2025

Customer Segmentation Analysis Report

Rye Alyanna Po Ngo Seng (吴明明)

Table of Contents

Introduction

Project Context

The "Automobile Retail Customer Segmentation" dataset, available on Kaggle, offers a comprehensive view into customer segmentation tailored for targeted marketing within the automobile retail sector, specifically the bike/ bicycle sector. This dataset is designed to facilitate detailed analyses of customer behaviors and preferences, enabling businesses to identify distinct customer segments and tailor their marketing strategies accordingly.

Project Goal

The primary goal of this project is to develop a robust and scalable customer segmentation framework that can inform and guide future marketing strategies. The methodology is designed not just to analyze the current customer base, but to create a predictive tool applicable to new and future customers.

The project includes two key stages:

1. Unsupervised Clustering: First, natural customer groupings are identified by applying clustering algorithms to a combination of transactional and demographic data. This reveals the inherent structure and segments within the sample data.
2. Predictive Classification: Following clustering, a classification model is built to predict which segment a customer belongs to. The crucial objective here is to ensure the segmentation model is generalizable and sustainable over time. To achieve this, the model is trained in a strategic selection of features: both those that are highly correlated with the clusters and, critically, those proven to be statistically significant. This approach of using both correlated and statistically significant features ensures the predictive model is founded on stable, reliable differentiators, enhancing its robustness and making it a powerful tool for applying personalized marketing efforts to all future customers.

Evaluation Criteria for Data Analysis

To evaluate the effectiveness of the segmentation and subsequent classification, a series of evaluation metrics is adopted. These metrics are used at different stages of the workflow to assess feature significance, clustering relevance, and classification performance:

1. Chi-square Test
   Used to evaluate the independence between categorical features and cluster labels. A low p-value from Chi-square statistic indicates a strong association, helping to identify features that significantly differ across clusters.
2. ANOVA (Analysis of Variance)
   Also to test whether there are statistically significant differences in the means of a feature across clusters. The difference with Chi-square is that features applied need to be numerical. A low p-value from ANOVA suggests that the feature varies meaningfully between groups.
3. Mutual Information Score

Measures the amount of shared information between a feature and the cluster labels. Higher mutual information indicates a stronger dependency, making the feature more relevant for classification.

4. Classification Report
   Provides a detailed breakdown of classification performance by presenting several key components for each class. It helps evaluate how well the classification model distinguishes among the clusters.
   (1) Precision: The ratio of true positives to the sum of true positives and false positives. It measures how accurate the model's positive predictions are.
   (2) Recall (Sensitivity): The ratio of true positives to the sum of true positives and false negatives (actual instances of the class that the model failed to identify). It indicates how well the model captures all actual positives.
   (3) F1-Score: The mean of precision and recall, combining both false positives and false negatives into a single metric to balance the trade-off between them.
   (4) Support: The total number of actual occurrences of each class in the dataset, showing the true distribution of the classes.
5. Accuracy Score
   The ratio of the sum of true positives and true negatives to the total number of predictions. It gives a general sense of overall model performance.
6. ROC AUC Score
   Measures the model's ability to differentiate between classes. It plots the True Positive Rate against the False Positive Rate at various threshold levels. A higher ROC AUC score reflects better classification performance, especially in imbalanced datasets.

By employing these evaluation metrics, this project aims to ensure both the validity of the unsupervised clustering and the reliability of the following classification models, eventually supporting the goal of deriving actionable insights from customer data for more targeted and effective marketing strategies.

Data Understanding

Data Source & Columns Description

The dataset used for this project was sourced from Kaggle, AUTOMOBILE RETAIL CUSTOMER SEGMENTATION. It is classified under the Apache 2.0 license, making it freely available for public use. The data includes one Excel file with 4 tables in it.

The first table is `Transactions`, rename as transactions. There's a total of 20,000 records. Description for each column is stated in the table:

transaction_id	int	The ID number of the transaction record.
product_id	int	The ID number of the product purchased during the transaction. For each product, there are different brands, lines, classes and sizes.
customer_id	int	The ID number of the customer who made the transaction.
transaction_date	datetime	Date of the transaction made. Initially int (yyyy-mm-dd hh:mm:ss), is converted to datetime during the preprocessing stage.
online_order	int	Whether or not the transaction is done online. 1 for yes, and 0 for no.
order_status	string	Whether or not the transaction is approved. Approved or Cancelled.
brand	string	The brand of the product. There are 6 brands: 1. Solex 2. Trek Bicycles 3. OHM Cycles 4. Norco Bicycles 5. Giant Bicycles 6. WeareA2B
product_line	string	The type of the product, based on its intended use or design focus. 1. Standard (general purpose) 2. Road (for paved surface and speed) 3. Mountain 4. Touring (for long distance travel)
product_class	string	The class of the product, based on its quality or performance level. 1. low 2. medium 3. high

product_size	string	The size of the product. 1. small 2. medium 3. large
product	string	The combination of the `brand`, `product_id`, `product_line`, `product_class`, and `product_size`. There are 188 kinds of unique products. Created during the preprocessing stage.
list_price	float	The suggested retail price of the unique product.
standard_cost	float	The predetermined cost of the unique product under normal conditions.
margin_profit	float	The gross profit margin per unit of the product. Calculated between `list_price` and `standard_cost` during the preprocessing stage.
product_first_sold_date	datetime	The date when the customer first buys the unique product. Converted to datetime during preprocessing stage.

The `CustomerAddress`, renamed as customer_address. There's a total of 3,999 records. Description for each column is stated in the table:

customer_id	int	The ID number of the customer.
address	string	The street where the customer lives in.
postcode	string	The postcode of the address where the customer is staying. It is converted from int to string in the preprocessing stage.
state	string	The state of the country where the customer lives in.
country	string	The country where the customer is staying. This dataset only contains one value—Australia.
property_valuation	int	The assessed value of the property where the customer lives in, ranging from 1 to 12.

The `CustomerDemographic` table records the demographical data from customers in `CustomerAddress`. It is renamed as customer_demographic. There's a total of 4,000 records. Description for each column is stated below:

customer_id	int	The ID number of the customer.

first_name	string	The first name (given name) of the customer.
last_name	string	The last name (surname / family name) of the customer.
full_name	string	The merged column of `first_name` and `last_name`, created during preprocessing stage.
gender	string	The gender of the customer, Male or Female.
past_3_years_bike_related_purchases	int	The total quantity of bike related purchase events of the customer for the past three years.
DOB	datetime	Date of Birth of the customer. Initially int (yyyy-mm-dd hh:mm:ss), is converted to datetime during the preprocessing stage.
decade	string	The decade when the customer is born, created based on `DOB` during preprocessing stage. 4. 1930s 5. 1940s 6. 1950s 7. 1960s 8. 1970s 9. 1980s 10. 1990s 11. 2000s
job_title	string	The job position of the customer, 196 variations.
job_category	string	The grouped category for `job_title`, created based on `job_title` during preprocessing stage. 43 variations including: Administrative, System Analyst, Engineer, Developer, Accountant, Junior Executive, Middle Manager, Field Specialist, Researcher, Editor, Nurse, Legal, Social Worker, Media, Coordinator, Chemist, Executive Manager, Data Analyst, Sales, Operator, Statistician, Auditor, Therapist, Pathologist, Quality Engineer, Community Services, Teacher, Professor, Designer, Programmer, Human Resources, Administrator, Recruiter, Librarian, Pharmacist, Low Manager, Geologist, IT Support, Actuary, Writer, Marketing, Scientist, Software Engineer
Job_industry_category	string	The industry where the customer is working in. 9 variations including: Health, Financial Services, Argiculture, Manufacturing,

		Telecommunications, Entertainment, Retail, Property, IT
wealth_segment	string	The financial standing and wealth level of the customer 1. Mass Customer 2. Affluent Customer 3. High Net Worth
deceased_indicator	string	Whether the customer is deceased. N for No means still alive; Y for Yes means no longer alive.
default	string	Cannot be identified as the values are random characters, will be deleted during the preprocessing stage.
owns_car	string	Whether or not the customer owns a car, Yer or No.
tenure	float	The number of years the customer has spent with the company, ranging from 1 to 22.

The `NewCustomerList` table has a total of 1,000 records. Since the tenure data is also ranging from 1 year to 22 years, indicating this table is not recording new customer data. Combined with the fact that there are two columns named Rank and Value, I assumed this table was cleaned and labeled based on specific characteristics of existing customers, and named it cleaned_customer. Description for each column is stated below:

first_name	string	The first name (given name) of the customer.
last_name	string	The last name (surname / family name) of the customer.
full_name	string	The merged column of `first_name` and `last_name`, created during preprocessing stage.
gender	string	The gender of the customer, Male or Female.
past_3_years_bike_related_purchases	int	The total quantity of bike related purchase events of the customer for the past three years.
DOB	datetime	Date of Birth of the customer. Initially int (yyyy-mm-dd hh:mm:ss), is converted to datetime during the preprocessing stage.
job_title	string	The job position of the customer, 196 variations.
Job_industry_category	string	The industry where the customer is working in. 9 variations including: Health, Financial Services, Argiculture, Manufacturing, Telecommunications, Entertainment, Retail, Property, IT

wealth_segment	string	The financial standing and wealth level of the customer 4. Mass Customer 5. Affluent Customer 6. High Net Worth
deceased_indicator	string	Whether the customer is deceased. N for No means still alive; Y for Yes means no longer alive.
owns_car	string	Whether or not the customer owns a car, Yer or No.
tenure	float	The number of years the customer has spent with the company, ranging from 1 to 22.
Rank	int	The raking of the customer based on their `Value`. The higher the `Value`, the smaller the number of rank is.
Unnamed:16 Unnamed:17 Unnamed:18 Unnamed:19 Unnamed:20	Float	Cannot be identified, Will be deleted during the preprocessing stage.
Value	float	The calculated value the customer based on their characteristic.
address	string	The street where the customer lives in.
postcode	string	The postcode of the address where the customer is staying. It is converted from int to string in the preprocessing stage.
state	string	The state of the country where the customer lives in.
country	string	The country where the customer is staying. This dataset only contains one value—Australia.
property_valuation	int	The assessed value of the property where the customer lives in, ranging from 1 to 12.

cleaned_customer has similar columns to customer_demographic and customer_address, I decided to merge them based on name and gender to identify whether it has any sort of relation. As below indicates, cleaned_customer is a complete independent file to customer_demographic.

```
merged_df = pd.merge(customer_demographic, new_customer, on=['first_name', 'last_name', 'gender'], how='inner', \
                     indicator=True, suffixes=['_cus', '_new'])
```

DOB_cus	DOB_new	job_title_cus	job_title_new	past_3_years_bike_related_purchases_cus	past_3_years_bike_related_purchases_new
1966-07-03 00:00:00	1945-02-13	Software Test Engineer III	General Manager	54	47

Here is the brief inspection of the 4 tables. Besides customer_address, the other 3 tables have missing values, handling them is necessary later before analyzing.

```
Customer Adress
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3999 entries, 0 to 3998
Data columns (total 6 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   customer_id          3999 non-null   int64
 1   address              3999 non-null   object
 2   postcode             3999 non-null   int64
 3   state                3999 non-null   object
 4   country              3999 non-null   object
 5   property_valuation   3999 non-null   int64
dtypes: int64(3), object(3)
memory usage: 187.6+ KB
None
```

```
Customer Demographic
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4000 entries, 0 to 3999
Data columns (total 13 columns):
 #   Column                               Non-Null Count  Dtype
---  ------                               --------------  -----
 0   customer_id                          4000 non-null   int64
 1   first_name                           4000 non-null   object
 2   last_name                            3875 non-null   object
 3   gender                               4000 non-null   object
 4   past_3_years_bike_related_purchases  4000 non-null   int64
 5   DOB                                  3913 non-null   object
 6   job_title                            3494 non-null   object
 7   job_industry_category                3344 non-null   object
 8   wealth_segment                       4000 non-null   object
 9   deceased_indicator                   4000 non-null   object
 10  default                              3698 non-null   object
 11  owns_car                             4000 non-null   object
 12  tenure                               3913 non-null   float64
dtypes: float64(1), int64(2), object(10)
memory usage: 406.4+ KB
None
```

```
Cleaned Customer
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 23 columns):
 #   Column                               Non-Null Count  Dtype
---  ------                               --------------  -----
 0   first_name                           1000 non-null   object
 1   last_name                            971 non-null    object
 2   gender                               1000 non-null   object
 3   past_3_years_bike_related_purchases  1000 non-null   int64
 4   DOB                                  983 non-null    object
 5   job_title                            894 non-null    object
 6   job_industry_category                835 non-null    object
 7   wealth_segment                       1000 non-null   object
 8   deceased_indicator                   1000 non-null   object
 9   owns_car                             1000 non-null   object
 10  tenure                               1000 non-null   int64
 11  address                              1000 non-null   object
 12  postcode                             1000 non-null   int64
 13  state                                1000 non-null   object
 14  country                              1000 non-null   object
 15  property_valuation                   1000 non-null   int64
 16  Unnamed: 16                          1000 non-null   float64
 17  Unnamed: 17                          1000 non-null   float64
 18  Unnamed: 18                          1000 non-null   float64
 19  Unnamed: 19                          1000 non-null   float64
 20  Unnamed: 20                          1000 non-null   int64
 21  Rank                                 1000 non-null   int64
 22  Value                                1000 non-null   float64
dtypes: float64(5), int64(6), object(12)
memory usage: 179.8+ KB
None
```

```
Transactions
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 13 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   transaction_id           20000 non-null  int64
 1   product_id               20000 non-null  int64
 2   customer_id              20000 non-null  int64
 3   transaction_date         20000 non-null  datetime64[ns]
 4   online_order             19640 non-null  float64
 5   order_status             20000 non-null  object
 6   brand                    19803 non-null  object
 7   product_line             19803 non-null  object
 8   product_class            19803 non-null  object
 9   product_size             19803 non-null  object
 10  list_price               20000 non-null  float64
 11  standard_cost            19803 non-null  float64
 12  product_first_sold_date  19803 non-null  float64
dtypes: datetime64[ns](1), float64(4), int64(3), object(5)
memory usage: 2.0+ MB
None
```

Handling Missing, Duplicated and Incorrect Records

Customer Related Tables

After dropping columns that cannot be interpreted or unnecessary (default, product first sold date, unnamed columns), I start with the missing values in each column. The first one is last_name. As last_name won't affect the clustering output, I decided to impute it by combining first_name.

```
#combine first name and last name as full name
customer_demographic['first_name'] = customer_demographic['first_name'].fillna('')
customer_demographic['last_name'] = customer_demographic['last_name'].fillna('')
customer_demographic['full_name'] = customer_demographic['first_name'] + ' ' + customer_demographic['last_name']
customer_demographic['full_name'] = customer_demographic['full_name'].str.strip()
customer_demographic =  customer_demographic.drop(['first_name', 'last_name'], axis=1)
display(customer_demographic.head())
print("\n")

cleaned_customer['first_name'] = cleaned_customer['first_name'].fillna('')
cleaned_customer['last_name'] = cleaned_customer['last_name'].fillna('')
cleaned_customer['full_name'] = cleaned_customer['first_name'] + ' ' + cleaned_customer['last_name']
cleaned_customer['full_name'] = cleaned_customer['full_name'].str.strip()
cleaned_customer =  cleaned_customer.drop(['first_name', 'last_name'], axis=1)
display(cleaned_customer.head())
```

Then the DOB column and the tenure column. There are only a very small percentage of missing values, so I decided to drop them then set the type of DOB to datetime. After dropping, there is a total of 3913 records for customer_demographic, and 983 for cleaned_customer.

```
#drop missing values
customer_demographic.dropna(subset=['DOB', 'tenure'], inplace=True)
customer_demographic.reset_index(drop=True, inplace=True)
print(customer_demographic[['DOB', 'tenure']].isnull().sum())

cleaned_customer.dropna(subset=['DOB'], inplace=True)
cleaned_customer.reset_index(drop=True, inplace=True)
print("\n",cleaned_customer[['DOB', 'tenure']].isnull().sum())
```

```
#change datetime type
customer_demographic['DOB'] = pd.to_datetime(customer_demographic['DOB'])
cleaned_customer['DOB'] = pd.to_datetime(cleaned_customer['DOB'])
```

The last ones are job_title and job_industry_category. There are quite a lot of missing values. The fact that there might be the possibility of trends or patterns existing between the missing group (like they don't prefer sharing their job to others), I decided to impute them with "Unknown".

```
#handle job and industry
customer_demographic['job_title'] = customer_demographic['job_title'].fillna("Unknown")
cleaned_customer['job_title'] = cleaned_customer['job_title'].fillna("Unknown")

customer_demographic['job_industry_category'] = customer_demographic['job_industry_category'].fillna("Unknown")
cleaned_customer['job_industry_category'] = cleaned_customer['job_industry_category'].fillna("Unknown")
```

Next, I start checking if there are any incorrect records. The first one is gender. We can see that there are basically 3 types of gender, Male, Female and U. I assumed 'U' represents Undefined.

```
#fix the value of gender
print("Before fixing:")
print(customer_demographic['gender'].unique())
print(cleaned_customer['gender'].unique())
```

```
Before fixing:
['F' 'Male' 'Female' 'U' 'Femal' 'M']
['Male' 'Female']
```

```
female_mask = customer_demographic['gender'].str.contains("F")
undefined_mask = customer_demographic['gender'].str.contains("U")
male_mask = customer_demographic['gender'].str.contains("M")
customer_demographic.loc[female_mask, 'gender'] = "Female"
customer_demographic.loc[undefined_mask, 'gender'] = "Undefined"
customer_demographic.loc[male_mask, 'gender'] = "Male"

print("After fixing:")
print(customer_demographic['gender'].unique())
print(cleaned_customer['gender'].unique())
```

```
After fixing:
['Female' 'Male' 'Undefined']
['Male' 'Female']
```

There are 196 variations for job_title, I therefore decided to create a new column called job_category that categorizes each job_title based on similarity. The new column now only includes 43 unique values. The function categorize_job() can be looked up on the code file.

```
customer_demographic['job_category'] = customer_demographic['job_title'].apply(categorize_job)
job_col = customer_demographic.pop(customer_demographic.columns[-1])
customer_demographic.insert(6, job_col.name, job_col)
display(customer_demographic.head())

cleaned_customer['job_category'] = cleaned_customer['job_title'].apply(categorize_job)
job_col = cleaned_customer.pop(cleaned_customer.columns[-1])
cleaned_customer.insert(5, job_col.name, job_col)
display(cleaned_customer.head())
```

I also created a new column called decade, which functions as the age group of the client.

```
customer_demographic['decade'] = customer_demographic['DOB'].apply(get_decade)
decade_col = customer_demographic.pop(customer_demographic.columns[-1])
customer_demographic.insert(5, decade_col.name, decade_col)
display(customer_demographic.head())

cleaned_customer['decade'] = cleaned_customer['DOB'].apply(get_decade)
decade_col = cleaned_customer.pop(cleaned_customer.columns[-1])
cleaned_customer.insert(4, decade_col.name, decade_col)
display(cleaned_customer.head())
```

The address information for customer_demographic is stored at customer_address. The latter contains important features such as property_valuation. I decided to merge the two and named in full_customer. Before merging, it is important to check whether there are duplicated records. As shown below, they are all unique.

```
#check duplicates
print(customer_demographic.duplicated().sum())
print(customer_address.duplicated().sum())
print(cleaned_customer.duplicated().sum())
```

```
0
0
0
```

```
# merging two tables
full_customer = customer_demographic.merge(customer_address, on='customer_id', how='left')
display(full_customer.head())
```

The final step is to identify if there are errors in the data. As shown here, there is a client who was born in the 1840s. I remove it because it doesn't make sense.

```
print(full_customer['decade'].unique())
print(cleaned_customer['decade'].unique())
```

```
['1950s' '1980s' '1960s' '1970s' '1990s' '2000s' '1840s' '1940s' '1930s']
['1950s' '1970s' '1960s' '1980s' '1990s' '1930s' '1940s' '2000s']
```

```
full_customer = full_customer[~ (full_customer['decade'] == "1840s")]
print(full_customer['decade'].unique())
```

```
['1950s' '1980s' '1960s' '1970s' '1990s' '2000s' '1940s' '1930s']
```

The state column also appears to have inconsistent values. Some are recorded with the full name of the state, while some aren't. I therefore changed them all into short version ones.

```
print(full_customer['state'].unique())
print(cleaned_customer['state'].unique())
```

```
['New South Wales' nan 'QLD' 'VIC' 'NSW' 'Victoria']
['QLD' 'NSW' 'VIC']
```

```
full_customer.loc[full_customer['state'] == 'New South Wales', 'state'] = 'NSW'
full_customer.loc[full_customer['state'] == 'Victoria', 'state'] = 'VIC'
cleaned_customer.loc[cleaned_customer['state'] == 'New South Wales', 'state'] = 'NSW'
cleaned_customer.loc[cleaned_customer['state'] == 'Victoria', 'state'] = 'VIC'
full_customer = full_customer.dropna(subset=['state'])
cleaned_customer = cleaned_customer.dropna(subset=['state'])
```

```
print(full_customer['state'].unique())
print(cleaned_customer['state'].unique())
```

```
['NSW' 'QLD' 'VIC']
['QLD' 'NSW' 'VIC']
```

I noticed there are null values in the address, I dropped them as there are only four of them.

```
display(full_customer[full_customer['state'].isna()])
```

	customer_id	full_name	gender	past_3_years_bike_
2	3	Arlin Dearle	Male	
9	10	Fiorenze Birdall	Female	
21	22	Deeanne Durtnell	Female	
22	23	Olav Polak	Male	

Transaction Table

In the transactions table, online_order, brand, product_line, product_class, product_size and standard_cost have missing values. I noticed that each product_id and list_price combination will only have one unique standard_cost, which means the brand, product_line, product_class and product_size should also be unique accordingly. However, the missing values only have one record presented, it is hard for me to impute them according to existing data.

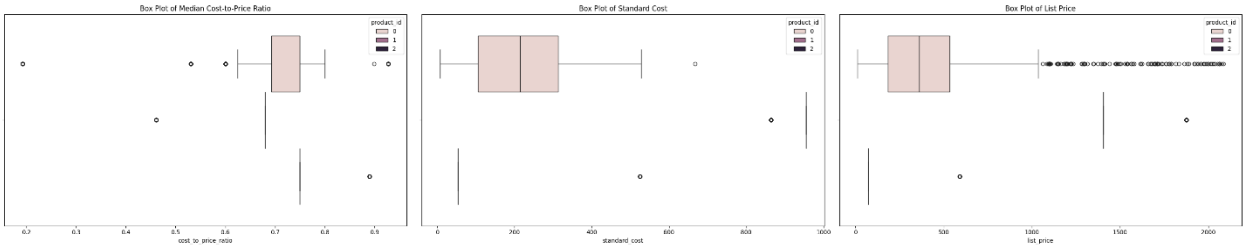
index	product_id	list_price	standard_cost	count
0	0	12.01	7.21	50
1	0	16.08	NaN	1
2	0	26.15	NaN	1
3	0	32.44	NaN	1
4	0	36.78	NaN	1
5	0	56.21	NaN	1

To maintain the consistency between records, options like either dropping the missing columns (maintaining the completeness of list_price) or dropping the missing values (capture as many features as possible but loose some information) are not quite ideal. I decided to impute the categorical features as "Missing Details", since there's the possibility that they exist some sort of relations with other records that cannot be grouped (e.g., special orders, pre-production models, or very old/rare items not fully cataloged).

```
transactions[['brand', 'product_line', 'product_class', 'product_size']] = transactions[['brand', 'product_line', 'product_class', 'product_size']].fillna("Missing Details")
display(transactions[transactions['brand'] == "Missing Details"])
```

	transaction_id	product_id	customer_id	transaction_date	online_order	order_status	brand	product_line	product_class	product_size	list_price	standard_cost
136	137	0	431	2017-09-23	0.0	Approved	Missing Details	Missing Details	Missing Details	Missing Details	1942.61	NaN
159	160	0	3300	2017-08-27	0.0	Approved	Missing Details	Missing Details	Missing Details	Missing Details	1656.86	NaN
366	367	0	1614	2017-03-10	0.0	Approved	Missing Details	Missing Details	Missing Details	Missing Details	850.89	NaN

For standard_cost, while common practices will be imputing with median or mean, I noticed it will occur a problem as not all list_price are larger than the imputed value, resulting negative profit for the transactions. To handle this issue, I decided to impute it based on its relationship with list_price.



As the graph above, each product_id has its own range of cost_to_price_ratio, I therefore impute the standard_cost based on the cost_to_price_ratio of each product_id.

```
] temp_df_for_ratio = transactions.dropna(subset=['product_id', 'list_price', 'standard_cost']).copy()

  if (temp_df_for_ratio['list_price'] == 0).any():
      print("Warning: Some list_price values are zero in non-null records. This will affect ratio calculation.")
```

```
] temp_df_for_ratio['cost_to_price_ratio'] = temp_df_for_ratio['standard_cost'] / temp_df_for_ratio['list_price']

  median_cost_to_price_ratio = temp_df_for_ratio.groupby('product_id')['cost_to_price_ratio'].median()
```

```
missing_cost_mask = transactions['standard_cost'].isnull()
valid_price_mask = transactions['list_price'].notnull()

product_ids_for_imputation = transactions.loc[missing_cost_mask & valid_price_mask, 'product_id']

# Map the product_ids to their corresponding median_cost_to_price_ratio
ratios_for_imputation = product_ids_for_imputation.map(median_cost_to_price_ratio)

transactions.loc[missing_cost_mask & valid_price_mask, 'standard_cost'] = \
    transactions.loc[missing_cost_mask & valid_price_mask, 'list_price'] * ratios_for_imputation
```

Next, the online_order column. I decided to impute it with the mode, the most frequent value appearing. The reason is that it is a binary column with a very low percentage of missing values. Assuming the missing values are following the majority is the most common practice for this situation.

```
transactions['online_order'] = transactions['online_order'].fillna(transactions['online_order'].mode()[0])
```

Lastly, the product_first_sold_date. It may present the buying behavior of the product of each customer, so it is better to retain the completeness. The strategy I took for this column is to impute it with the earliest transaction_date for each customer and unique product pair.

```
earliest_sold = transactions.groupby(['customer_id', 'product_id', 'list_price'])['transaction_date'].transform('min')
```

```
transactions['product_first_sold_date'] = transactions['product_first_sold_date'].fillna(earliest_sold)
```

I also converted the transaction_date into datetime for better handling.

```
# check the date is correct
transactions['transaction_date'] = pd.to_datetime(transactions['transaction_date'], errors='coerce')
```

The transactions data should be able to match the customer data, the next step I do is checking whether there are customer_id that are not presented in full_customer. As below presented, there are 450 customer_id that cannot be found in full_customer. I then remove them.

```
#remove customer id that are not in customer_demographic
no_customer_mask = ~transactions['customer_id'].isin(full_customer['customer_id'])
```

```
# remove the rows with no demographic data
transactions = transactions[~no_customer_mask].reset_index(drop=True)
```

I also changed the type of online_order to Boolean since float doesn't make sense:

```
# change online order to boolean
transactions['online_order'] = transactions['online_order'].astype(bool)
```

There are no duplicated records.

```
# check duplicate
print(transactions.duplicated().sum())
```

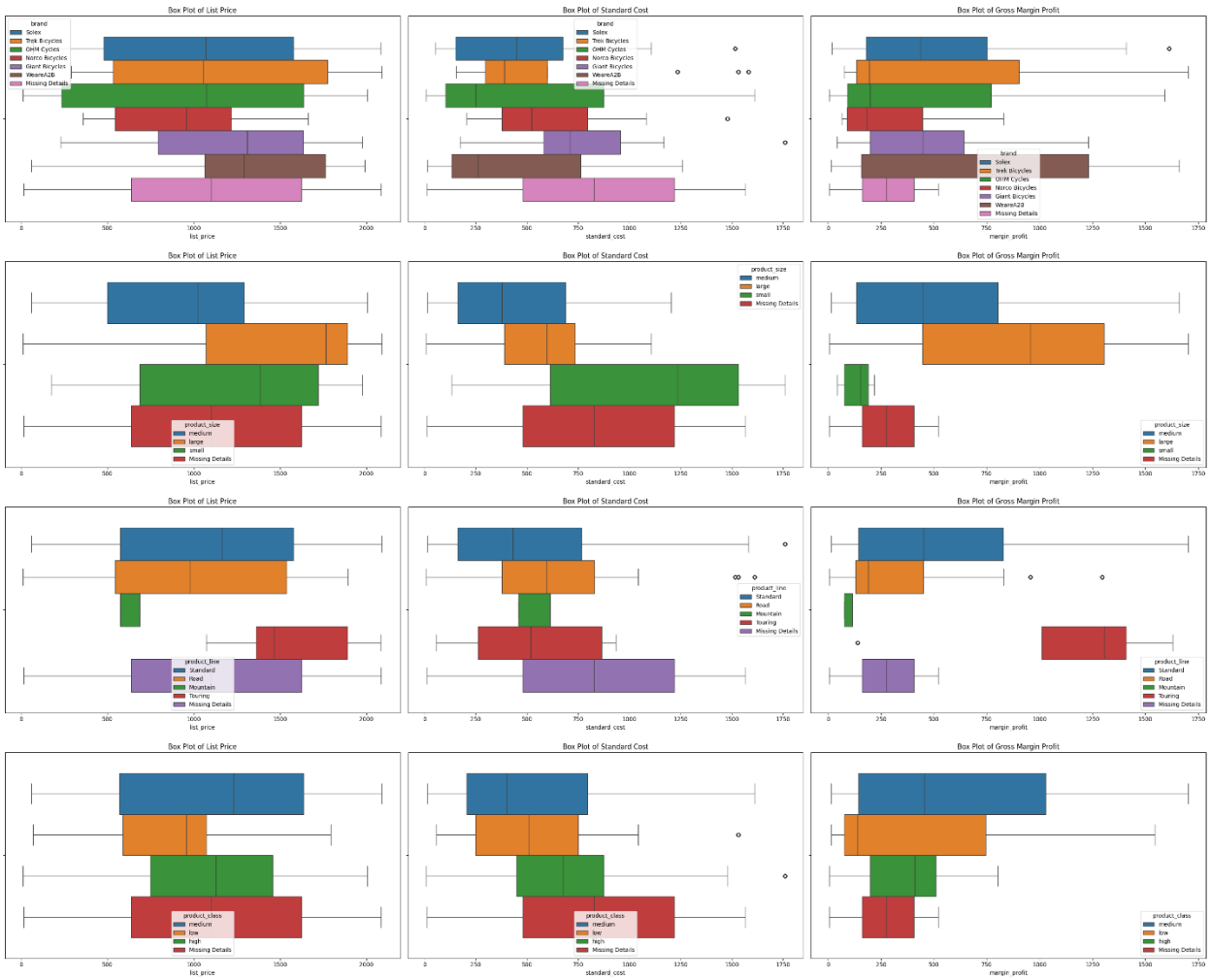
```
0
```

Then, I created two columns: product and margin_profit.

```
transactions['margin_profit'] = transactions['list_price'] - transactions['standard_cost']
```

```
transactions.loc[transactions['brand'] != "Missing Details", 'product'] = (
     transactions['brand'].astype(str) + '_' +
     transactions['product_id'].astype(str) + '_' +
     transactions['product_line'].astype(str) + '_' +
     transactions['product_class'].astype(str) + '_' +
     transactions['product_size'].astype(str)
 )
transactions.loc[transactions['brand'] == "Missing Details", 'product'] = (
     'Unknown_' +
     transactions['product_id'].astype(str)
 )
```

The last thing is to handle the outliers. list_price, standard_cost and margin_profit vary hugely, but there might be the chance that it is due to the nature of the product, for example line or size. I therefore chose to remove outliers based on the cost and profit of each group. The function remove_outliers_iqr() can be found in code.



```python
numerical_cols_to_check = ['standard_cost', 'list_price', 'margin_profit']
categorical_cols_to_check = ['brand', 'product_line', 'product_class', 'product_size']

final_keep_mask = pd.Series(True, index=transactions.index)

for cat_col in categorical_cols_to_check:
    print(f"\n--- Processing outliers grouped by: {cat_col} ---")

    current_cat_grouping_masks = {}

    for num_col in numerical_cols_to_check:
        print(f"  - Checking {num_col} outliers within {cat_col} groups.")

        mask = transactions.groupby(cat_col)[num_col].transform(
            lambda x: remove_outliers_iqr(x, threshold=1.5)
        )
        current_cat_grouping_masks[num_col] = mask

    # Combine masks for this specific categorical grouping (e.g., if any of the three num_cols is outlier in 'brand' group)
    combined_mask_for_this_cat_grouping = pd.Series(True, index=transactions.index)
    for mask_series in current_cat_grouping_masks.values():
        combined_mask_for_this_cat_grouping = combined_mask_for_this_cat_grouping & mask_series

    # Accumulate into the final mask
    # If a row is an outlier in Brand-based check OR ProductLine-based check, it will be False.
    final_keep_mask = final_keep_mask & combined_mask_for_this_cat_grouping
```

Now, here are the final datasets.

Customer Demographic

```
<class 'pandas.core.frame.DataFrame'>
Index: 3908 entries, 0 to 3908
Data columns (total 17 columns):
 #   Column                               Non-Null Count  Dtype
---  ------                               --------------  -----
 0   customer_id                          3908 non-null   int64
 1   full_name                            3908 non-null   object
 2   gender                               3908 non-null   object
 3   past_3_years_bike_related_purchases  3908 non-null   int64
 4   DOB                                  3908 non-null   datetime64[ns]
 5   decade                               3908 non-null   object
 6   job_title                            3908 non-null   object
 7   job_category                         3908 non-null   object
 8   job_industry_category                3908 non-null   object
 9   wealth_segment                       3908 non-null   object
 10  deceased_indicator                   3908 non-null   object
 11  owns_car                             3908 non-null   object
 12  tenure                               3908 non-null   float64
 13  address                              3908 non-null   object
 14  postcode                             3908 non-null   object
 15  state                                3908 non-null   object
 16  property_valuation                   3908 non-null   float64
dtypes: datetime64[ns](1), float64(2), int64(2), object(12)
memory usage: 549.6+ KB
None
```

Cleaned Customer

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 983 entries, 0 to 982
Data columns (total 18 columns):
 #   Column                               Non-Null Count  Dtype
---  ------                               --------------  -----
 0   full_name                            983 non-null    object
 1   gender                               983 non-null    object
 2   past_3_years_bike_related_purchases  983 non-null    int64
 3   DOB                                  983 non-null    datetime64[ns]
 4   decade                               983 non-null    object
 5   job_title                            983 non-null    object
 6   job_category                         983 non-null    object
 7   job_industry_category                983 non-null    object
 8   wealth_segment                       983 non-null    object
 9   deceased_indicator                   983 non-null    object
 10  owns_car                             983 non-null    object
 11  tenure                               983 non-null    int64
 12  address                              983 non-null    object
 13  postcode                             983 non-null    object
 14  state                                983 non-null    object
 15  property_valuation                   983 non-null    int64
 16  Rank                                 983 non-null    int64
 17  Value                                983 non-null    float64
dtypes: datetime64[ns](1), float64(1), int64(4), object(12)
memory usage: 138.4+ KB
None
```

Transactions

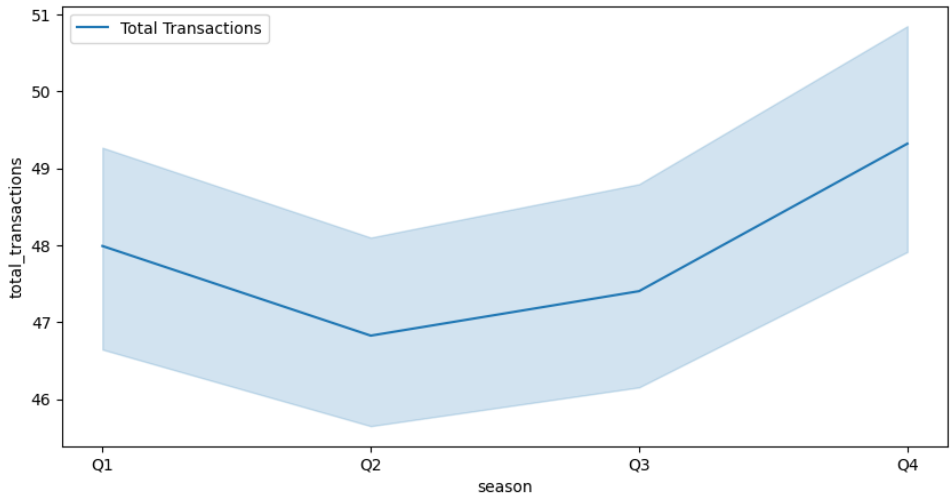
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17462 entries, 0 to 17461
Data columns (total 15 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   transaction_id           17462 non-null  int64
 1   product_id               17462 non-null  int64
 2   customer_id              17462 non-null  int64
 3   transaction_date         17462 non-null  datetime64[ns]
 4   online_order             17462 non-null  bool
 5   order_status             17462 non-null  object
 6   brand                    17462 non-null  object
 7   product_line             17462 non-null  object
 8   product_class            17462 non-null  object
 9   product_size             17462 non-null  object
 10  list_price               17462 non-null  float64
 11  standard_cost            17462 non-null  float64
 12  product_first_sold_date  17462 non-null  datetime64[ns]
 13  margin_profit            17462 non-null  float64
 14  product                  17462 non-null  object
dtypes: bool(1), datetime64[ns](2), float64(3), int64(3), object(6)
memory usage: 1.9+ MB
None
```

Time Series Based Exploratory Data Analysis

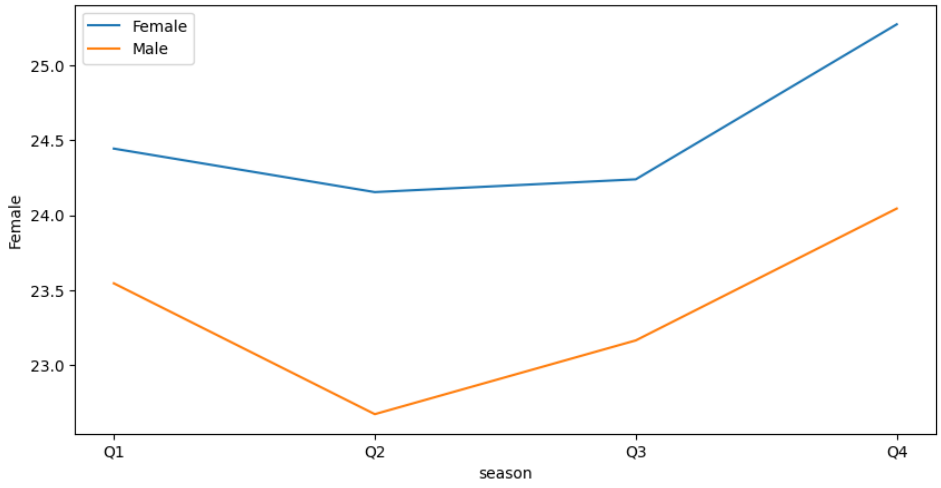
The Kaggle dataset's introduction mentions that the company has been experiencing a significant sales decline within the young professional segment. My initial step is to validate this observed trend using the provided data. However, the data only includes the year 2017, it may not fully reveal the trends or pattern of the past historical transaction behaviors.

Transaction Numbers

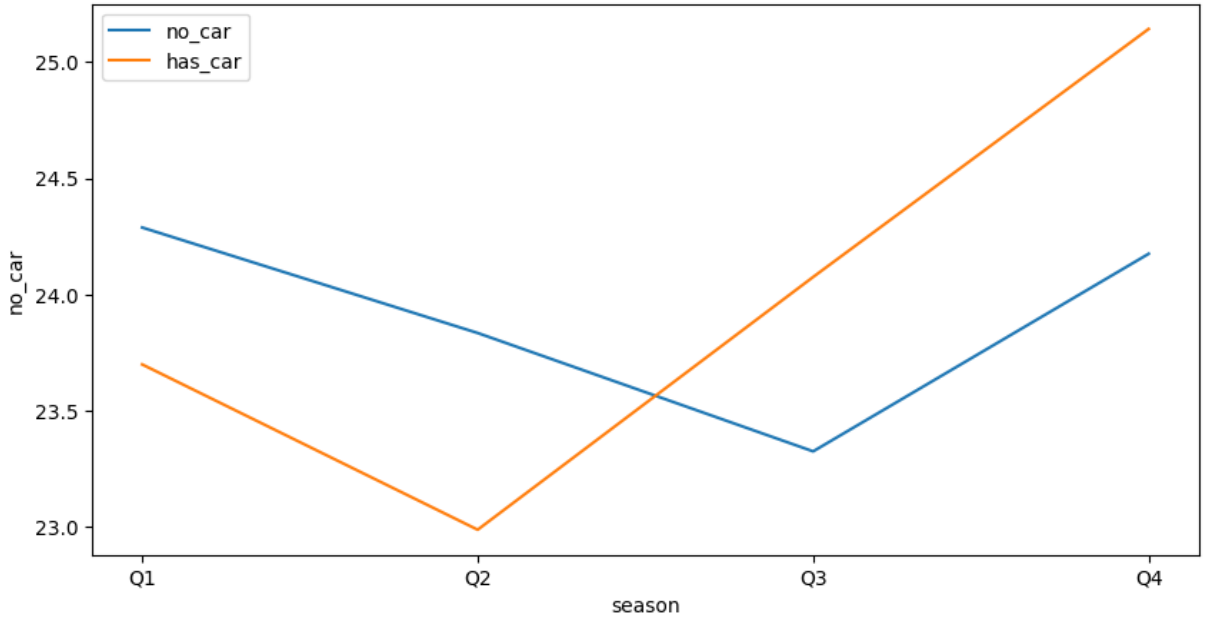
The total number of transactions shows a distinct seasonal pattern. Transactions dipped from Q1 to Q2, reaching the lowest point of the year. They then begin to recover in Q3 and peak in Q4, indicating the fourth quarter is the strongest sales period.



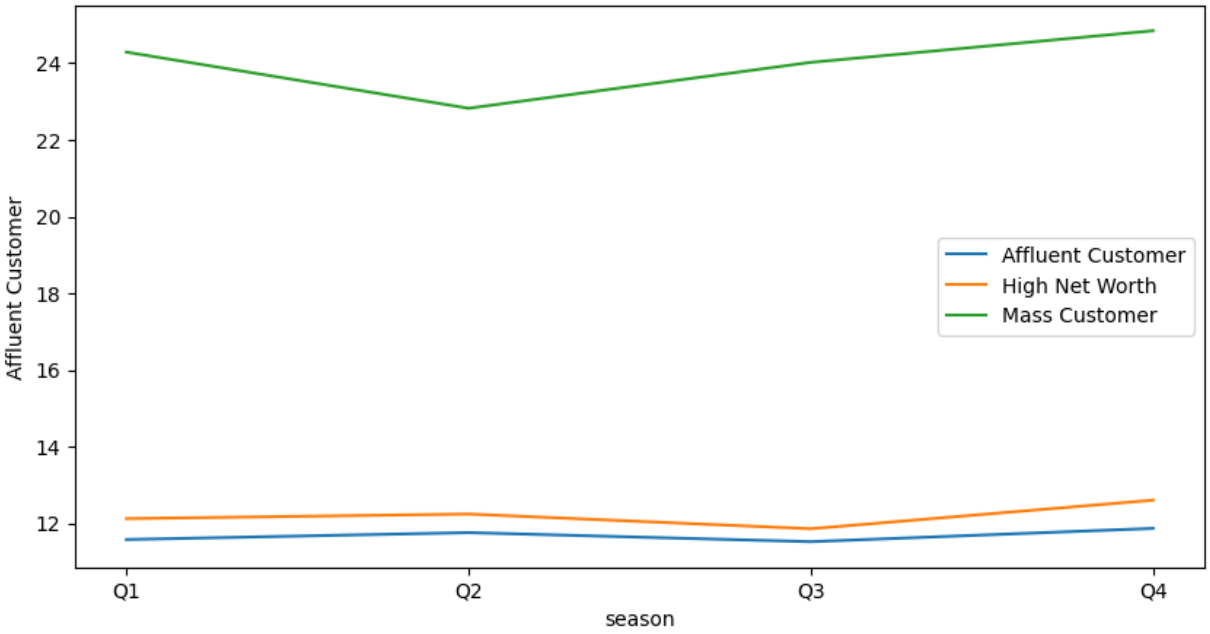
A breakdown by customer demographics reveals varying transaction patterns as well. Starting with gender, both male and female customers follow a similar trend of dipping in Q2 and peaking in Q4. However, female customers consistently make more transactions than male customers throughout the year.



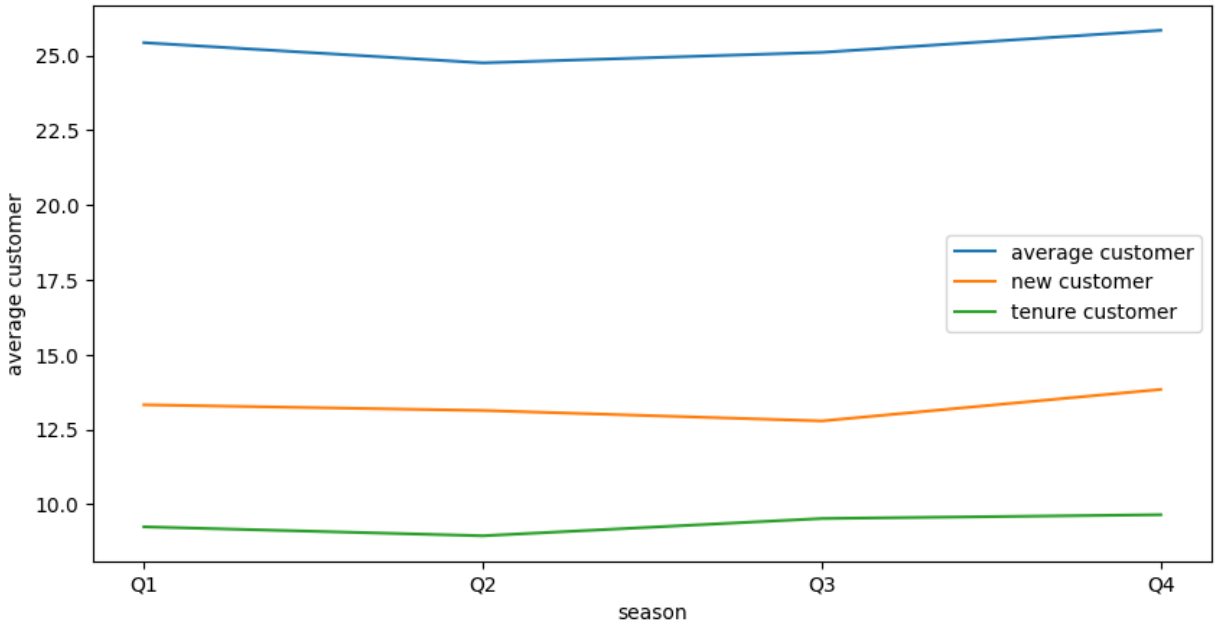
The transaction behavior of customers with and without cars diverges significantly either. Customers without a car have a dip in Q3 and a rise to their peak in Q4. Conversely, customers who own a car have the lowest transactions in Q2 but then rise sharply to a peak in Q4, overtaking those without cars. This suggests a different motivation for purchasing, perhaps for recreational use, which is higher towards the end of the year.



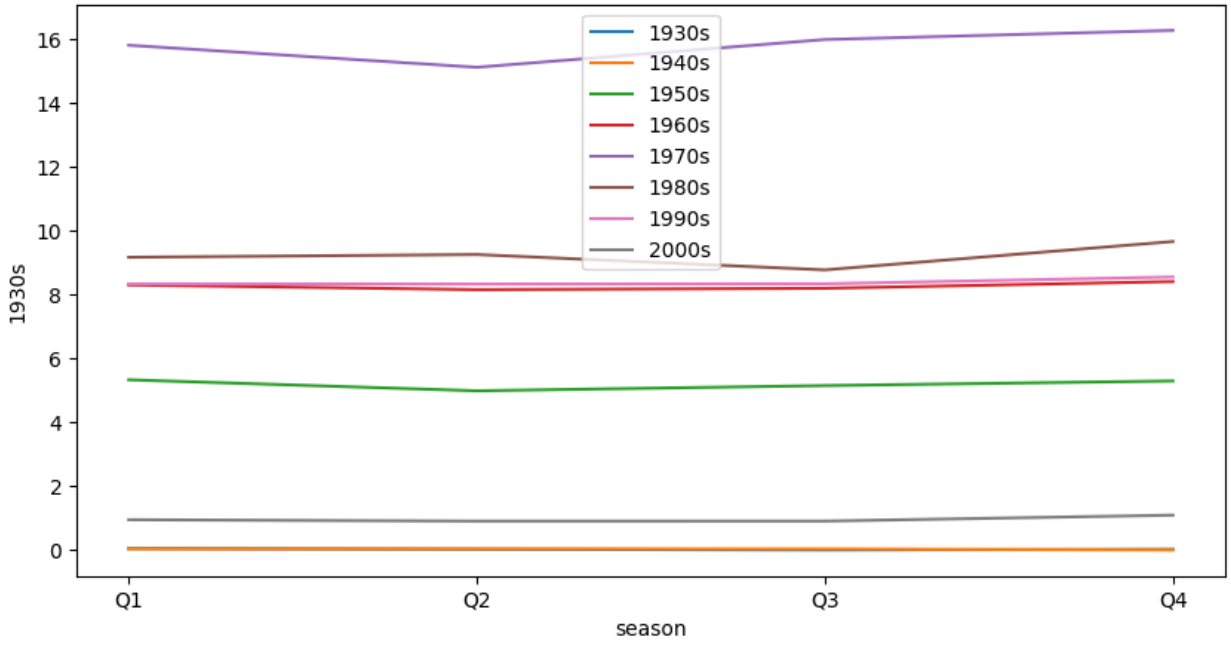
Differentiation by customer value and lifecycle also presents distinct purchasing behaviors. When segmented by wealth, "Mass Customers" account for the highest number of transactions, followed by "High Net Worth" and "Affluent Customers," who have the lowest transaction volume.



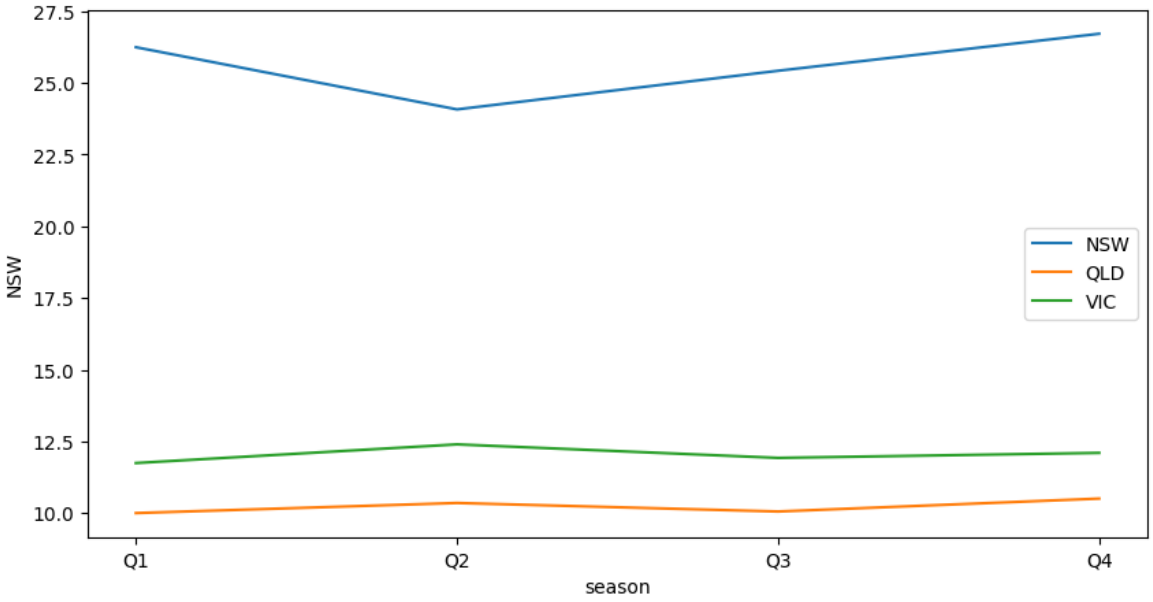
When segment by customer Lifecycle: The "average customer" transaction numbers are consistently the highest. "New customers" have a higher transaction rate than "tenure customers," with both categories showing a slight dip in Q2 or Q3.



When analyzing transaction data based on the customer's birth decade, those born in the 1970s consistently make the most purchases. Though those born in 2000s are not the lowest of all, it is the lowest after 30s and 40s; 1990s have the average transaction numbers. All decades show a relatively stable transaction pattern across the quarters, with minor fluctuations.

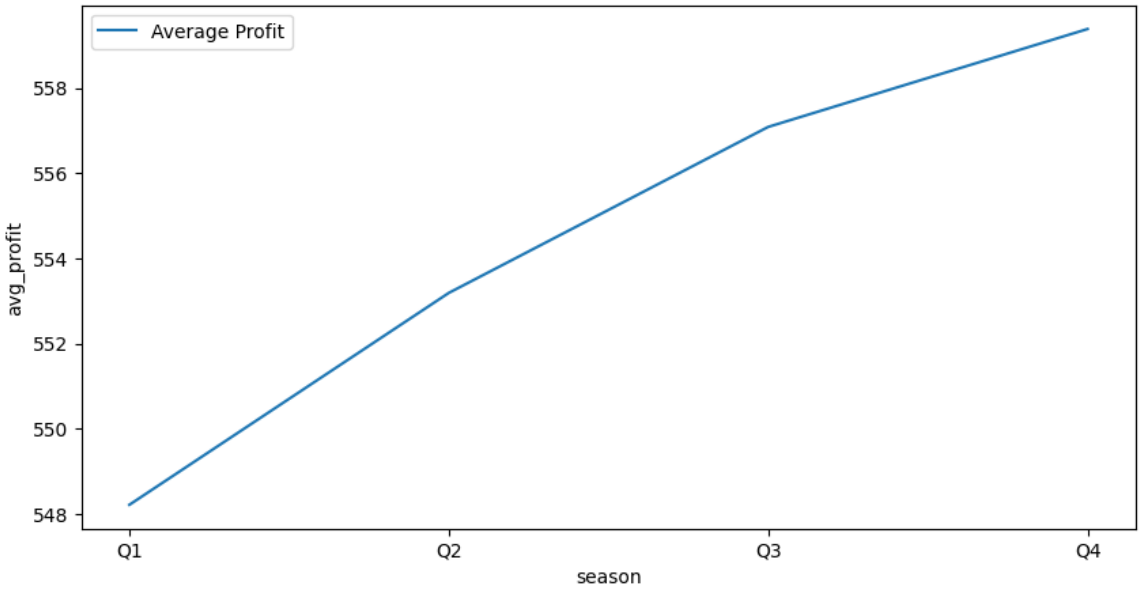


Finally, transaction data from different Australian states reveals regional variations. New South Wales (NSW) leads in transaction volume, following the general trend of a Q2 dip and a Q4 peak. Victoria (VIC) has the next highest number of transactions and maintains a relatively stable purchasing pattern throughout the year. Queensland (QLD) has the lowest number of transactions, with a slight dip in Q3.

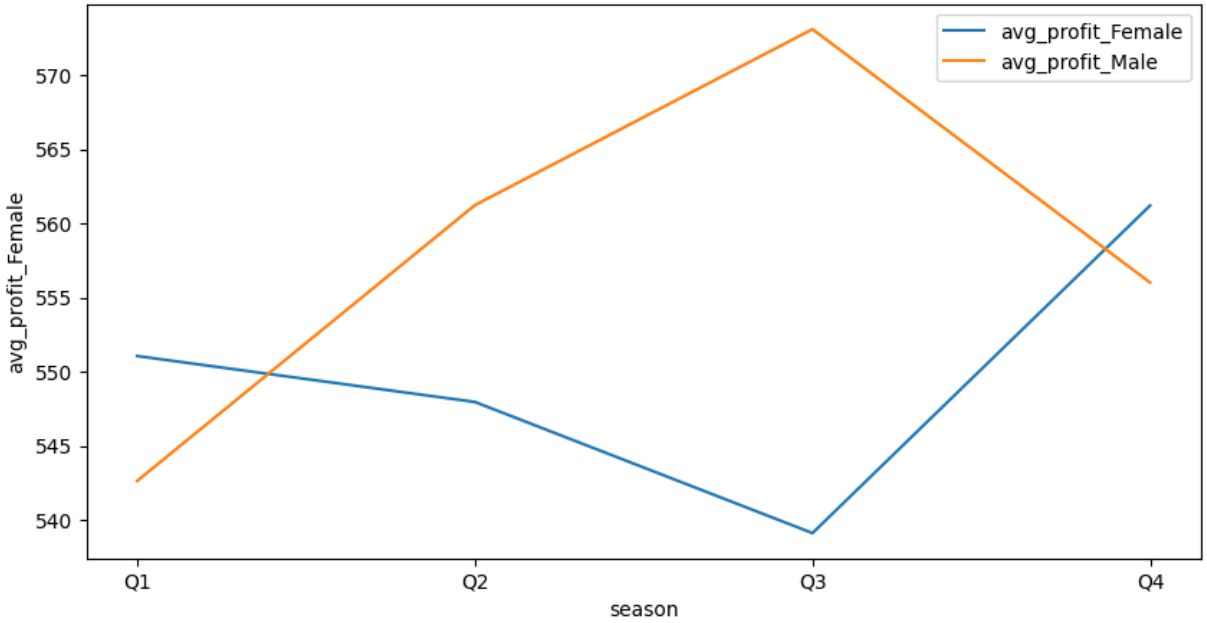


Gross Margin Profit

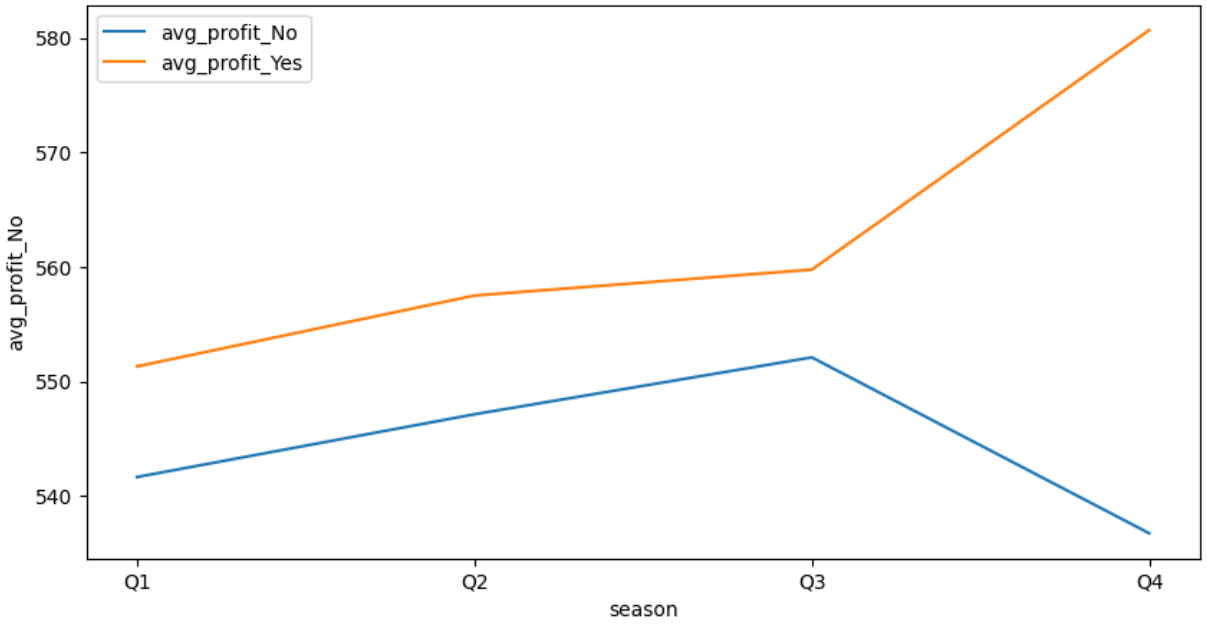
The average profit per transaction demonstrates a consistent upward trend throughout the year. It starts at its lowest point in Q1 and steadily increases through Q2 and Q3, reaching its peak in Q4. This suggests that either customers are buying higher-margin products or the company's profitability per sales improves as the year progresses.



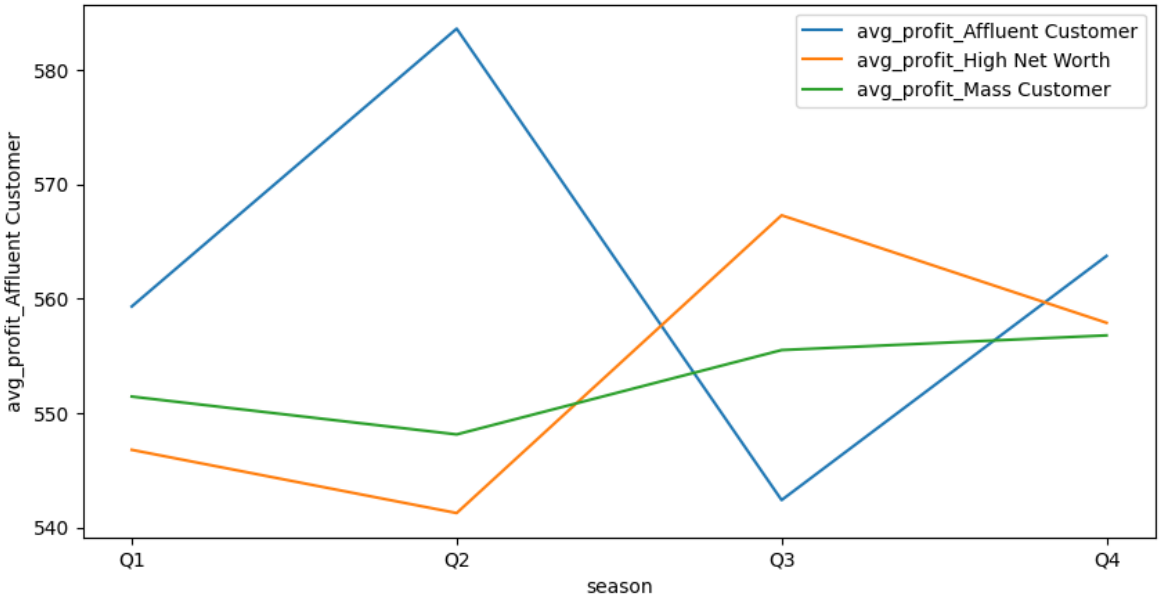
Analyzing profitability across different customer demographics reveals distinct patterns in this case too. The profitability for male and female customers shows a crossing pattern. The average profit from male customers starts higher in Q1 but declines until Q3, before rising in Q4. Conversely, the profit from female customers starts lower, peaks significantly in Q3, and then declines in Q4, though the overall profit remains higher than male customers.



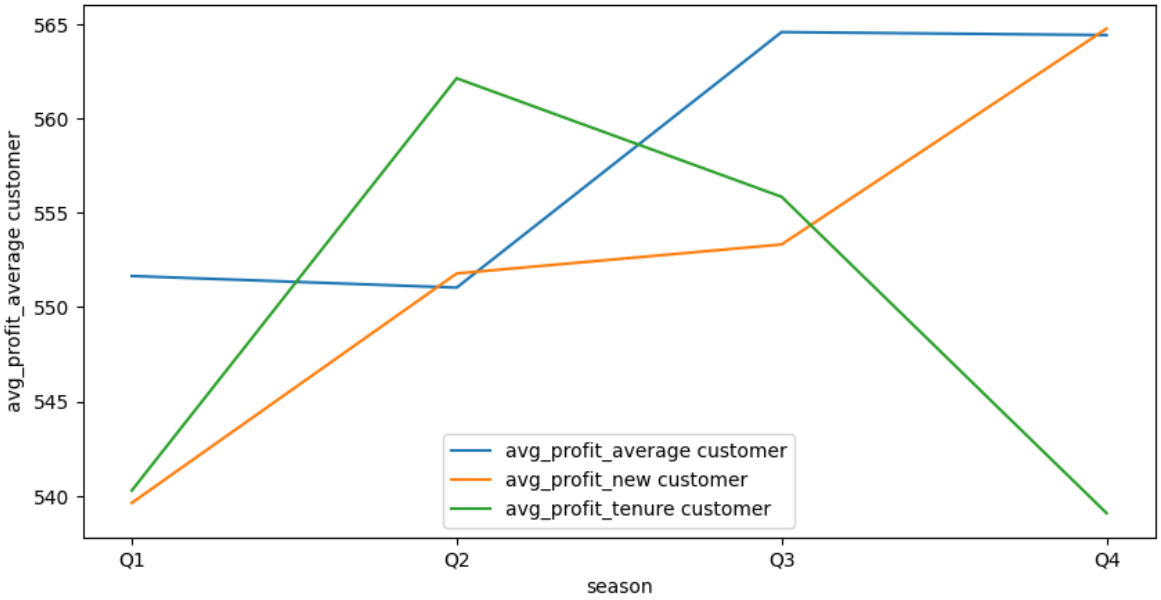
Customers who own a car consistently generate higher average profits than those who do not. The profit from car owners shows a steady and strong increase throughout the year, peaking in Q4. The profit from customers without a car also increases until Q3 but then sees a sharp drop in Q4.



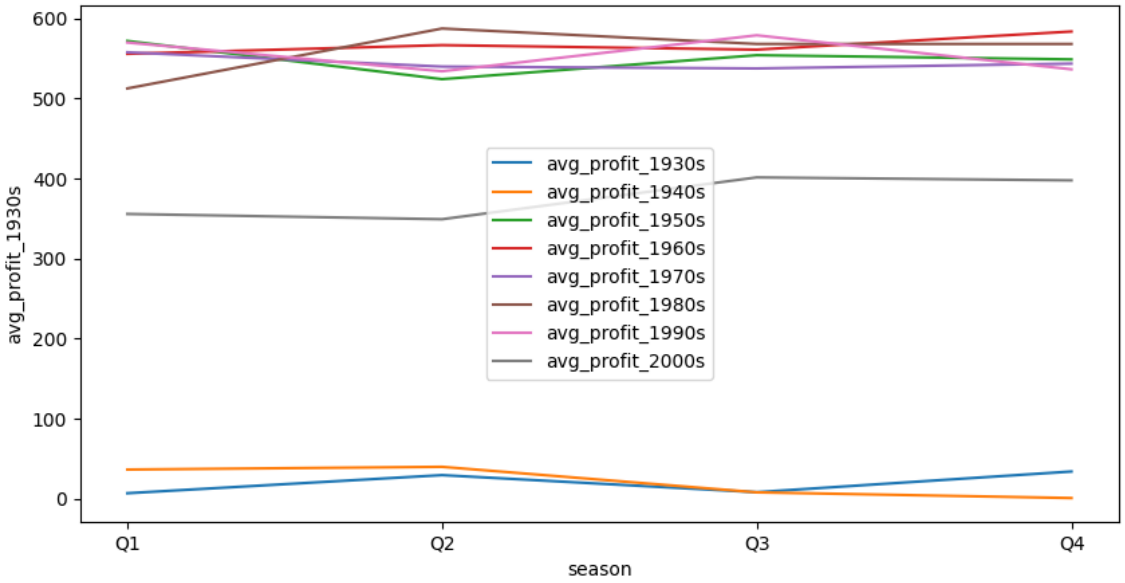
The analysis of customer segments based on wealth and tenure also shows notable variations. The profitability of different wealth segments is very different. Affluent Customers show the most dramatic changes, with a huge spike in profitability in Q2, followed by a sharp drop in Q3, and a recovery in Q4. High Net Worth customers' profitability is lowest in Q2, peaks in Q3 and is the highest in Q3. Mass Customers show a more stable and gradual increase in profitability throughout the year.



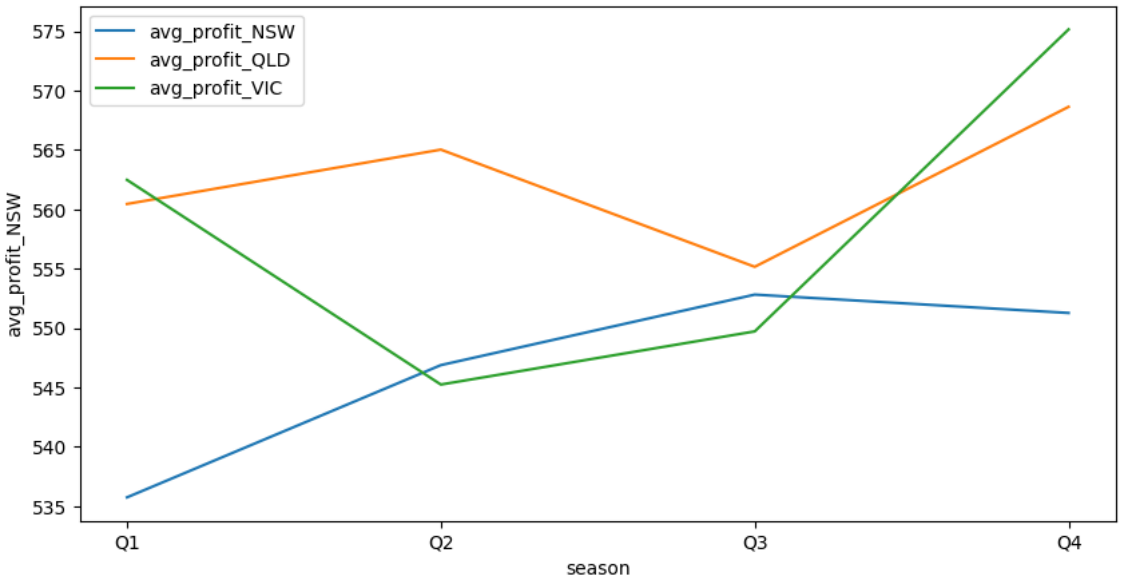
The profitability of different customer tenure groups also shows interesting trends. Tenure customers are most profitable in Q2, followed by a sharp decline. New customers show a steady increase in profitability, peaking in Q4 and the highest in Q4. The average customer profit remains relatively stable until an increase in Q3 and Q4.



Profitability by the decade of birth shows that customers born in the 1950s to 1990s are generally the most profitable, with no significant differences. Those born in 1980 have a pretty obvious growth in Q2 then remain steadily high; 1950s and 1990s have a drop in Q2, rise back in Q3, but drop again in Q4. 1930s and 1940s have the lowest overall profitability; the 2000s are in the middle, but it is clear that the profitability is steadily growing.



Finally, the average profit generated from different Australian states shows dynamic changes over the year. Initially, in Q1, Victoria (VIC) has the highest average profit. In Q2, Queensland (QLD) becomes the most profitable state. Towards the end of the year, in Q4, both VIC and QLD show a strong increase in profitability, with VIC becoming the most profitable state. New South Wales (NSW) shows a more gradual and steady increase in profit throughout the year but is relatively the lowest in average.



Analysis Summary

This initial analysis evaluates the corporate assumption that the company is "experiencing a significant sales decline within the young professional segment" due to a "lack of personalized customer experience and tailored after-sales services." To assess the validity of the company's assumption, we define the "young professional segment" as customers born in the 1990s and 2000s.

Finding 1: Transaction Volume is Stable, Not Declining.

An analysis of transaction volumes does not show a "significant sales decline" for this segment.

1. 1990s: This group contributes "average transaction numbers." Their performance is stable and consistent, placing them in the middle of all customer segments. This is not indicative of a decline.
2. 2000s: This group currently has low transaction volume. However, a low starting point for the youngest active customer segment is expected and should be interpreted as a baseline for growth, not a state of decline.

Finding 2: Profitability is Growing, Contradicting the Narrative of Decline.

The most critical finding from the data directly contradicts the idea of a segment in decline.

1. 2000s: The average gross margin profit for this group is "steadily growing" throughout the year. This is a key indicator of positive engagement and future potential. A segment whose value-per-customer is increasing is a growth opportunity. A decline would be evidenced by falling profitability, which is the opposite of what the data shows.
2. 1990s: This group maintains a position among the "most profitable" customer segments, demonstrating their continued value to the company.

The combination of stable transaction volume and growing profitability invalidates the assumption of a "significant sales decline." The data does not support a narrative of decline. Instead, it reveals a significant opportunity within this demographic, characterized by stable transaction volumes in its older customer groups and, most importantly, steadily growing profitability in its youngest customer segment. The core issue is not the loss of existing customers but a failure to fully capture and cultivate this emerging market. The company's research identifying a need for personalization and service is correct, but these factors are barriers to growth, not drivers of a decline. Therefore, our analysis indicates that the assumption should be rejected.

Our initial recommendation is to "Nurture the Emerging Professional Segment" by focusing on hyper-personalization, value-added services, and community building to accelerate the already positive profitability trends and convert latent potential into significant revenue. Eventually becoming the Brand of Choice for the Next Generation of Cyclists. Detailed Strategies will be revealed after defining the exact customer segments.

Data Preprocessing

Creating Input Dataset for Clustering

Since the main purpose is to conduct customer segmentation, I need to create an input dataset that is grouped by customer. I started with creating a function that groups the transactional data by customers. The grouping function, transaction_features(), can be found in code. After grouping it, I merged with existing demographic data, full_customer, as cleaned_customer doesn't have customer_id for merging.

```
customer_portfolio = transaction_features(transactions)
```

Here is the explanation of the columns.

customer_id	int	The ID number of the customer who made the transaction.
last_transaction_days	int	The number of days between "current date" (the most recent transaction date + 1 day) and the customer's recent transaction date.
order_frequency	int	The total number of transactions (or orders) a customer has made within the analyzed period, in this case 2017.
monetary_total	float	The sum of `margin_profit` generated by the customer across all their transactions.
monetary_max	float	The maximum `margin_profit` recorded for this customer.
monetary_min	float	The minimum `margin_profit` recorded for this customer.
avg_order_value	float	The average `margin_profit` for this customer. Calculated as `monetary_total` / `order_frequency`.
order_value_range	float	The standard deviation (std) of `margin_profit` across all transactions of the customer. For customers with only one transaction, this value is 0
num_unique_products	int	The total number of distinct products that a customer has purchased.
avg_transaction_frequency	float	The average number of days between consecutive transactions for a customer. Calculated from the time differences between their purchases.
min_time_gap	float	The shortest duration (in days) between any two consecutive transactions for a customer.
max_time_gap	float	The longest duration (in days) between any two consecutive transactions for a customer.

num_brands	int	The total number of distinct brands that a customer has purchased from.
preferred_product	string	The product that the customer has purchased most frequently. If there's a tie, the first one encountered by mode() is returned.
preferred_brand	String	The brand that the customer has purchased from most frequently.
prefered_line	String	The product_line that the customer has purchased from most frequently.
preferred_class	String	The product_class that the customer has purchased from most frequently.
preferred_size	string	The product_size that the customer has purchased most frequently.
online_order_ratio	Float	The proportion of a customer's total orders that were placed online.
approved_ratio	float	The proportion of a customer's total orders that had an 'Approved' order_status.
brand_switching_rate	float	How often the customer switches brands. Calculated by dividing `tenure` with `num_brands`
chance_to_churn	string	The chance of the customer churning. Calculated using `last_transaction_days`, `avg_transaction_frequency`, `tenure`, and related thresholds. 1. New Buyer 2. Low 3. Middle 4. High
gender	string	The gender of the customer, Male or Female.
past_3_years_bike_related_purchases	int	The total quantity of bike related purchase events of the customer for the past three years.
decade	string	The decade when the customer is born. 1. 1930s 2. 1940s 3. 1950s 4. 1960s 5. 1970s 6. 1980s 7. 1990s

		8. 2000s
job_category	string	The grouped category for `job_title`, created based on `job_title` during preprocessing stage. 43 variations including: Administrative, System Analyst, Engineer, Developer, Accountant, Junior Executive, Middle Manager, Field Specialist, Researcher, Editor, Nurse, Legal, Social Worker, Media, Coordinator, Chemist, Executive Manager, Data Analyst, Sales, Operator, Statistician, Auditor, Therapist, Pathologist, Quality Engineer, Community Services, Teacher, Professor, Designer, Programmer, Human Resources, Administrator, Recruiter, Librarian, Pharmacist, Low Manager, Geologist, IT Support, Actuary, Writer, Marketing, Scientist, Software Engineer
Job_industry_category	string	The industry where the customer is working in. 9 variations including: Health, Financial Services, Argiculture, Manufacturing, Telecommunications, Entertainment, Retail, Property, IT
wealth_segment	string	The financial standing and wealth level of the customer 7. Mass Customer 8. Affluent Customer High Net Worth
deceased_indicator	string	Whether the customer is deceased. N for No means still alive; Y for Yes means no longer alive.
owns_car	string	Whether or not the customer owns a car, Yer or No.
tenure	float	The number of years the customer has spent with the company, ranging from 1 to 22.
state	string	The state of the country where the customer lives in.
property_valuation	int	The assessed value of the property where the customer lives in, ranging from 1 to 12.
tenure_group	string	How long the client has been with the company. Calculated using `tenure`. 1. new customer 2. average customer 3. tenure customer

The transactional columns include last_transaction_days, order_frequency, monetary_total, monetary_max, monetary_min, avg_order_value, order_value_range, num_unique_products, avg_transaction_frequency, min_time_gap, max_time_gap, num_brands, preferred_product, preferred_brand, prefered_line, preferred_class, preferred_size, online_order_ratio, approved_ratio, brand_switching_rate, chance_to_churn

The demographic columns include gender, past_3_years_bike_related_purchases, decade, job_category, job_industry_category, wealth_segment, deceased_indicator, owns_car, tenure, state, property_valuation, tenure_group

Unsupervised Clustering

Data preprocessing

Encoding

For ordinal features OrdinalEncoder is used; for nominal features OneHotEncoder. However, these columns need handling: preferred_class, preferred_size, chance_to_churn, and preferred_product.

```
print(customer_portfolio['preferred_class'].unique())
print(customer_portfolio['preferred_size'].unique())
print(customer_portfolio['chance_to_churn'].unique())
```

```
['medium' 'high' 'Missing Details' 'low']
['medium' 'large' 'Missing Details' 'small']
['Low' 'Middle' 'New Buyer' 'High']
```

As the photo above shows, these three columns have ordinal characteristics. I separated them to better capture the data.

```
# --- 2. Separate 'preferred_class' ---
customer_portfolio['is_preferred_class_missing'] = (
    customer_portfolio['preferred_class'] == 'Missing Details'
).astype(int)

# Column with only ordered categories (will have NaN where 'Missing Details' was)
ordered_classes_map = {'low': 'low', 'medium': 'medium', 'high': 'high'} # No actual change, just to map explicitly
customer_portfolio['preferred_class_ordered'] = customer_portfolio['preferred_class'].map(ordered_classes_map)


# --- 3. Separate 'preferred_size' ---
customer_portfolio['is_preferred_size_missing'] = (
    customer_portfolio['preferred_size'] == 'Missing Details'
).astype(int)

# Column with only ordered categories
ordered_sizes_map = {'small': 'small', 'medium': 'medium', 'large': 'large'}
customer_portfolio['preferred_size_ordered'] = customer_portfolio['preferred_size'].map(ordered_sizes_map)


# --- 4. Separate 'chance_to_churn' ---
customer_portfolio['is_new_buyer'] = (
    customer_portfolio['chance_to_churn'] == 'New Buyer'
).astype(int)

# Column with only ordered churn risk levels
ordered_churn_map = {'Low': 'Low', 'Middle': 'Middle', 'High': 'High'}
customer_portfolio['chance_to_churn_ordered'] = customer_portfolio['chance_to_churn'].map(ordered_churn_map)
```

```
nan_fill_value_for_ordinal = 'N/A_Ordinal'

customer_portfolio['preferred_class_ordered'] = customer_portfolio['preferred_class_ordered'].fillna(nan_fill_value_for_ordinal)
customer_portfolio['preferred_size_ordered'] = customer_portfolio['preferred_size_ordered'].fillna(nan_fill_value_for_ordinal)
customer_portfolio['chance_to_churn_ordered'] = customer_portfolio['chance_to_churn_ordered'].fillna(nan_fill_value_for_ordinal)
```

```
ordinal_encoder_categories = [
    # For 'chance_to_churn_ordered'
    ['Low', 'Middle', 'High', nan_fill_value_for_ordinal],
    # For 'preferred_class_ordered'
    ['low', 'medium', 'high', nan_fill_value_for_ordinal],
    # For 'preferred_size_ordered'
    ['small', 'medium', 'large', nan_fill_value_for_ordinal],
    # For 'decade' (from ['1930s', '1940s', ... '2000s'])
    ['2000s', '1990s', '1980s', '1970s', '1960s', '1950s', '1940s', '1930s'],
    # For 'tenure_group'
    ['new customer', 'average customer', 'tenure customer'],
    # For 'wealth_segment'
    ['Mass Customer', 'Affluent Customer', 'High Net Worth']
]
```

For preferred_product, there's a total of 160 variations. It is not ideal to simply conduct One Hot Encoding; I therefore categorize them into 3 groups based on buying frequency.

```
product_counts = customer_portfolio['preferred_product'].value_counts()
product_counts_for_thresholding = product_counts[~(product_counts.index.str.contains("Unknown"))]

# --- Define popularity thresholds based on quantiles of counts ---
popular_threshold = product_counts_for_thresholding.quantile(0.8) # Top 20% most preferred products
average_threshold = product_counts_for_thresholding.quantile(0.3) # Bottom 30% are rare, middle 50% are average

# Create a mapping from product name to its popularity category
def get_popularity_category(product_name):
    if "Unknown" in str(product_name):
        return 'Missing_Product_Preference' # Treat missing as a distinct category
    if product_name not in product_counts_for_thresholding.index:
         return 'Rare_Product' # Handle products not seen during training, or extremely rare ones

    count = product_counts_for_thresholding[product_name]
    if count >= popular_threshold:
        return 'Popular_Product'
    elif count > average_threshold: # Between average_threshold and popular_threshold
        return 'Average_Product'
    else: # Less than or equal to average_threshold
        return 'Rare_Product'
```

```
customer_portfolio['preferred_product_binned'] = customer_portfolio['preferred_product'].apply(get_popularity_category)
```

Here are the columns used for encodings:

```
#encoding
ordinal_columns = ['chance_to_churn_ordered', 'preferred_class_ordered', 'preferred_size_ordered', 'decade', 'tenure_group', 'wealth_segment']

nominal_columns = ['is_new_buyer', 'preferred_brand', 'preferred_line', 'is_preferred_class_missing', 'is_preferred_size_missing', \
                   'preferred_product_binned', 'gender', 'job_category', 'job_industry_category',
                   'deceased_indicator', 'owns_car', 'state']
```

```
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder

dummy_encoder = OneHotEncoder()
ordinal_encoder = OrdinalEncoder(categories=ordinal_encoder_categories)
encoded_classes = []

customer_portfolio[ordinal_columns] = ordinal_encoder.fit_transform(customer_portfolio[ordinal_columns])

for col in nominal_columns:
    encoded_data = dummy_encoder.fit_transform(customer_portfolio[[col]]).toarray()
    feature_names = dummy_encoder.get_feature_names_out([col])
    encoded_df = pd.DataFrame(encoded_data, columns=feature_names, index=customer_portfolio.index)
    customer_portfolio = pd.concat([customer_portfolio, encoded_df], axis=1)
    customer_portfolio = customer_portfolio.drop(col, axis=1)
    encoded_classes.append(dummy_encoder.categories_)
```

Train Test Split

This project will do both unsupervised clustering and supervised classification based on the cluster groups. While unsupervised clustering has no data leakage issue, it is important to split the data first before scaling for supervised classification, which I will perform afterwards.

Scaling

I also applied different scaling methods based on whether the data is skewed or not. If it is skewed, RobustScaler is applied, otherwise StandardScaler.

```
#scaling
skewed_columns = list(skewness_df[ (skewness_df['Skewness'] < -0.5) | (skewness_df['Skewness'] > 0.5) ]['Column'])
normal_columns = list(skewness_df[ (skewness_df['Skewness'] >= -0.5) & (skewness_df['Skewness'] <= 0.5) ]['Column'])
```

```python
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import StandardScaler, RobustScaler

class ColumnScaler(BaseEstimator, TransformerMixin):
    def __init__(self, scalers=None, skewed_columns=None, normal_columns=None):
        self.scalers = scalers or {}
        self.skewed_columns = skewed_columns
        self.normal_columns = normal_columns

    def fit(self, X, y=None):
        if not self.scalers:
            self.scalers['skewed'] = RobustScaler()
            self.scalers['normal'] = StandardScaler()

        if self.skewed_columns:
            self.scalers['skewed'].fit(X[self.skewed_columns])
        if self.normal_columns:
            self.scalers['normal'].fit(X[self.normal_columns])
        return self

    def transform(self, X, y=None):
        X_transformed = X.copy()
        if self.skewed_columns:
            X_transformed[self.skewed_columns] = self.scalers['skewed'].transform(X[self.skewed_columns])
        if self.normal_columns:
            X_transformed[self.normal_columns] = self.scalers['normal'].transform(X[self.normal_columns])
        return X_transformed

    def fit_transform(self, X, y=None):
        self.fit(X)
        return self.transform(X)
```

```python
# Unsupervised Clustering
scaler = ColumnScaler(skewed_columns=skewed_columns, normal_columns=normal_columns)
unsupervised_portfolio = scaler.fit_transform(customer_portfolio)
```

Feature Selection

With a total of 108 features, feature selection becomes a critical step to reduce the data complexity. By picking out only the most relevant features, not only will it contribute to faster model training and reduced computation overhead, but also simultaneously boosts the interpretability of the resulting customer segments, making them easier to understand and act upon. In this project, I will try three types of feature selection methods.

PCA (Principal Component Analysis)

PCA is one of the common methods for feature selection. It transforms the original features into a new, smaller set of entirely new features called Principal Components. These components are ordered by how much variance (or information) they capture from the original data, and importantly, they are uncorrelated with each other. My strategy is to pick components that explain 85% or 95 % of the data.

```
Number of components explaining at least 85% of variance: 16
Number of components explaining at least 95% of variance: 32
```

After trials, the best number of components for this is 16.

```
# step2. choose the number of components
n_components = 16

pca = PCA(n_components=n_components)
unsupervised_portfolio_pca = pca.fit_transform(unsupervised_portfolio)

print(f"Original number of features: {unsupervised_portfolio.shape[1]}")
print(f"Reduced number of features (principal components): {unsupervised_portfolio_pca.shape[1]}")
```

```
Original number of features: 108
Reduced number of features (principal components): 16
```

```
# create a new DataFrame with the principal components
pca_df = pd.DataFrame(unsupervised_portfolio_pca, columns=[f'PC{i+1}' for i in range(n_components)])
```

Correlation Analysis

Correlation Analysis is a straightforward, yet effective method used for feature selection by examining the linear relationships between variables. It quantifies how strongly two features move together. If two features are highly correlated with each other, it suggests they convey very similar information, meaning one of them is likely redundant. In such cases, removing one of these highly correlated features without significantly losing predictive power is often recommended. This process directly addresses multicollinearity, where features are overly dependent on each other, simplifying the model's input, improving its interpretability, and enhancing the stability of models sensitive to such dependencies.

After trials, the best threshold for this is 0.04.

```
corr_matrix = unsupervised_portfolio.corr()
```

```
# identify highly correlated features
threshold = 0.04

upper_triangle = np.triu(corr_matrix.values, k=1)
correlated_pairs = np.where(np.abs(upper_triangle) > threshold)

correlated_features = [
        (unsupervised_portfolio.columns[correlated_pairs[0][i]], unsupervised_portfolio.columns[correlated_pairs[1][i]])
        for i in range(len(correlated_pairs[0]))]

# remove one feature from each highly correlated pair
features_to_remove = set()
for feature1, feature2 in correlated_features:
    if feature1 not in features_to_remove and feature2 not in features_to_remove:

        if unsupervised_portfolio.columns.get_loc(feature1) > unsupervised_portfolio.columns.get_loc(feature2):
              features_to_remove.add(feature1)
        else:
              features_to_remove.add(feature2)

# create a list of features to keep
features_to_keep = [col for col in unsupervised_portfolio.columns if col not in features_to_remove]
```

```
# create a new DataFrame with the selected features
corr_df = unsupervised_portfolio[features_to_keep]
```

Feature Selection Based on Clustering

This involves using clustering techniques to identify redundant features. This strategy can effectively reduce the number of features by ensuring that the selected set is diverse enough to cover different aspects of the data while avoiding unnecessary complexity from highly similar features. It helps in building a more compact and efficient feature set that still captures the underlying structure of the data.

After trials, the best threshold for this is 0.001.

```
# Perform feature selection based on clustering over a range of k
cluster_range = [2, 3, 4, 5]  # Choose the range of cluster numbers to test
feature_scores = feature_selection_clustering_range(unsupervised_portfolio, cluster_range=cluster_range)

average_feature_scores = {}
for feature in unsupervised_portfolio.columns:
    total_score = 0
    for k in cluster_range:
        total_score += feature_scores[k][feature]
    average_feature_scores[feature] = total_score / len(cluster_range)

average_feature_scores = dict(sorted(average_feature_scores.items(), key=lambda item: item[1], reverse=True))
```

```
# Select the features with average scores above a threshold
threshold = 0.001 #try
selected_features = [feature for feature, score in average_feature_scores.items() if score > threshold]

clus_df = unsupervised_portfolio[selected_features]
```

Determine Number of Clusters

Among the most common approaches are partitioning methods like K-Means, which aim to divide data into a pre-defined number of clusters based on proximity to centroids. Hierarchical clustering, including Agglomerative Clustering, builds a tree-like hierarchy of clusters, either by progressively merging (agglomerative) or splitting (divisive) data points. Density-based methods such as DBSCAN, on the other hand, identify clusters as dense regions of data points separated by sparser areas.

Silhouette Score is used for defining the quality of the clusters.

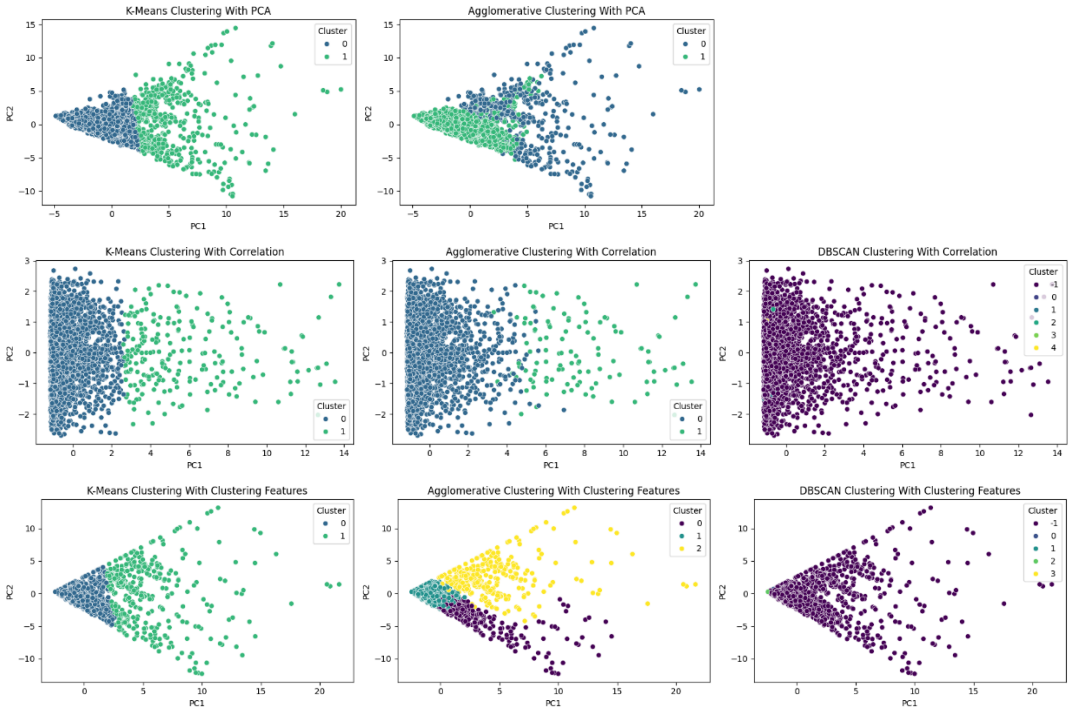
Here are the clusters based on each clustering method:

	K-Means	Agglomerative Clustering	DBSCAN
PCA features	2	2	
Correlation features	2	2	6 (eps = 0.5, min_sample = 5)
Clustering features	3	2	5 (eps = 1.5, min_sample = 5)

Finding Best Clusters

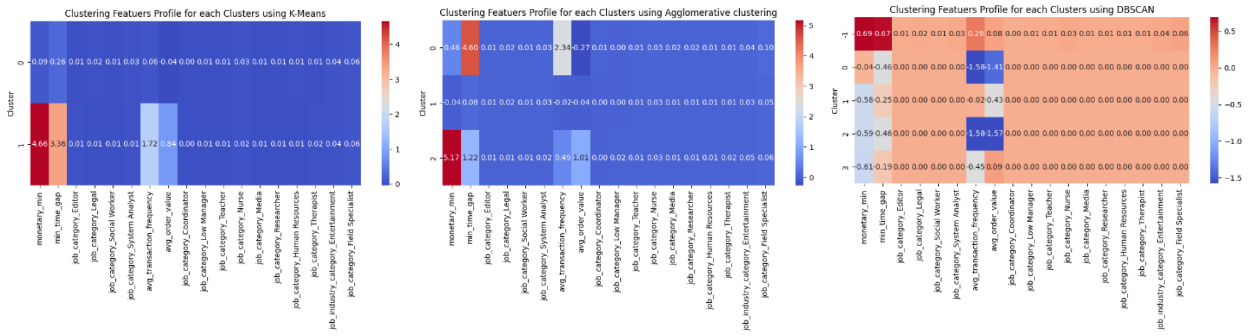
To find the best clustering result, I consider the following criteria:

1. whether the data points of the graph are clearly separated
2. whether the feature differences within each cluster are reasonable, interpretable and include both transactional and demographic features

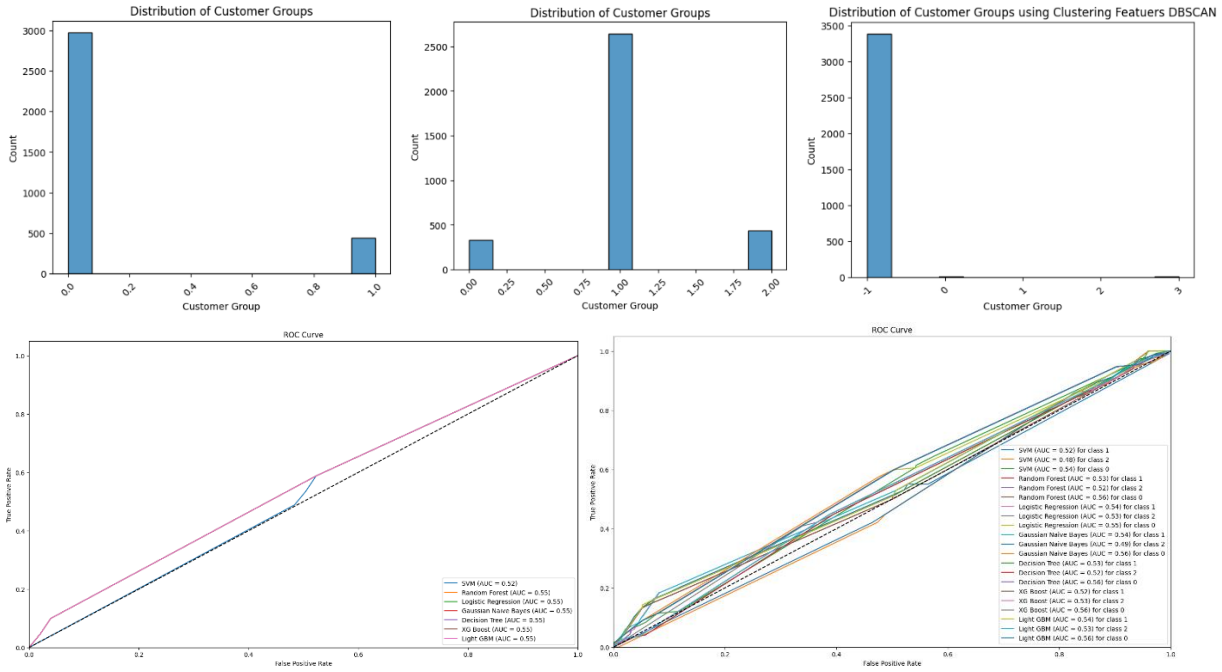


	PCA		Correlation Analysis			Clustering Features		
	K-Means	Agg	K-Means	Agg	DBSCAN	K-Means	Agg	DBSCAN
Clearly separated	O	X	O	X	X	O	△	X
Reasonable	△	△	X	△	O	O	O	O
Demographic features included	1	0	0	2	4	0	0	0
Silhouette score	0.3564	0.3347	0.5066	0.5919	0.6954	0.6281	0.5571	0.8586
Imbalance	High	High	Very	Very	Extreme	High	High	Extreme

For purely clustering, the 3 models with Clustering features seem to have better clustering outcome. Interpretation as below shown:



Although all capture the natural pattern of the data, the high imbalance classes make them not ideal for further prediction. I tried to do classification using the middle clustering result. The average accuracy is 50% for the left one and 67% for the middle one. The Area Under the Curve (AUC) for both are about 50%, indicating the model is performing no better than random guessing.



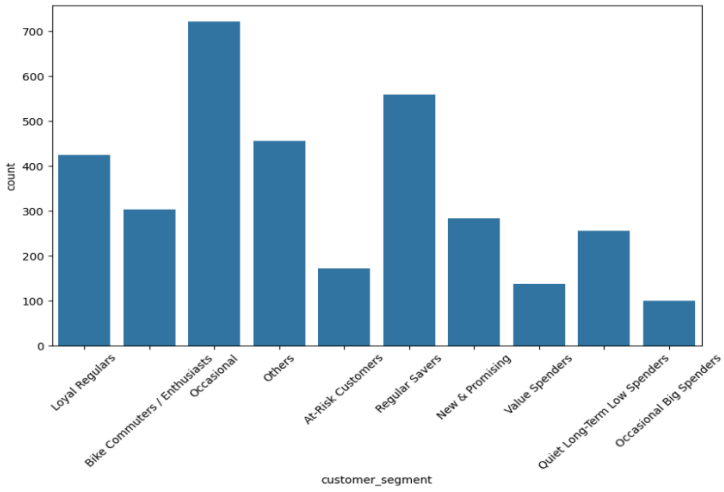
Rule-Based Clustering

Since the labels from unsupervised clustering are not ideal for classification, I decided to manually cluster them based on the EDA result. Here is the possible clustering I came up with.

Value Spenders	1. high monetary_total, high avg_order_value 2. moderate or high order_frequency 3. low last_transaction_days
Loyal Regulars	1. moderate or high monetary_total 2. good or high order_frequency 3. low last_transaction_days

Occasional Big Spenders	1. high monetary_max or high avg_order_value 2. low order_frequency
Bike Commuters / Enthusiasts	1. moderate or high monetary_total, good or high avg_order_value 2. moderate or high order_frequency 3. high past_3_years_bike_related_purchases
New & Promising	1. low last_transaction_days 2. new customer
Quiet Long-Term Low Spenders	1. average or low monetary_total 2. average or low order_frequency 3. low chance_to_churn 4. tenure customer
Regular Savers	1. average or low avg_order_value 2. good or high order_frequency
Occasional Buyers	1. average or low avg_order_value 2. average or low order_frequency
At-Risk Customers	1. middle or high chance_to_churn
Others	

Compared to the natural groupings, the new clustering groups seem much more equally distributed. I therefore decided to do classification using the newly created 10 labels.



Data Analysis Pipeline (Classification)

Data Preparation

Create Dataset

The data used is the same as unsupervised clustering, which is based on customer characteristics. However, I want to try and see if the clustering can also be applied to the other table, cleaned_customer (customer data without id and transactional records), So I start off by using only the demographic features for classification. However, the first trial returned a pretty bad result for classification (around 25% accuracy and average of 60% Area Under Curve). I decided to use full customer data and won't classify the cleaned_customer table. The detailed clustering function can be found in code.

```
supervised_portfolio = assign_customer_segments(supervised_portfolio.copy())
```

Encoding

Since there is categorical data, the next step I took is Encoding. The Encoding strategy is the same as previously: OrdinalEncoder for ordinal data; OneHotEncoder for nominal data. Now there's a total of 109 features.

Feature Selection

The feature selection consists of 2 stages: first to identify features with a statistically significant association with the clustering groups and then to quantify the strength and nature of that relationship, ensuring a robust and relevant feature set.

The initial step using Chi-Square and ANOVA helps determine if the observed relationship between a feature and the cluster assignments is likely real or simply due to random chance.

1. Chi-Square (χ^2) Test
   The Chi-Square test of independence is primarily used to determine if there's a significant association between two categorical variables. It compares the observed frequencies of data in each category against the frequencies that would be expected if the two variables were independent.
2. ANOVA (Analysis of Variance)
   ANOVA is used to test if there are any statistically significant differences between the means of two or more independent groups for a numerical variable. It compares the variance between the group's means to the variance within each group.

Identifying significancy acts as a filter to discard features that show no meaningful variation or association with the different customer groups identified by clustering. Essentially, it answers: "Does this feature even matter in distinguishing the groups?"

Once a feature is deemed statistically significant, mutual information is used to measure how much information that feature provides about the cluster assignments. It quantifies the dependency between the feature and the clusters.

1. Mutual Information (MI)
   Mutual Information is a measure from information theory that quantifies the amount of information one random variable contains about another. It measures the dependency between two variables, capturing both linear and non-linear relationships. A value of zero means the variables are independent; higher values indicate stronger dependency.

Adding this step helps in ranking the significant features by their relevance or predictive power regarding the clusters. It answers: "Among the features that matter, which ones provide the most information or have the strongest relationship with the groups?"

By using this two-stage process, I aimed to select features that are not only statistically linked to the cluster groups but also provide substantial information for differentiating and understanding these groups, leading to more meaningful and interpretable segmentation results. After performing, the final selected features are 16: brand_switching_rate, order_frequency, past_3_years_bike_related_purchases, monetary_min, monetary_total, min_time_gap, monetary_max, avg_order_value, max_time_gap, num_unique_products, chance_to_churn_ordered, avg_transaction_frequency, last_transaction_days, num_brands, tenure_group, order_value_range

```
if significant_features:
  potential_features = list(set(set(significant_features) & set(correlated_features)))
else:
  potential_features = correlated_features
```

Splitting Train Test Set

To prevent data leakage, it is important to split the data into train test set prior to the scaling process. Since this is a classification task, I will use train_test_split to randomly divide the dataset. I also include the stratify parameter to ensure both trainset and testset have equally y classes included, preventing the models from mis-learning.

I manually mapped the y so it can be reversed afterward if needed.

```
segment_map = get_segment_to_number_mapping()
print("Segment to Number Mapping:")
for name, number in segment_map.items():
        print(f"  '{name}': {number}")
```

```
Segment to Number Mapping:
  'Loyal Regulars': 0
  'Bike Commuters / Enthusiasts': 1
  'Occasional': 2
  'Others': 3
  'At-Risk Customers': 4
  'Regular Savers': 5
  'New & Promising': 6
  'Value Spenders': 7
  'Quiet Long-Term Low Spenders': 8
  'Occasional Big Spenders': 9
```

```
] """
 # If reverse mapping (number to name)
     number_to_segment_map = {v: k for k, v in segment_map.items()}
     print("\nNumber to Segment Mapping (for reference):")
     for number, name in number_to_segment_map.items():
         print(f"  {number}: '{name}'")

 """
```

Here are the X and y before splitting:

```
print(X.shape)
print(y.shape)
```

```
(3406, 16)
(3406, 1)
```

This is after splitting. I set the test size into 0.3, so that the models will have more data for testing than normally 0.2

```
X_train_pre, X_test_pre, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

```
print(X_train_pre.shape)
print(X_test_pre.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(2384, 16)
(1022, 16)
(2384, 1)
(1022, 1)
```

Scaling

The scaling strategy is also the same as the unsupervised clustering: StandardScaler for normal numerical data and RobustScaler for skewed data. The biggest difference is that the scaling should only be applied to the independent variables not including the target variable. Also, the scaler should be fit by the trainset first then apply the transforming to the testset. This is to prevent data leakage.

```
scaler = ColumnScaler(skewed_columns=skewed_columns, normal_columns=normal_columns)
scaler.fit(X_train_pre)


scale_train = scaler.transform(X_train_pre)
scale_test = scaler.transform(X_test_pre)

X_train = pd.DataFrame(
    scale_train,
    columns=X_train_pre.columns,
    index=X_train_pre.index # Use original index to ensure alignment if combining
)

X_test = pd.DataFrame(
    scale_test,
    columns=X_test_pre.columns,
    index=X_test_pre.index # Use original index to ensure alignment if combining
)

display(X_train.head())
display(X_test.head())
```

Handling Imbalanced Dataset

Although the manually labeled data are much more balanced than the unsupervised clustering groups, it is still imbalanced. SMOTE is here to address this issue. It stands for Synthetic Minority Over-sampling

Technique, a widely used method to address class imbalance in machine learning datasets. It works by creating new, synthetic data points for the minority class by interpolating between existing minority class in the feature space. I set the random state so that the SMOTE outcome is replicable.

```
import os
from imblearn.over_sampling import SMOTE

# initialize SMOTE
os = SMOTE(random_state=42)

# only oversample training dataset
X_train_smote, y_train_smote = os.fit_resample(X_train, y_train)
```

Here is the final shape of the datasets after SMOTE.

```
print(X_train_smote.shape)
print(y_train_smote.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(5770, 16)
(5770, 1)
(682, 16)
(682, 1)
```

Algorithm Candidates

For the algorithm models, I decided to test out several models. These models include:

1. Support Vector Machine (SVM, specifically svm.SVC):
   Chosen for its effectiveness in high-dimensional spaces and its ability to model complex non-linear decision boundaries using different kernels. SVMs work by finding an optimal hyperplane that best separates data points of different classes in the feature space, potentially mapping data to higher dimensions to achieve this separation.
2. Random Forest Classifier:
   Chosen for its robustness against overfitting, strong performance on diverse problems, and its inherent ability to provide feature importance measures. It's an ensemble method that builds multiple decision trees during training and outputs the mode of their predictions for classification.
3. Logistic Regression:
   Chosen for its simplicity, interpretability of coefficients, and good performance on linearly separable data. This model predicts the probability of a categorical outcome by fitting data to a logistic function.
4. Gaussian Naive Bayes:
   Chosen for its computational efficiency, simplicity, and often surprisingly good performance, especially when features are relatively independent. It's a probabilistic classifier based on Bayes' theorem, assuming features follow a Gaussian distribution and are conditionally independent given the class.
5. Decision Tree Classifier:

Chosen for its high interpretability, since the tree structure can clearly visualize the decision rules, and its capacity to capture non-linear relationships in the data. It partitions the data into subsets based on feature values to make predictions.

6. XGBoost (xgb.XGBClassifier):
   Selected for its renowned high performance and efficiency. XGBoost is an optimized gradient boosting algorithm that builds decision trees sequentially, with each new tree correcting errors of the previous ones and includes regularization to prevent overfitting.
7. LightGBM (LGBMClassifier):
   Chosen for its exceptional speed and efficiency, especially on large datasets, while often delivering performance on par with or exceeding other gradient boosting methods.

Before the main training, I used GridSearchCV and RandomSearchCV to find the best parameters for the model. The choice of parameters critically influences a model's performance and ability to generalize to unseen data, thereby mitigating the risk of overfitting (where the model learns the training data too well, including its noise) or underfitting (where the model is too simple to capture underlying patterns), ultimately aiming for improved performance on new, independent data.

The cross-validation method used for this project is Stratified K-Fold. It preserves the proportion of each class in each fold. By doing so, Stratified K-Fold provides a more reliable estimate of the model's performance.

I also split the data into validation data to evaluate whether the model is overfitting. I set the threshold to 0.05.

```
scores = []
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

X_train_full, X_val, y_train_full, y_val = train_test_split(
    X_train_final, y_train_final, test_size=0.2, random_state=42, stratify=y_train_final
)
```

```
# Use GridSearchCV if the parameter space is small, otherwise use RandomizedSearchCV
if len(param_dist) <= 5:
    search = GridSearchCV(
        model,
        param_dist,
        cv=cv,
        scoring='accuracy',
        n_jobs=-1,
        verbose=2,
        error_score='raise'
    )
    print(f"Using GridSearchCV for {model_name}")
else:
    search = RandomizedSearchCV(
        model,
        param_dist,
        n_iter=10,
        cv=cv,
        scoring='accuracy',
        random_state=42,
        n_jobs=-1,
        verbose=2,
        error_score='raise'
    )
    print(f"Using RandomizedSearchCV for {model_name}")
```

```
# Fit
search.fit(X_train_full, y_train_full)

best_model = search.best_estimator_

y_pred_train = best_model.predict(X_train_full)
y_pred_val = best_model.predict(X_val)

train_accuracy = accuracy_score(y_train_full, y_pred_train)
val_accuracy = accuracy_score(y_val, y_pred_val)

is_overfitting = (train_accuracy - val_accuracy) > overfitting_threshold

scores.append({
    'model': model_name,
    'best_model': best_model,
    'best_score': search.best_score_,
    'best_params': search.best_params_,
    'train_accuracy': train_accuracy,
    'val_accuracy': val_accuracy,
    'is_overfitting': is_overfitting
})
```

Here are the best-chosen parameters for each model:

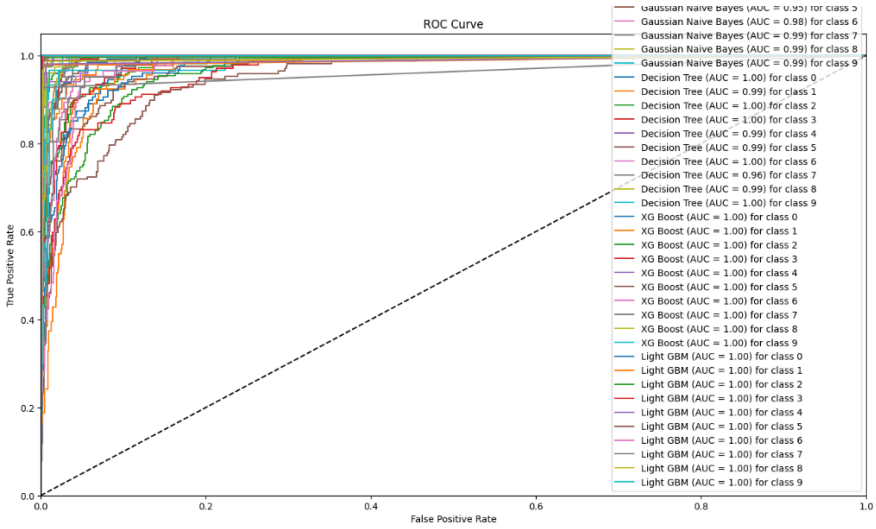
	model	best_model	best_score	best_params	train_accuracy	val_accuracy	is_overfitting
0	SVM	SVC(C=100, probability=True, random_state=42)	0.960149	{'C': 100, 'gamma': 'scale', 'kernel': 'rbf'}	0.997525	0.967327	False
1	RandomForestClassifier	(DecisionTreeClassifier(criterion='entropy', m...	0.995792	{'criterion': 'entropy', 'max_depth': 10, 'min...	1.000000	0.999010	False
2	LogisticRegression	LogisticRegression(C=100, max_iter=300, penalt...	0.875743	{'C': 100, 'max_iter': 300, 'penalty': 'l1', '...	0.890099	0.891089	False
3	DecisionTreeClassifier	DecisionTreeClassifier(criterion='entropy', mi...	0.994059	{'criterion': 'entropy', 'max_depth': None, 'm...	0.999752	0.991089	False
4	XGBClassifier	XGBClassifier(base_score=None, booster=None, c...	0.995792	{'subsample': 0.9, 'objective': 'multi:softmax...	1.000000	0.998020	False
5	LGBMClassifier	LGBMClassifier(boosting_type='dart', learning_...	0.996535	{'objective': 'multiclass', 'num_leaves': 63, ...	1.000000	0.997030	False

Model Performance

Based on all the models we applied to our dataset, we have compiled a performance summary in the table below.

	Accuracy	ROC_AUC	Is Overfitting
SVM	89.33%	99.49%	True
Random Forest	99.02%	99.99%	False
Logistic Regression	82.29%	98.58%	True
Gaussian Naive Bayes	72.11%	97.53%	True
Decision Tree	99.02%	99.30%	False
XGBoost	99.80%	99.99%	False
LightGBM	99.51%	99.99%	False

Here is the ROC Curve for each class of the models' output:



From the results above, we can conclude that: tree-based models, particularly XGBoost (99.80% accuracy), LightGBM (99.61%), and Random Forest (99.02%), demonstrated an exceptional ability to learn and predict the predefined customer segments, achieving near-perfect accuracy and ROC_AUC scores (approaching 99.99%) without indications of overfitting. The ROC curves also strongly suggests that the features used in the models are highly effective at capturing the logic embedded in the rule-based definition of the target variable y. While other models like SVM, Logistic Regression, and Gaussian Naive Bayes also produced high ROC_AUC values, their lower accuracy scores and tendency to overfit made them less optimal choices compared to the more robust and highly accurate tree-based models for this specific classification task.

Customer Segments

Unsupervised Clusters

	Portfolio
Seasonal Roadies	This segment represents the hobbyist or specialist cyclist who purchases infrequently but with specific needs in mind. Their low overall spending and long purchase cycles are offset by a clear preference for higher-performance categories like Road and Mountain bikes from brands such as Giant and Norco. Although they show little brand loyalty and have a moderate risk of churning, their targeted purchasing behavior suggests they could be engaged with content and offers related to specific cycling disciplines, events, or seasons to encourage more consistent activity.
Core Customers	This segment is the foundation of the business, comprising the largest and most loyal customer group. They are highly active and consistent, driving significant revenue through frequent purchases of a wide variety of products, particularly from the Standard line and in lower-class/smaller-size categories. Their exploratory nature and extremely low churn risk make them the ideal audience for loyalty programs, new product trials, and cross-selling campaigns designed to deepen their already strong engagement with the brand.
High-Value Shoppers	This segment consists of selective, high-spending customers who are crucial for overall profitability. While their purchase frequency is moderate, each transaction is of significant value, with a focus on premium Standard and Touring lines from brands like Giant and WeareA2B. This group is the most volatile, containing many new buyers with a high risk of churning, which indicates they are likely still in a trial or evaluation phase. The key strategy for this segment is to foster loyalty through premium after-sales service, exclusive offers, and personalized communication that reinforces the value and quality of their high-ticket purchases to convert them into long-term, stable customers.

Segment 1: Seasonal Roadies

This segment can be characterized by infrequent but specific purchasing behavior, suggesting they may be hobbyists or enthusiasts who buy for a particular need.

Purchasing Behavior & Value

> This group is definitively infrequent. They have the highest average_transaction_period and the longest max_time_gap between purchases, confirming they go long stretches without buying. Their overall financial contribution is low, as evidenced by the lowest total spending (monetary_total) and lowest maximum single purchase value (monetary_max). Their overall order frequency is the lowest of the three groups.

Product & Brand Preferences

A defining characteristic is their significantly higher preference for Road and Mountain bikes compared to the other segments. They have a high brand-switching rate, suggesting less loyalty to a single brand. But they do show the highest preference for Giant Bicycles and Norco Bicycles. This segment has a slightly lower proportion of car owners compared to the others.

Churn Risk

While mostly stable, this segment contains a noticeable portion of customers with a middle chance of churning, reflecting their less committed relationship with the brand.

Segment 2: Core Customers

This segment is the heart of the business, representing a loyal, active, and highly engaged customer base. They are the most predictable and consistent shoppers.

Purchasing Behavior & Value

This is the largest customer segment by a significant margin. They are the most frequent shoppers, confirmed by having the lowest average_transaction_period (shortest time between purchases) and the highest overall order frequency. They are explorers, purchasing the highest number of unique products and engaging with the widest variety of brands. Their total spending (monetary_total) is high, driven by the consistency and frequency of their purchases.

Product & Brand Preferences

This group strongly prefers the Standard line of products. They show a higher preference for low-class and small-size products compared to other segments.

Churn Risk

This is the most stable and loyal segment, with an overwhelmingly low chance to churn.

Segment 3: High-Value Shoppers

This segment is characterized by high-value, selective purchases. While not as frequent as Core Customers, their individual transactions are significantly more valuable, making them a crucial segment for profitability.

Purchasing Behavior & Value

The defining trait of this segment is its immense value per order. They have the highest avg_order_value and the highest monetary_min, meaning they consistently make high-value purchases and avoid low-cost items. Their purchasing is less frequent than Core Customers (higher average_transaction_period) but more frequent than Seasonal Roadies. Their total spending is high on par with Core Customers, but it's achieved through fewer, higher-ticket transactions.

Product & Brand Preferences

They show a distinct preference for Giant Bicycles and WeareA2B brands. They also have a strong preference for the Standard and Touring line of products compared to the other 2 segments. Their brand switching rate is high, suggesting they are selective and not yet loyal.

Churn Risk

This segment is the most unstable one. It contains the highest proportion of customers flagged as "new buyer" and simultaneously has the largest group with a "high" and “middle” chance to churn. This highlights their status as a high-value, high-risk segment that requires active nurturing to secure their loyalty.

Rule-Based Clusters

	Portfolio
Loyal Regulars	This large and highly valuable segment forms the stable bedrock of the business, comprised of tenured, older customers (1950s-1970s) who exhibit extreme loyalty. They are high-frequency explorers, purchasing a wide variety of unique products from many brands like Norco and Solex, resulting in a consistently high monetary total. Their low brand-switching rate and exclusively low churn risk confirm their status as a reliable and deeply engaged customer base that has been recently active.
Bike Commuters / Enthusiasts	This group of highly engaged, tenured customers (primarily from the 1960s-1980s) represents the loyal core of enthusiasts whose past bike purchases are the highest. They are frequent, high-spending shoppers with a strong preference for the Standard product line and the WeareA2B brand. Their low brand-switching rate and extremely low churn risk signify a stable and dedicated group of cycling lovers.
Mass Occasional	As the largest customer segment, this group consists of infrequent, need-based shoppers whose financial contribution is low but who remain a stable part of the customer base. Characterized by long transaction periods and a tendency to buy very few unique products, this group has a high brand-switching rate, indicating low loyalty. Despite their infrequency and a lack of tenured members, their churn risk is surprisingly low, suggesting they are a reliable, if low-engagement, group that includes a notable portion of new customers.
Regular Savers	This second largest group consists of frequent and highly loyal customers (from the 1960s-1980s), this segment is a reliable source of consistent activity, even if their individual transaction value is low. They are recent shoppers with high order frequency but low total spending, preferring popular, middle-sized products of either low or high class from brands like Norco. Their extremely low brand-switching rate and churn risk make them a predictable and stable part of the business.

New & Promising	This segment represents the next generation of potential loyal customers, as it is composed entirely of young, new buyers (from the 1990s and 2000s) showing strong initial engagement. While their spending is moderate, they have been active very recently and show a strong preference for Standard and Mountain line bikes from brands like Giant and Norco. Their exceptionally low brand-switching rate suggests they are still forming their loyalties, making this low-churn group a critical segment to cultivate.
Value Spenders	This is a cornerstone of profitability, consisting of stable, tenured, and older customers (1970s and earlier) defined by their high-value purchases. They have the highest total cost and are highly profitable on a per-transaction basis, with a significant preference for middle-class products from the Standard and Touring lines, particularly from Trek and WeareA2B. Having been active recently and possessing a very low churn risk, they are a vital, high-yield segment that also includes a healthy proportion of new customers.
Quiet Long-Term Low Spenders	This segment consists of the oldest customers (1950s-1970s), who are exclusively tenured but have become largely inactive. Their spending and order frequency are very low, but they show a preference for Road line bikes and the Giant brand. Despite their inactivity and moderately high brand-switching rate, their churn risk is very low, suggesting they are a loyal but disengaged group that could potentially be reactivated.
Occasional Big Spenders	This small but influential segment is comprised of infrequent, specialist buyers who make exceptionally large purchases when they do shop. They have the longest transaction periods and lowest order frequencies but boast the highest minimum, maximum, and average order values, favoring large-sized products from the Touring line. As a high-reward group with low churn risk, their mix of new and established customers presents a unique opportunity to cultivate loyalty among high-value, niche consumers.
At-Risk Customers	This segment represents a group of formerly active customers who are now on the verge of churn. They are defined by their prolonged inactivity, low order frequency, and low total spending. Despite having once been active, their high brand-switching rate and preference for specialist Mountain and Road line bikes from Norco did not translate into lasting engagement. Their middle-to-high churn risk confirms their disengaged status, making them a priority for re-engagement or churn-prevention strategies.
Average Buyers	This third largest, diverse group functions as the baseline for a typical customer, exhibiting moderate and stable behavior across most metrics. They have high order frequency, and a significant total expenditure, preferring popular, middle-class, medium-sized products from the Standard line and showing interest in brands like Solex, Trek, and WeareA2B. With a very low churn risk and a demographic profile that spans all ages and tenure groups, they represent the stable, mainstream core of the business.

Segment 1: Loyal Regulars

This is a large and highly valuable segment, characterized by extreme loyalty and high-frequency purchasing. They are explorers who engage deeply with a wide variety of products and brands, forming the stable, high-activity bedrock of the customer base.

Purchasing Behavior & Value

> They have the highest order_frequency and the lowest last_transaction_days of any group, indicating they purchase often and have done so very recently. Their monetary_total is consistently high. They purchase the highest num_unique_products and from the most num_brands, but they also have relatively low brand-switching rate.

Product & Brand Preferences

> Demographically, they are well-established customers, with a high proportion of "average" and "tenured" members. They are predominantly from older decades (1950s-1970s). Their brand preferences varied, but they show a preference for Norco Bicycles and Solex.

Churn Risk

> Their chance_to_churn_ordered is exclusively "low," making them an extremely stable and reliable segment.

Segment 2: Bike Commuters / Enthusiasts

This group represents highly engaged, tenured customers whose purchase is almost entirely focused on bike-related products. They are loyal, frequent, and high-spending enthusiasts.

Purchasing Behavior & Value

> They are frequent purchasers with high order_frequency and high monetary_total. Their defining characteristic is that nearly all of their purchases in the last 3 years have been bike-related, more so than any other group.

Product & Brand Preferences

> Demographically, they tend to be older (skewing towards the 1960s-1980s decades). They have a strong preference for the Standard product line. They are brand-loyal, with a low brand switching rate, and show a preference for WeareA2B.

Churn Risk

> Their churn risk is overwhelmingly low, indicating strong loyalty.

Segment 3: Mass Occasional

This is the largest customer segment, consisting of infrequent shoppers who interact with the business on a need-basis. While their financial contribution is low, they are a stable part of the customer base.

Purchasing Behavior & Value

They have a long average transaction period and a relatively high number of days since their last transaction. Their order_frequency is the lowest of all segments, and their total spending (monetary_total) is also correspondingly low. They have also the fewest tenured customers.

Product & Brand Preferences

They have similar age distribution to Loyal Regulars but have a higher brand switching rate, indicating low loyalty to any particular brand. They also buy the fewest unique products.

Churn Risk

Despite their infrequency, their churn risk is surprisingly low, suggesting they are a stable, though low-engagement, segment. It is important to note that they have a proportion of new customers as well.

Segment 4: Regular Savers

This segment is comprised of frequent, loyal, and reliable customers who consistently purchase lower-value items. They are a stable source of activity, even if their individual transaction value is low.

Purchasing Behavior & Value

They have a high order_frequency and a very low number of days since their last transaction. However, their monetary_total, monetary_max, and avg_order_value are all significantly lower than high-spending groups.

Product & Brand Preferences

They prefer low- or high-class products, middle-sized, and show a strong preference for Norco Bicycles and popular products. Demographically, they have similar age distribution to Bike Commuters / Enthusiasts (skewing towards the 1960s-1980s decades) and are highly loyal with a low brand switching rate.

Churn Risk

Their churn risk is extremely low.

Segment 5: New & Promising

This segment is made up entirely of new customers who are young and show strong initial engagement. They represent the next generation of potential loyal customers.

Purchasing Behavior & Value

They have a very low last transaction day. Although the order_frequency varies quite hugely, their spendings (monetary_total and avg_order_value) are moderate.

Product & Brand Preferences

This is the youngest group, dominated by customers born in the 2000s and 1990s, especially the latter. They show a strong preference for the Standard product line and have a high preference in the mountain line as well compared to other segments. They have the lowest brand switching

rate of all segments, suggesting they are still establishing their preferences (Giant Bicycles and Norco Bicycles).

Churn Risk

Their churn risk is low, and they also have the highest proportion of customers flagged as "new buyers," highlighting the critical opportunity to cultivate their initial loyalty.

Segment 6: Value Spenders

This is a highly valuable and stable segment of tenured customers defined by their high-value purchases. They are the most profitable customers on a per-transaction basis.

Purchasing Behavior & Value

They have the highest monetary_total, highest monetary_max and a very high avg_order_value. While their order frequency is moderate, each transaction is highly profitable. They also have the lowest last transaction days.

Product & Brand Preferences

Demographically, they are primarily tenured customers from older decades (1970s and earlier), having similar age distribution to Bike Commuters / Enthusiasts. They have relatively high new customer proportions compared to most groups. They show a significant preference for middle-class products. They also have a strong preference for the Standard lines, Trek Bicycles and WeareA2B.

Churn Risk

Their churn risk is very low, making them a cornerstone of profitability.

Segment 7: Quiet Long-Term Low Spenders

This segment consists of inactive yet tenured customers. They are loyal but have become inactive over time.

Purchasing Behavior & Value

Their average transaction period varies quite a lot. They have very low monetary_total, and pretty low average order value and order_frequency.

Product & Brand Preferences

Demographically, they are the oldest segment, dominated by those born in the 1950s to 1970s, especially the proportion of 1970s are highest among others. They are exclusively "tenured," indicating they have been with the brand for quite a year. They have a high preference for Giant Bicycles but have a moderately high brand switching rate. They also have the higher preference of Road line bikes compared to others.

Churn Risk

Despite their inactivity, their churn risk is very low, suggesting there may be an opportunity to re-engage them.

Segment 8: Occasional Big Spenders

This is a small but influential segment of highly infrequent, specialist buyers who make very large purchases when they do shop. They are a high-risk, high-reward group.

Purchasing Behavior & Value

They have a larger proportion of the longest average_transaction_period compared to the other segments and lowest order frequencies. However, they have the highest monetary_min, monetary_max, and avg_order_value of any segment.

Product & Brand Preferences

They purchase very few unique products from very few brands. They favor Touring lines products and prefer either low- or high-class products, but the product sizes are mostly large and not commonly bought by others. They also have a significant proportion of new customers compared to other groups (besides New & Promising).

Churn Risk

This segment also has a pretty low churn risk, but it is important to note that they also have a few new customers.

Segment 9: At-Risk Customers

This segment represents customers who are on the verge of churning. They are defined by their prolonged inactivity and high churn probability.

Purchasing Behavior & Value

They have, by far, the highest number of last_transaction_days and the lowest order frequencies, indicating they have not made a purchase in a very long time. However, their average transaction period is not very high, suggesting they were once active but have since become disengaged. Their monetary_total is among the lowest.

Product & Brand Preferences

Demographically, they are a mix of ages from all decades. Their unique products and brands purchased are relatively low, but they have quite a high brand switching rate. They favor Norco Bicycles, Mountain and Road line bikes, and prefer either low- or high-class products

Churn Risk

The most telling characteristic is their chance_to_churn_ordered, which is almost entirely "middle" and "high," confirming their at-risk status.

Segment 10: Average Buyers

This is a large, diverse group that serves as a general baseline for the average customer. They are stable and exhibit moderate behavior across most metrics.

Purchasing Behavior & Value

They have quite high order_frequency, monetary_total, monetary_max, and moderate avg_order_value. Their past 3 years bike related purchases have been slightly lower.

Product & Brand Preferences

They spread across all decade groups besides 1930s and 1940s and all tenure groups. Their product preferences are heavily skewed towards the middle-class, medium-size, Standard line and products that are often bought by other people. Their brand switching rate is moderate, but they show a higher preference for Solex, Trek Bicycles and WeareA2B.

Churn Risk

Their churn risk is very low.

Marketing Strategy

Each of the two different clustering methods has its pros and cons. The segments from the unsupervised clustering most represent the natural patterns within the existing data, making them ideal for developing deep customer personas and informing high-level strategic planning. On the other hand, rule-based segments achieve high accuracy in classification and robustness against overfitting, which provides strong confidence for ensuring long-term consistency in new or future customer targeting. Therefore, I developed two marketing strategies for these two purposes.

For current customer base strategy

Segment 1: The Seasonal Roadies - "The Pro Shop Experience"

Goal

Capture their specific, high-intent purchases and build trust to become their go-to for specialized needs.

Product & Brand Focus

1. Targeted Campaigns: Create marketing campaigns centered around Road and Mountain bikes, specifically featuring Giant and Norco Bicycles. Launch these campaigns during peak seasonal periods (spring and fall) when they are most likely to be in the market.
2. "Pro-Level" Content: Develop expert content like guide to upgrade road bike or mountain bike brands comparison to position the brand as an authoritative source.

Engagement & Retention

1. Service-Oriented Messaging: Since they are infrequent buyers, I believe they focus more on service and expertise. Offering customizable after-sale services and pre-sale consulting may have the chance to act as the final trigger for them.
2. Re-engagement Campaigns: While they have long gaps between purchases, given the fact that their chance to churn is low, ensuring the company will still be their first choice for future purchases is essential. The strategy should focus on value-driven content and subtle brand presence, not aggressive sales. For instance, instead of sending sales emails or promotional emails, they can send content-rich newsletters timed for the start of spring or autumn. This could reinforce their brand as a helpful expert either!
    (1) Celebrate Purchase Anniversaries: A simple, automated email sent on the anniversary of their last major purchase can be a powerful, personal touch that reminds them of the quality of their previous purchase. (Something like, it has been XXX years since you got your XXX! Here are some tips to XXXX)
    (2) Personalized Upgrade Path Content: Based on their last purchase, create content that anticipates their next logical need. If they bought a mid-range road bike three years ago, sending a guide on when the best time is to upgrade is a good consideration. This plants a seed for a future high-value purchase.

(3) Targeted New Model Announcements: Tag customers based on their preferred brands when a new model is released from that specific brand, send them a dedicated, high-quality announcement. This respects their known preference and treats them like an insider.
(4) Exclusive "Welcome Back" Incentive: Instead of frequent discounts, create offers for returning customers (something like, as a returning customer, enjoy this benefit or privilege). Make the offer like a value-added service that acknowledges their loyalty and previous investment.
(5) Accessory-Focused Promotions: During the long gaps, they may need smaller items. Send them a targeted promotion once a year focused on high-margin accessories relevant to their last purchase, like new tires, lights, or cycling computers, to maintain a soft connection to the brand.

Segment 2: The Core Customers - "The Loyalty & Discovery Program"

Goal

Reinforce loyalty, increase their already high frequency, and leverage their engagement to drive discovery of new products.

Product & Brand Focus

1. Loyalty Rewards: Implement a points-based loyalty program that rewards their high frequency. Offer exclusive early access to new Standard line products.
2. Personalized & Exclusive Access: Provide proactive, value-added recommendations that feel like a privilege of membership, not a sales pitch.
   (1) Instead of generic automated emails, provide intelligent, contextual suggestions that feel like advice from a trusted expert. The suggestion should focus on delivering the sense of “helping the customer create and discover the best value for them”
   (2) Giving them exclusive access to new products, framing it as a community privilege.

Engagement & Retention

1. Community Building: Make them feel like insiders transform their loyalty from being transactional (based on products and price) to being emotional (based on identity and belonging). To truly make them feel like insiders, the program should be built on three pillars: Exclusive Knowledge, Genuine Influence, and Recognition.
2. Predictable Value and Surprise Rewards:
   (1) Instead of a discount, offer a predictable value-added each month. The offer could rotate every fixed period. By doing so, it still creates a monthly reason to check in and purchase, but it protects the perceived value of your products and often introduces them to new items.
   (2) Gamification for Loyalty Rewards: Reward consistency directly. This encourages the habit of purchasing, not just waiting for a sale. Can also implement unexpected rewards. Once a quarter, randomly select, or a piece of new merchandise "just to say thanks." The positive word-of-mouth from this is far more valuable than a site-wide sale.

Segment 3: The High-Value Shoppers - "The VIP Nurturing Program"

Goal

Convert this high-value, high-risk segment into loyal brand advocates by providing a premium, personalized experience that justifies their significant investment.

Product & Brand Focus

1. Premium Brand Showcase: Create dedicated marketing channel that highlight the quality and innovation of Giant Bicycles and WeareA2B. Focus on storytelling around the craftsmanship and performance of the Standard and Touring lines.
2. High-Value Bundles: Offer curated product bundles that cater to their high-ticket preference. For example, a "WeareA2B Urban Commuter Package" that includes a premium bike, lock, and apparel.

Engagement & Retention (CRITICAL)

1. Personalized Onboarding: Since this segment has many new buyers and a high churn risk, implement a VIP onboarding email sequence immediately after their first purchase. This should include a personal welcome, information about product care, and an introduction to the brand's value.
2. Concierge Service: Offer a dedicated customer service channel for this segment. Proactively reach out to them post-purchase to ensure satisfaction and offer personalized advice.
3. Exclusive Access: Provide exclusive invitations to virtual events with brand ambassadors or product designers to foster a sense of belonging and justify their premium spending.

For a scalable and future-proof strategy

This will pivot from a defensive posture to an offensive growth strategy focused on hyper-personalization, community building, and value-added services. This will accelerate the positive profitability trends in our youngest customers and establish our brand as the definitive choice for the next generation.

1. Stable Core: Our established customer segments (born in the 1990s and earlier) remain consistent and highly profitable.
2. Emerging Growth: The youngest segment (born in the 2000s), identified as Segment 5: New & Promising, shows steadily growing profitability, a key indicator of future potential.

This strategy will be executed through three core pillars: Cultivate the Future, Maximize the Core, and Activate & Retain.

Pillar 1: Cultivate the Future (The Growth Engine)

This is the highest priority pillar, directly addressing the strategic findings.

Segment 5: New & Promising

The goal is onboard and lock in the loyalty of this young, highly engaged segment to convert their initial interest into long-term, high-value relationships. Possible tactics include:

1. "First Year" Onboarding Journey: Create a dedicated 12-month email and social media campaign triggered by their first purchase. This includes a welcome message, setup guides for their new bike (Mountain, Standard), and introductions to the Giant Bicycles and Norco Bicycles brands they prefer.
2. Community Integration: Drive them towards a dedicated Discord server or private social media group for "Next Gen Riders." This space will foster community, allow them to share experiences, and connect them directly with the brand in a way that feels authentic to their generation.
3. Skill-Building Content: Develop content tailored to their interests. This positions us as a partner on their cycling journey.
4. Loyalty Milestone Reward: After one year, grant them "Pro" status in the loyalty program, unlocking exclusive perks and acknowledging their commitment.

Pillar 2: Maximize the Core (The Profitability Engine)

These segments are the bedrock of the business. The goal is to reinforce their loyalty and strategically increase their value.

Segment 1: Loyal Regulars

The goal is to reward their extreme loyalty and empower them to become brand advocates. Possible tactics include:

1. VIP "Explorer's Club": Acknowledge their high-frequency, multi-brand purchasing with an exclusive loyalty tier. Offer perks like early access to new products from all brands, especially Norco Bicycles and Solex.
2. Product Review & Feedback Program: Leverage their exploratory nature by inviting them to be the first to review new items, making them feel valued insiders.

Segment 2: Bike Commuters / Enthusiasts

The goal is to serve their specialist needs and solidify our position as their go-to resource for all things bike-related. Possible tactics include:

1. Service-Oriented Content: Provide high-value content around bike maintenance, commuting tips, and deep dives into the Standard product line and WeareA2B brand.
2. Personalized Service Reminders: Based on purchase dates, send reminders for yearly tune-ups or component checks, adding value beyond the sale.

Segment 6: Value Spenders

The goal is to acknowledge and retain these highly profitable customers with a premium, white-glove experience. Possible tactics include:

1. Concierge Service: Offer a dedicated customer service line or email for this segment to handle inquiries and provide personalized recommendations.

2. Exclusive Previews: Grant them exclusive early access to new high-end Standard line products and releases from Trek Bicycles and WeareA2B.

Segment 4: Regular Savers

The goal is to nurture their consistent activity and encourage incremental growth in their average order value. Possible tactics include:

1. Gamified Loyalty: Implement "streak" bonuses for consecutive monthly purchases or "spend and get" promotions that offer a desirable mid-range product as a reward.
2. "Step-Up" Recommendations: When they purchase a Norco bike, gently introduce them to the benefits of slightly higher-tier accessories that enhance their experience, framing it as a smart upgrade.

Segment 10: Average Buyers

The goal is gently nudging this large, stable group towards higher-engagement behaviors.

1. Personalized Nudges: Analyze their purchase history to identify leanings. If they buy a Trek or WeareA2B product, send them content usually reserved for "Value Spenders" or "Bike Commuters" to draw them into a more specialized funnel.

Pillar 3: Activate & Retain (Opportunity & Risk Management)

This pillar focuses on extracting value from infrequent buyers and mitigating churn.

Segment 8: Occasional Big Spenders

The goal is to be their undisputed choice for their next major, high-value purchase.

1. High-Touch, Low-Frequency Outreach: Do not bombard them. A semi-annual, high-quality "lookbook" showcasing the latest in Touring lines and large-size frames is ideal.
2. Consultative Approach: Offer a "Personal Shopping Consultation" service to help them plan their next significant investment, reinforcing our expertise.

Segment 7: Quiet Long-Term Low Spenders

The goal is to re-engage these inactive but tenured customers with a compelling reason to return.

1. Nostalgic Re-engagement Campaign: Launch a campaign titled "It's Been a While." Acknowledge their long history with the brand and offer a significant, value-added incentive like a free professional tune-up on their old Giant bike or an exclusive discount on a new Road line model.

Segment 3: Mass Occasional

The goal is to increase top-of-mind awareness to capture their need-based purchases.

1. Broad-Reach Seasonal Campaigns: Target them with general brand awareness campaigns during peak seasons.
2. Showcase Variety: Since they have a high brand switching rate, marketing should emphasize the breadth of brands and products available to position us as a one-stop shop.

Segment 9: At-Risk Customers

The goal is to make a final attempt to win back these customers and gather valuable feedback.

1. Aggressive Win-Back Offer: Target them with a time-sensitive, high-discount offer. This is a last-ditch effort.
2. Feedback Survey: For those who don't respond, send a simple survey: "Help us understand what went wrong." The data gathered on why they left is extremely valuable for preventing future churn.