

Potential Loyalists (13.4%, 523 customers) and Promising (3.7%, 142 customers) show potential to be upgraded to higher-value segments with proper engagement.

SUGGESTIONS

LAUNCH AGGRESSIVE RE-ENGAGEMENT CAMPAIGN FOR HIBERNATING CUSTOMERS

Deploy personalized win-back offers, discount codes, or exclusive access to reactivate 1,060 dormant customers and prevent permanent churn.

DEVELOP TRANSACTION VOLUME PROGRAMS FOR POTENTIAL LOYALISTS

Introduce bundle deals, volume discounts, or subscription models to help 523 Potential Loyalists increase order frequency and move toward Champion segment.nia elit.

IMPLEMENT FREQUENCY-BOOSTING INITIATIVES FOR LOYAL CUSTOMERS

Create urgency-driven promotions, flash sales, or time-limited rewards to encourage 546 Loyal Customers to transact more frequently and upgrade to Champion status

DEPLOY IMMEDIATE RETENTION TACTICS FOR AT RISK SEGMENT

Provide VIP treatment, personal outreach, or exclusive offers to 426 At Risk customers who have high historical value but show 140+ days of inactivity to prevent further deterioration.

Let’s Connect

083109376438

hernandiazanua@gmail.com

HERNANDIA ZANUA LERAS

Thank You