**CECS 550: Pattern Recognition**

**Repeat Buyer’s Prediction for E-Commerce**

**Prepared and Submitted by:**

**Nikita Lalwani**

**Reema Pandya**

**Rishwa Patel**

**Rohan Borad**

**Varun Gunda**

**Instructor: Mahshid Fardadi**

# **Abstract**

While promotions are frequently utilized by businesses to draw in prospective customers, many of these consumers are only drawn to one-time offers. As a result, there is little impact of promotions on long-term sales. Merchants must distinguish between one-time purchasers and potential loyal consumers and concentrate their efforts on converting the latter in order to maximize their ROI and reduce the cost of promotions.

A dataset on promotional shopping events from an e-commerce platform is made available by the project. Objective is to create a system that will lower promotional expenses, detect one-time customers, and improve ROI by forecasting the likelihood that new customers would make another purchase from the same merchant within six months.

# **Data preprocessing**

Step 1. Feed data into pandas dataframes

Step 2. Rename the seller\_id column in user behavior logs to merchant\_id

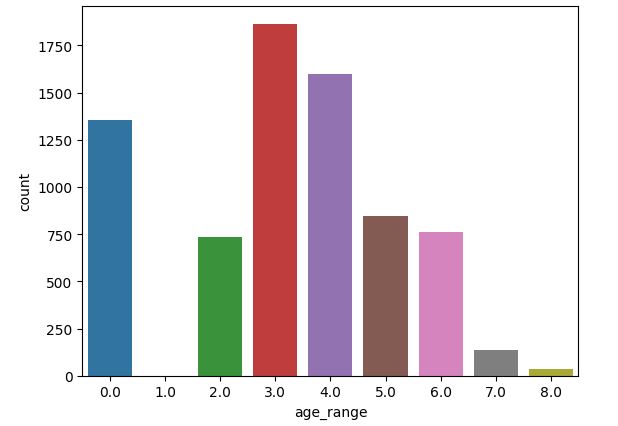
Step 3. Take dataset from item id 641 to 800 inclusively

Step 4. Combine user behavior logs with user\_profile and training data

Step 5. Check for duplicates and null values in dataframe, also check for distribution of target variable in the training data

# **Data visualization**

## **3.1 Graph for age range column**

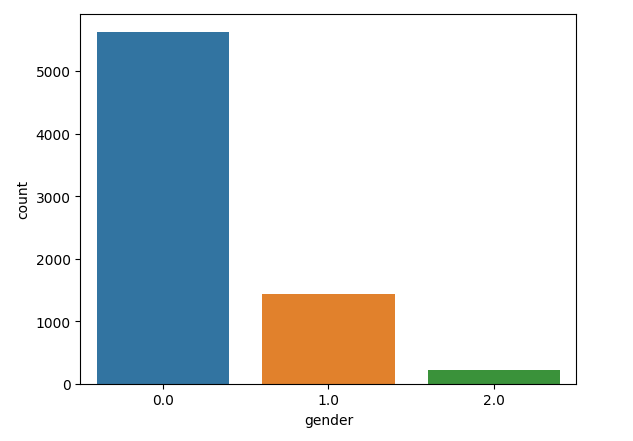
****

The graph indicates that the majority of buyers fall within the age range of 25-29 years old, which suggests that this age group is the most active consumer group. This could be due to a variety of factors such as having more disposable income, being more tech-savvy and comfortable with online shopping, or being in a life stage where they are more likely to make significant purchases, such as purchasing a home or starting a family.

The absence of shoppers below the age of 18 may be due to legal restrictions on their ability to make purchases independently, or it could be because they are still dependent on their parents for financial support and are not yet in a position to make significant purchases on their own.

Overall, the age distribution of shoppers can provide valuable insights for businesses looking to target specific age groups with their marketing and sales strategies.

## **3.2 Graph for gender column**



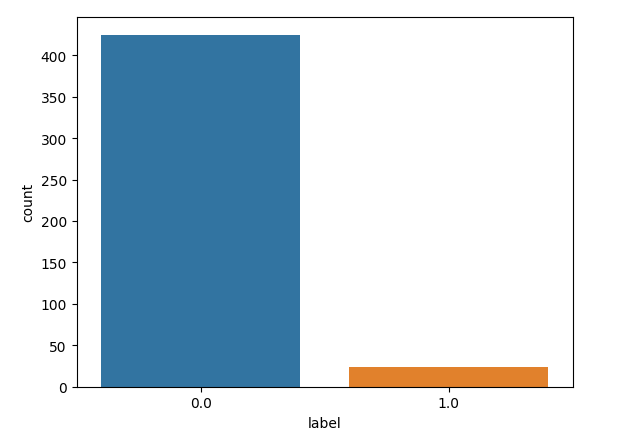
The graph represents the distribution of gender within a dataset. It shows the number of instances of each gender category within the dataset.

In this dataset, there are three gender categories represented: female (0), male (1), and unknown (2). The vertical axis of the graph shows the count or frequency of each gender category, while the horizontal axis shows the gender category itself.

According to the graph, the majority of the customers in the dataset are female, with a count of approximately 5,800. The second largest group is male customers, with a count of around 1,800. The smallest group is unknown customers, with a count of approximately 200.

It's worth noting that the reason for the unknown gender category could be due to missing or incomplete data, or it could be intentionally left blank for privacy reasons.

## **3.3 Graph for label column**

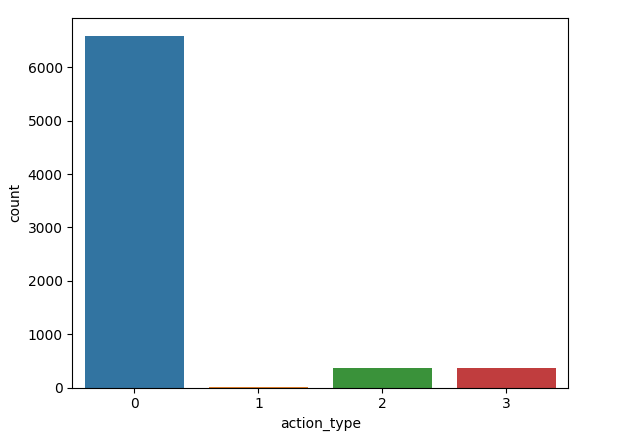


This statement is referring to a binary classification problem where a merchant is trying to predict whether a customer will be a repeated buyer or not, based on historical data.

In this context, the label "1" represents customers who have made purchases from the merchant multiple times in the past, and are thus considered to be repeated buyers. On the other hand, the label "0" represents customers who have only made a single purchase from the merchant, or who have never made a purchase at all, and are therefore not considered to be repeated buyers.

To make this prediction, the merchant may use various features or variables related to the customer's behavior and characteristics, such as their purchase history, frequency of visits, demographics, and other factors. These features are used to train a machine learning model that can learn patterns and relationships in the data, and make accurate predictions about whether a customer is likely to become a repeated buyer or not.

## **3.4 Graph for action type column**



Based on the given information, we can assume that there is a tracking system in place to monitor user behavior on a website or application. The tracking system assigns categories to user actions based on their level of engagement with the website/application. These categories are:

- Category 0 (click): When a user clicks on a link or button, but does not perform any further action.

- Category 1 (add-to-cart): When a user adds a product to their cart.

- Category 2 (purchase): When a user completes a purchase.

- Category 3 (add-to-favorite): When a user adds a product to their favorites list.

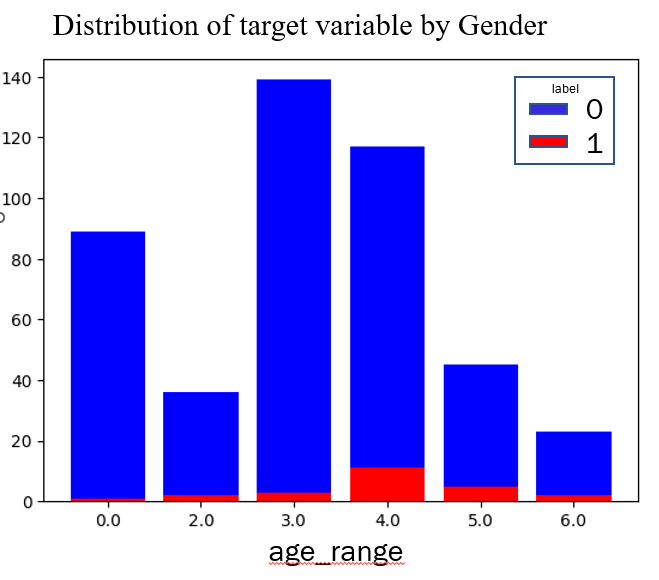
The information suggests that a large percentage of users fall into the category 0 (click), indicating that many users are browsing or exploring the website/application but not taking any significant actions such as adding products to the cart, making a purchase or saving products as favorites.

On the other hand, the minimum percentage of users falls into category 1 (add-to-cart), indicating that a relatively small number of users are actively engaged with the website/application and adding products to their cart.

Interestingly, categories 2 (purchase) and 3 (add-to-favorite) have the same level of engagement. This could mean that while users are not adding products to their cart frequently, they are either making purchases or saving products as favorites at a similar rate.

Overall, the data suggests that there may be opportunities to encourage users to move from category 0 to category 1, by providing incentives or making it easier for users to add products to their cart. Additionally, the similarity in engagement levels between categories 2 and 3 could indicate that users are more likely to make a purchase if they save products as favorites, so optimizing the favorites feature could be beneficial.

## **3.5 Distribution of target variable by age range**



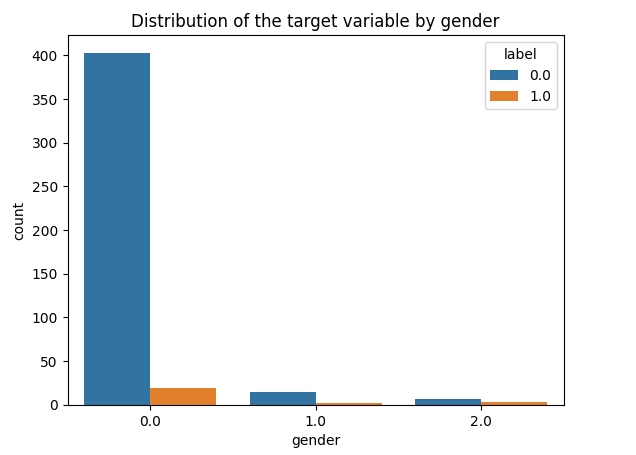
The given statement is talking about two different age ranges in relation to buying behavior: 30-34 and 25-29.

The first part of the statement suggests that the age range of 30-34 contains the maximum number of buyers. This means that people within this age range are more likely to make purchases compared to other age ranges. This could be because they have a higher income, more disposable income, or are at a stage in their life where they need to make more purchases (such as buying a house or starting a family).

The second part of the statement suggests that the age range of 25-29 contains the maximum number of one-time buyers. This means that people within this age range are more likely to make a purchase once, but not return for future purchases. This could be due to a variety of factors, such as being in a transitional period of their life, having limited disposable income, or not having established brand loyalty yet.

Overall, these two age ranges demonstrate different buying behaviors. The age range of 30-34 represents consistent buyers who are likely to make multiple purchases, while the age range of 25-29 represents one-time buyers who may require additional incentives to return for future purchases.

## **3.6 Distribution of target variable by gender**



The statement suggests that female customers exhibit a higher likelihood of being both repeated buyers and one-time buyers compared to other customer groups, while customers in an unknown category tend to have similar proportions of one-time and repeated buyers.

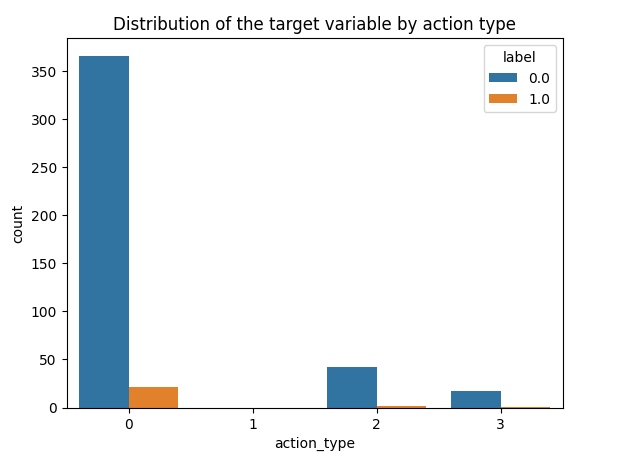
One possible explanation for female customers being more likely to be repeated buyers is that they may have a stronger brand loyalty or emotional attachment to the products they purchase. Female consumers may also be more likely to engage in repeat purchasing behavior as a result of their perceived need for or interest in certain types of products or services.

At the same time, the fact that female customers are also likely to be one-time buyers suggests that they may be more open to exploring different brands and products. Female consumers may also be more influenced by factors such as price, convenience, and promotional offers when making purchasing decisions, which could lead them to switch to different brands or products over time.

As for the unknown category, it is unclear why this group exhibits similar proportions of one-time and repeated buyers. It is possible that the unknown category includes a diverse range of customers with varying needs, preferences, and purchasing behaviors, which could explain the lack of a clear pattern in their buying habits.

Overall, the statement highlights the importance of understanding customer demographics and behavior in order to develop effective marketing strategies that cater to different segments of the market. By identifying which groups are more likely to be repeated buyers or one-time buyers, businesses can tailor their messaging, promotions, and product offerings to better meet the needs and preferences of their target customers.

## **3.7 Distribution of target variable by action type**



It appears that you are referring to a classification of user actions or behaviors on a website or application. The four categories mentioned are:

- 0 (click)

- 1 (add-to-cart)

- 2 (purchase)

- 3 (add-to-favorite)

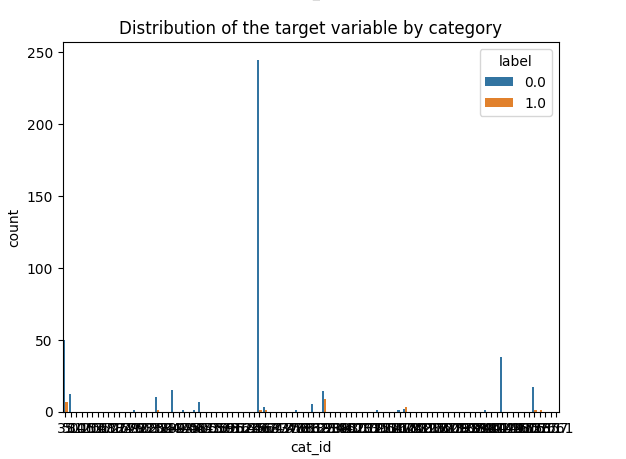
Based on the information provided, it seems that the majority of users fall into the category of 0 (click), which suggests that they are mainly browsing or viewing items on the website or application without taking further action. This could indicate that the website or application may need to improve its user experience or provide more compelling reasons for users to take action. Amongst the majority of click users majority are one time buyers and few are repeated buyers in comparison.

On the other hand, the minimum percentage of users fall into the category of 1 (add-to-cart), indicating that a relatively small number of users are adding items to their shopping carts. This could be due to various reasons, such as unclear product descriptions or a complicated checkout process.

Interestingly, both categories 2 (purchase) and 3 (add-to-favorite) have similar levels of user buying characteristics. Most of them in this category are one time buyers only.

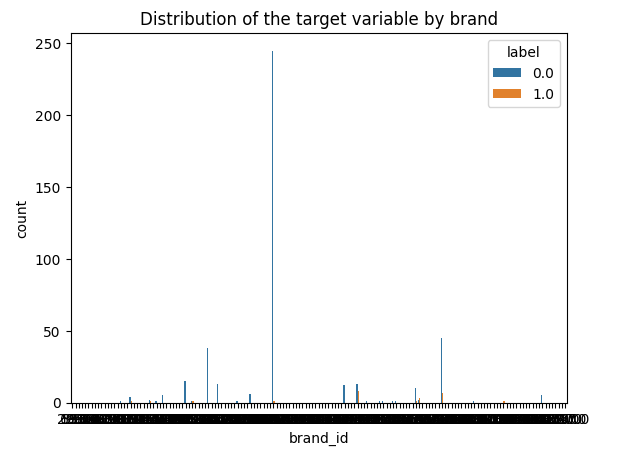
Overall, it is important to analyze and understand user behavior in order to optimize the user experience and drive conversions. By identifying areas of improvement and tailoring the website or application to the needs of users, it is possible to increase engagement and ultimately achieve business goals.

## **3.8 Distribution of target variable by category**



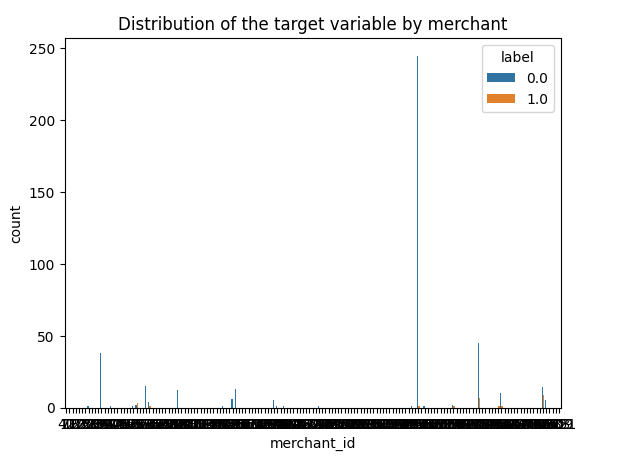
The above bar chart demonstrates the distribution of the target variable against the cat\_id where 0(blue) represents the non repetitive buyer and 1(orange) denotes repetitive buyers.

## **3.9 Distribution of target variable by brand**



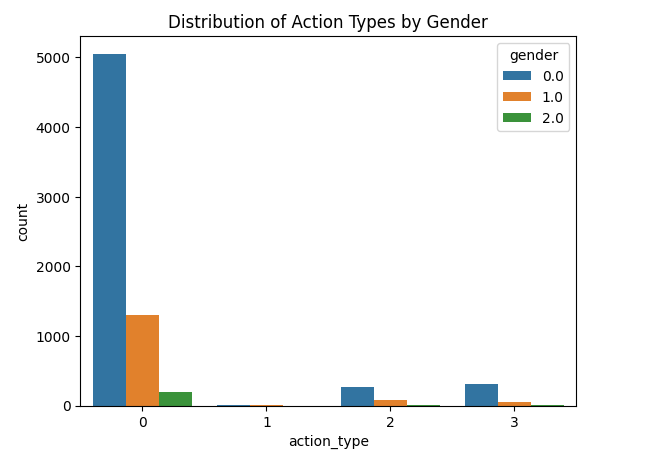
The above bar chart depicts the distribution of the target variable against the brand\_id where 0(blue) represents the non repetitive buyer and 1(orange) denotes repetitive buyers.

## **3.10 Distribution of target variable by merchant**



The above bar chart depicts the distribution of the target variable against the merchant\_id where 0(blue) represents the non repetitive buyer and 1(orange) denotes repetitive buyers.

## **3.11 Distribution of Action types by gender**



Based on the given statement, it seems that there is data available about user behavior in relation to a certain product or website. The data suggests that there are different categories of user actions being tracked, such as clicks, adding items to the cart, purchase and adding items to favorites.

It appears that the click category has the highest number of users, which suggests that many people are interested in exploring the product or website but may not necessarily be ready to make a purchase. This could be due to a variety of reasons, such as needing more information about the product, comparing prices with other sites, or simply browsing without a clear intention to buy.

The statement also mentions that the highest percentage of users in the click category are females. This could indicate that the product or website is more appealing to women, or that women are more likely to engage in browsing behavior.

On the other hand, the add to cart category has negligible customers. This means that very few people are actually adding items to their shopping cart, even though they are clicking on the product or website. This could be a concern for the website or business, as it suggests that there may be a problem with the shopping experience, such as unclear pricing or confusing checkout process.

The statement suggests that a larger proportion of people who make purchases and add items to their favorite category are female, while relatively fewer males do so.

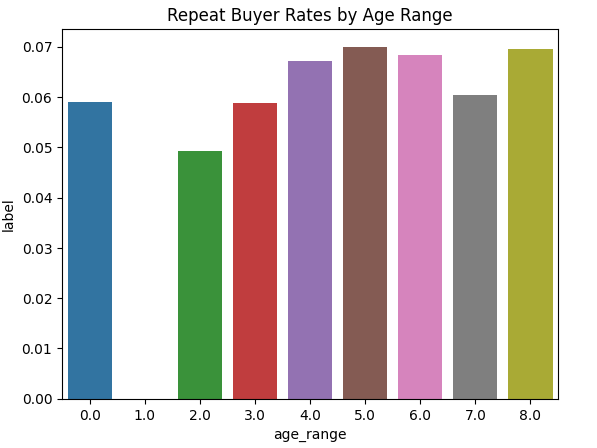
There could be several reasons for this observed trend. One possibility is that the types of products or services that are typically purchased or added to the favorite category appeal more to women than men. For example, if the products are related to fashion, beauty, or home decor, it's possible that more women would be interested in purchasing or adding them to their favorite category.

Another possibility is that there are societal or cultural factors at play that influence men and women's shopping behaviors differently. For example, women may be more likely to be interested in shopping as a leisure activity, while men may prioritize other activities such as sports or video games. Additionally, there may be differences in the way that men and women are socialized to think about and prioritize spending their money.

It's worth noting that while the statement suggests a trend, it's important to avoid making generalizations or assumptions about any individual's behavior based on gender. People's shopping habits are influenced by a wide range of factors beyond their gender, including their age, income, interests, and values.

Lastly, the statement mentions an unknown category that forms the list. It's unclear what this category refers to, but it could be an indication that there are other user behaviors being tracked that are not explicitly mentioned. Without more information, it's difficult to draw any conclusions about this category.

## **3.12 Repeat Buyer rates by Age Range categorisation**



The given bar chart shows the repeat buyer rates across different age ranges. The age ranges are divided into seven categories, with "0" and "NULL" representing unknown age ranges.

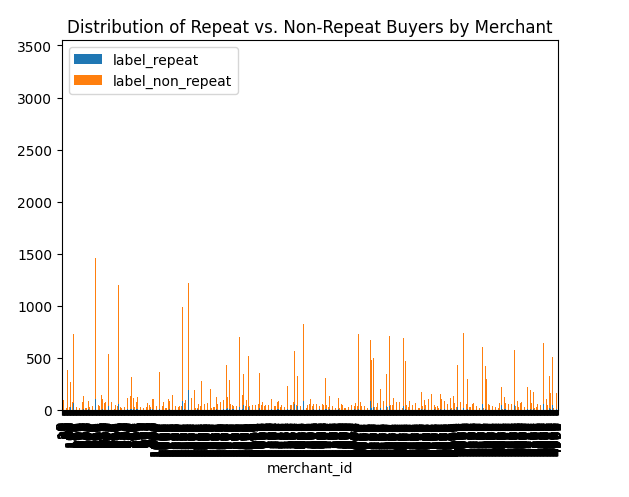
Based on the chart, it can be observed that age ranges from 35-39 and greater than or equal to 50 have the highest repeat buyer rates. This suggests that individuals within these age ranges tend to be more loyal to the product or service being offered and are more likely to make multiple purchases.

The age range of 40-49 also has a relatively high repeat buyer rate, indicating that individuals within this age range are also loyal customers.

On the other hand, the age range below 18 has the lowest repeat buyer rate, indicating that individuals in this age group are less likely to make repeat purchases. This could be due to a variety of factors such as limited disposable income, lack of brand loyalty, or simply being new to the market.

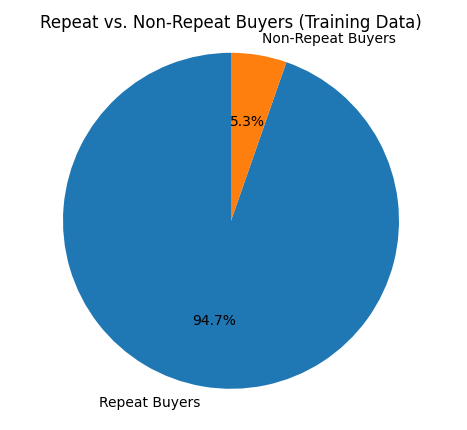
In summary, the given chart shows that repeat buyer rates vary across different age ranges, with older age groups being more likely to make repeat purchases and younger age groups being less likely to do so.

## **3.13 Categorisation of Repeat vs Non Repeat Buyer rates by Merchants**



In the case of the distribution of repeat and non-repeat buyers for a given merchant\_id, a bar chart could be used to show the number or proportion of customers who have made repeat purchases versus those who have made only one purchase from the merchant. The x-axis of the bar chart would represent the merchant\_id, while the y-axis would represent the number or proportion of customers. The blue bars indicate repeat buyers while the orange colored bars indicate non-repeat buyers.

## **3.14 Distribution of Repeat vs Non Repeat Buyers in Training data**



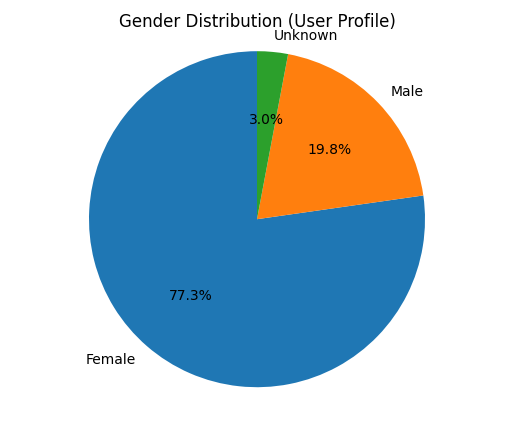
A pie chart is a type of graph that represents data in a circular shape, with different sectors or slices representing different categories of data. The size of each slice is proportional to the percentage of data it represents.

The above pie chart shows the percentage of users who are repeat buyers and non-repeat buyers. Specifically, it indicates that 94.7% of the users are repeat buyers, while only 5.3% of the users are non-repeat buyers.

This means that the vast majority of the users are purchasing products or services from the business multiple times. Only a small fraction of the users are non-returning customers who have made one time purchases.

This information could be important for the business to consider in terms of their marketing and customer retention strategies. If they want to increase the percentage of new buyers they may want to consider new ways to reach potential customers who have not yet made a purchase, they may need to focus on improving the customer experience or offering incentives to encourage customers to return. Alternatively, if they are primarily focused on increasing the percentage of repeat customers then they may want to continue their current marketing strategies.

## **3.15 Distribution of dataset by gender category in Training data**

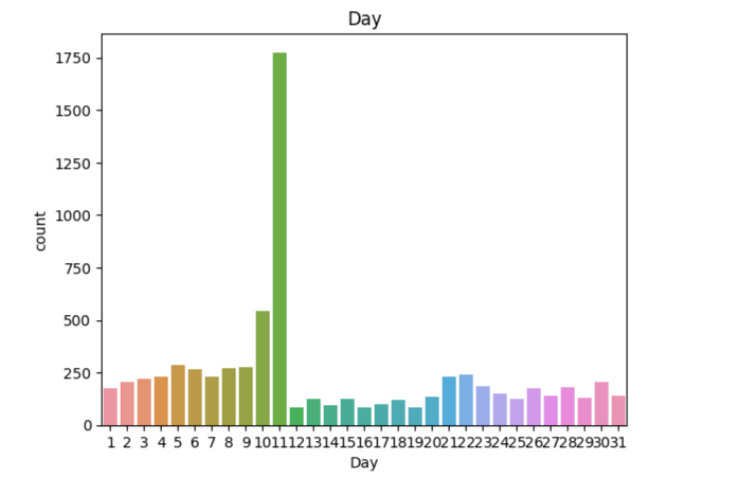


The pie chart represents the gender distribution of buyers, where 77.3% of buyers are female, 19.8% of buyers are male, and 3% of buyers' gender is unknown. This means that out of every 100 buyers, 77 of them are female, 20 of them are male, and the gender of the remaining 3 buyers is unknown.

The high percentage of female buyers may indicate that the product or service being sold is more popular among women or that the marketing efforts have been targeted towards women. On the other hand, the lower percentage of male buyers may suggest that there is room for improvement in targeting and attracting male customers. It is also important to note that the gender of a small percentage of buyers is unknown, which may be due to various reasons such as incomplete data or buyers who prefer not to disclose their gender.

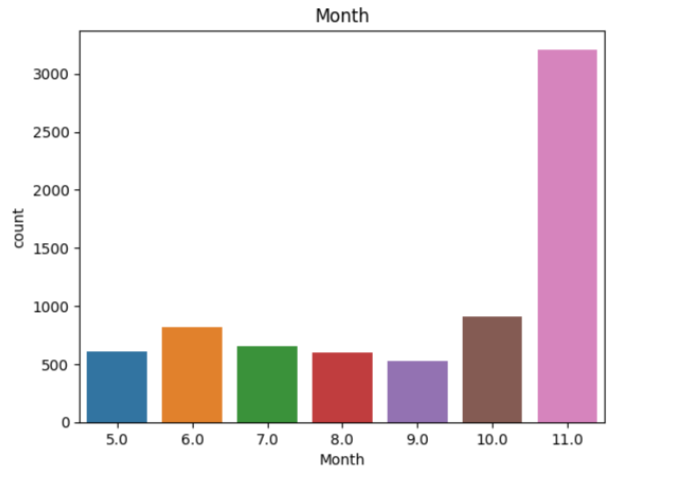
Overall, the pie chart provides valuable information about the gender distribution of buyers, which can be useful for businesses to improve their marketing strategies and better understand their target audience.

## **3.16 Sales on each day of the month in the training dataset**



Based on the information provided, it seems that sales are recorded for every day of the month, and the data shows that the sales on the 11th day are the highest among all the other days.Since it had a “double 11” promotional event going on that day.

## **3.17 Sales on each month of the year in the training dataset**

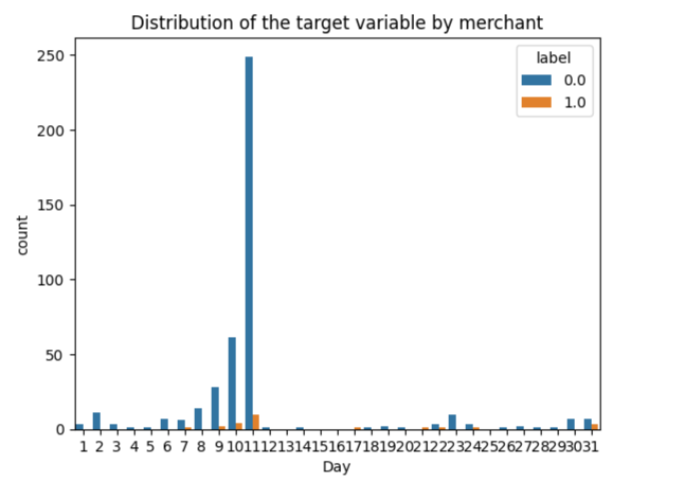


The graph shows the sales figures for each month from May to November, with November having the highest sales. It appears that the sales figures started off relatively low in May and gradually increased over time, with a significant jump in sales from October to November.

This could be due to a variety of factors, such as seasonal trends or promotional activities. It is also possible that the product being sold experienced a surge in demand during this time, leading to increased sales figures.

It is important to note that without additional information, it is difficult to draw any definitive conclusions about the reasons for the increase in sales. However, the graph does clearly show that November had the highest sales figures of any month in the given time period.

## **3.18 Distribution of target variable by merchant along the days in a month**

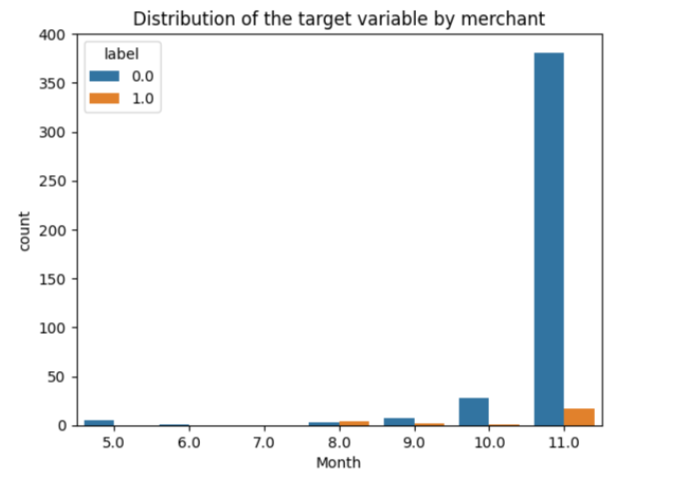


The graph shows the sales for a particular label over the course of a month, with each day being represented on the horizontal axis and the sales volume being represented on the vertical axis. The graph is divided into two segments, one representing first-time buyers in blue and the other representing repetitive buyers in orange.

The highest point on the graph is on the 11th day of the month, which represents the highest number of first-time and repetitive buyers for this particular label. This indicates that the 11th day was the most successful day for sales, with a large number of new customers making purchases, as well as a significant number of returning customers making additional purchases.

Overall, the graph suggests that the label experienced relatively steady sales throughout the month, with a slight uptick in sales on the 11th day. This could be due to a variety of factors, such as a promotional event on this day of “double 11”.

## **3.19 Distribution of target variable by merchant along the months**



To elaborate on the sales for the particular label (merchant\_id) in the months from May to November, with November recording the highest number of buyers, we would need more information such as the type of product or service being sold, the geographical location of the merchant, and the marketing strategies employed during this period.

Assuming that there are no major external factors affecting sales such as economic recessions, supply chain disruptions, or natural disasters, there are a few possible explanations for the increase in sales in November compared to the other months:

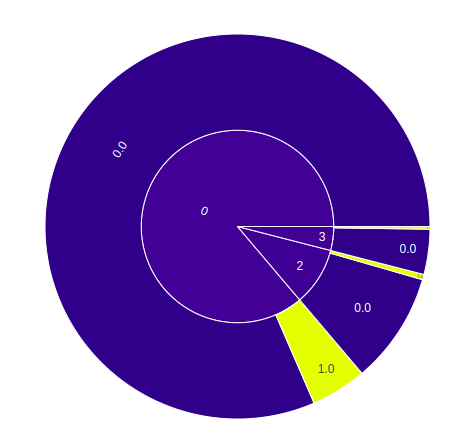
1. Seasonal factors: Depending on the product or service being sold, there may be certain seasons when demand is higher. For example, November is the month leading up to the holiday season, which typically sees increased spending on gifts, decorations, and travel.

2. Marketing promotions: The merchant may have run promotional campaigns or offered discounts during November to attract more buyers. This could have encouraged existing customers to make repeat purchases and attracted new customers who were looking for deals.

3. Inventory management: The merchant may have stocked up on popular products or services in anticipation of the holiday season, which resulted in higher sales in November.

It is also possible that there were other factors at play that contributed to the increase in sales in November. However, without more specific information, it is difficult to provide a more detailed explanation.

## **3.20 Distribution of target variable by action type**



0 = Click

1 = add to cart

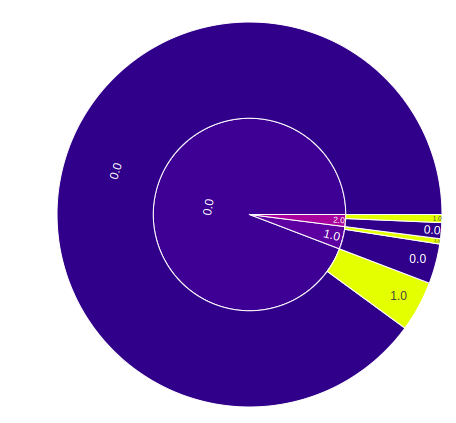
2 = purchase

3 = add to favorite

The categories mentioned refer to classifying user actions or behaviors on a website or application. These categories include click, add-to-cart, purchase, and add-to-favorite. Based on the given information, the majority of users are just browsing or viewing items on the website or application without taking further action, while a relatively small number of users are adding items to their shopping carts. The categories of purchase and add-to-favorite have similar levels of user buying characteristics, with most users being one-time buyers. It is crucial to analyze and comprehend user behavior to enhance the user experience and drive conversions. Identifying areas for improvement and customizing the website or application to the needs of users can increase engagement and achieve business goals.

It appears that the click category has the highest number of users, which suggests that many people are interested in exploring the product or website but may not necessarily be ready to make a purchase. This could be due to a variety of reasons, such as needing more information about the product, comparing prices with other sites, or simply browsing without a clear intention to buy.

## **3.21 Distribution of target variable by gender**



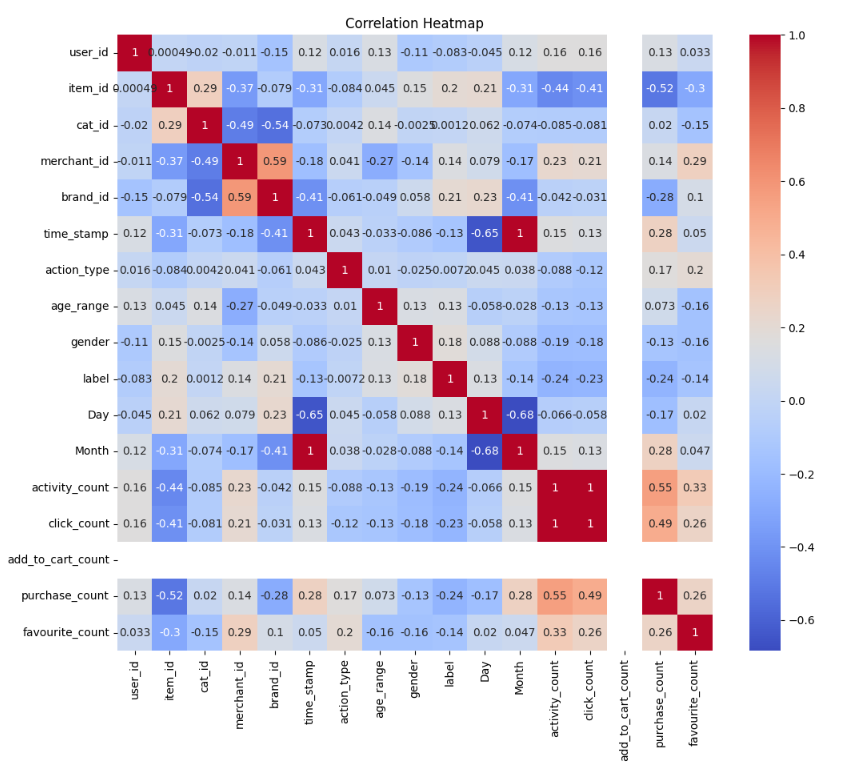
0 = Female

1 = male

2 = unknown

According to the statement, female customers have a higher likelihood of being both repeated buyers and one-time buyers compared to other customer groups. This could be due to their brand loyalty and emotional attachment to products, or their willingness to explore different brands and products. The unknown customer category shows similar proportions of one-time and repeated buyers, which may be due to the diversity of customers in this group. The statement emphasizes the need for businesses to understand customer demographics and behaviors to develop effective marketing strategies that cater to different segments of the market. By identifying which groups are more likely to be repeated buyers or one-time buyers, businesses can tailor their messaging, promotions, and product offerings to better meet the needs and preferences of their target customers.

## **3.22 Correlation before Feature Engineering**



# **Feature Engineering**

## **4.1 Reading the data**

Firstly, we are reading three CSV files into memory and create pandas DataFrames that can be used for data analysis and manipulation.

* user\_info\_format1.csv is read into the DataFrame **user\_info**
* user\_log\_format1.csv is read into the DataFrame **user\_log**
* train\_format1.csv is read into the DataFrame **train\_format**

1.user\_info has three columns. Which are:

Index(['user\_id', 'age\_range', 'gender'], dtype='object')

2.user\_log has seven columns. Which are:

Index(['user\_id', 'item\_id', 'cat\_id', 'seller\_id', 'brand\_id', 'time\_stamp', 'action\_type'], dtype='object')

Here will only keep the rows where the value in the 'item\_id' column is greater than or equal to 641 and less than or equal to 800.

The resulting DataFrame will only include rows where the 'item\_id' column has a value between 641 and 800, inclusive.

3.train\_format has three columns. Which are:

Index(['user\_id', 'merchant\_id', 'label'], dtype='object')

## **4.2 Merging the Data**

We are merging two pandas DataFrames, **user\_log** and **user\_info**, using the 'user\_id' column as the common key. It creates a new DataFrame called **data**. The **how='left'** parameter specifies to use a left outer join, meaning that all rows from the **user\_log** DataFrame are included in the merged DataFrame, but only matching rows from the **user\_info** DataFrame are included. Any non-matching rows from **user\_info** are filled with null values.

From the data which comes from this inner join, we are using it DataFrame called **data** that comes from the above outer Join. Now we are using, **pd.merge,** that uses an inner join to combine the DataFrames, meaning that only the rows with matching values in both DataFrames are included in the resulting DataFrame. if there are any missing values (NaN) in the columns used for the merge keys, those rows will be excluded from the merged DataFrame.

## **4.3 Timestamp to Day and Month:**

Firstly, We are extracting the day and month information from the 'time\_stamp' column in the **data** DataFrame. Like, **data['time\_stamp'] % 100** extracts the last two digits of the 'time\_stamp' value, which represent the day of the month. The expression **(data['time\_stamp'] - data['Day']) / 100** subtracts the day of the month from the 'time\_stamp' value and then divides by 100 to extract the month.

Now we can drop the column 'time\_stamp' from the **data** DataFrame using the **drop()** method with the **columns** parameter. The resulting DataFrame now includes new 'Day' and 'Month' columns that contain the day and month information extracted from the original 'time\_stamp' column.

## **4.4 Activity Count of User**

Now we are counting the number of activity that user has done. We first groups the **data** DataFrame by the 'user\_id' column and counts the number of occurrences of each unique value in the 'action\_type' column using the **count()** method. The resulting DataFrame is assigned to a variable **activity\_count**. The **reset\_index()** method is then called to reset the index of the DataFrame to a simple range index.

The next line of code renames the columns of **activity\_count** to 'user\_id' and 'activity\_count'.

Finally, the **pd.merge()** method is used to merge the **data** DataFrame with the **activity\_count** DataFrame, using the 'user\_id' column as the common key. This adds a new column to the **data** DataFrame called 'activity\_count', which represents the total number of actions performed by each user. The **how='left'** parameter specifies to use a left outer join, meaning that all rows from the **data** DataFrame are included in the merged DataFrame, but only matching rows from the **activity\_count** DataFrame are included. Any non-matching rows from **activity\_count** are filled with null values.

Based on the given information, These categories are:

· Category 0 (click): When a user clicks on a link or button, but does not perform any further action.

· Category 1 (add-to-cart): When a user adds a product to their cart.

· Category 2 (purchase): When a user completes a purchase.

· Category 3 (add-to-favorite): When a user adds a product to their favorites list.

### **4.4.1 User Click Count:**

For user click count, we only keep the rows where the 'action\_type' column has a value of 0 (indicating a click action). The resulting DataFrame is then grouped by the 'user\_id' column and the number of occurrences of each unique value in the 'action\_type' column is counted using the 'count()` method. The resulting DataFrame is assigned to `click\_count`. The `reset\_index()` method is then called to reset the index of the DataFrame to a simple range index.

### **4.4.2 Add to Cart Count:**

For Add To Click count, we only keep the rows where the 'action\_type' column has a value of 1 (indicating an add-to-cart action). The resulting DataFrame is then grouped by the 'user\_id' column and the number of occurrences of each unique value in the 'action\_type' column is counted using the **count()** method. The resulting DataFrame is assigned to **add\_to\_cart\_count**.

### **4.4.3 Purchase Count:**

To Find out Purchase count, we only keep the rows where the 'action\_type' column has a value of 2 (indicating a purchase action). The resulting DataFrame is then grouped by the 'user\_id' column and the number of occurrences of each unique value in the 'action\_type' column is counted using the **count()** method. The resulting DataFrame is assigned to **purchase\_count**.

### **4.4.4 Add to Favourite Count:**

To Find out the user who has selected add to favourite for the items, we only keep the rows where the 'action\_type' column has a value of 3 (indicating a favourite action). The resulting DataFrame is then grouped by the 'user\_id' column and the number of occurrences of each unique value in the 'action\_type' column is counted using the **count()** method. The resulting DataFrame is assigned to **favourite\_count**.

Lastly, We are filling any missing (null) values in the columns 'click\_count', 'activity\_count', 'add\_to\_cart\_count', 'purchase\_count', and 'favourite\_count' of the **data** DataFrame with zeros using the **fillna()** method. This ensures that any user who did not perform a particular action (e.g., did not add any items to their cart) is still represented in the DataFrame with a count of zero for that action, rather than being excluded from the DataFrame entirely.

## **4.5 Gender Feature Selection:**

Firstly, We will fill any missing (null) values in the 'gender' column of the **data** DataFrame with a value of 2 using the **fillna()** method. The value 2 likely represents an "unknown" or "unspecified" gender category, which is a common approach to handling missing data in gender or sex fields.

This code creates three new binary columns in the **data** DataFrame based on the values in the original 'gender' column. The **label\_binarize()** function is used to convert the 'gender' column into a one-hot encoded binary representation where '0' represents 'female', '1' represents 'male', and '2' represents 'unknown'. The resulting binary values are stored in the **user\_gender** DataFrame.

The code then creates three new columns in the **data** DataFrame ('user\_gender\_female', 'user\_gender\_male', and 'user\_gender\_unknown') and assigns the binary values from the **user\_gender** DataFrame to these new columns. Finally, the original 'gender' column is dropped from the **data** DataFrame using the **drop()** method, and the resulting DataFrame is displayed with the **head()** method.

We can see the result as below:

| **index** | **user\_id** | **item\_id** | **cat\_id** | **merchant\_id** | **brand\_id** | **action\_type** | **age\_range** | **label** | **Day** | **Month** | **activity\_count** | **click\_count** | **add\_to\_cart\_count** | **purchase\_count** | **favourite\_count** | **user\_gender\_female** | **user\_gender\_male** | **user\_gender\_unknown** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 310303 | 650 | 656 | 3473 | 3305.0 | 0 | 4.0 | 0 | 11 | 11.0 | 4 | 3.0 | 0.0 | 1.0 | 0.0 | 1 | 0 | 0 |
| **1** | 310303 | 650 | 656 | 3473 | 3305.0 | 0 | 4.0 | 0 | 11 | 11.0 | 4 | 3.0 | 0.0 | 1.0 | 0.0 | 1 | 0 | 0 |
| **2** | 310303 | 650 | 656 | 3473 | 3305.0 | 2 | 4.0 | 0 | 11 | 11.0 | 4 | 3.0 | 0.0 | 1.0 | 0.0 | 1 | 0 | 0 |
| **3** | 310303 | 650 | 656 | 3473 | 3305.0 | 0 | 4.0 | 0 | 11 | 11.0 | 4 | 3.0 | 0.0 | 1.0 | 0.0 | 1 | 0 | 0 |
| **4** | 156939 | 653 | 35 | 4267 | 6046.0 | 0 | 4.0 | 1 | 11 | 11.0 | 2 | 1.0 | 0.0 | 1.0 | 0.0 | 0 | 1 | 0 |

This code block appears to be a duplicate of the previous block you executed, which created additional columns based on the user's activity on the platform. It uses the same **user\_id** column from the **data** dataframe, extracts the **action\_type** column, binarizes the values using **label\_binarize** from scikit-learn, and then creates separate columns for each of the four activity types (**Click**, **add\_to\_cart**, **purchase**, **add\_to\_favorite**) using **pd.DataFrame**. Finally, it merges the new columns back into the **data** dataframe using **pd.merge** on the **user\_id** column.

## **4.6 Age Feature Selection:**

We are creating one-hot encoded DataFrame for the 'age\_range' column in the 'data' DataFrame. It first replaces the value 8 in 'age\_range' with 7, and then creates the one-hot encoded DataFrame using the 'get\_dummies' function from pandas. The resulting one-hot encoded DataFrame has columns for age ranges 0-18, 18-24, 25-29, 30-34, 35-39, 40-49. The columns are named according to the 'column\_name' dictionary, which maps the numerical age range values to the corresponding column names.

Then we are creating dummy variables for the **age\_range** column using **pd.get\_dummies** method, and renames the columns based on a dictionary. Then, it loops over the column names and adds them to the **data** dataframe using **data[val] = age[val]**. Finally, it drops the original **age\_range** column from the **data** dataframe.

## **4.7 Seller Level Information:**

### **4.7.1 Merchant to item:**

We are extracting information about the number of unique items sold by each merchant in the dataset.

First, it selects two columns from the **data** DataFrame, namely **merchant\_id** and **item\_id**, and creates a new DataFrame named **merchant\_item** that only contains these two columns.

Next, it drops the duplicate rows based on the **item\_id** column, so that each item is counted only once per merchant.

Then, it groups the resulting DataFrame by the **merchant\_id** column and counts the number of unique items for each merchant using the **agg()** function with the **count()** method as an argument. The result is stored in a new DataFrame named **merchant\_item\_info**, where each row represents a merchant and the number of unique items they have sold.

Finally, it renames the **item\_id** column to **merchant\_item\_count** for clarity, and merges this information back into the original **data** DataFrame based on the **merchant\_id** column using a left join.

### **4.7.2 Merchant to Brand:**

Here we will extract some features related to the merchant of each item in the dataset.

First, we need to extract the number of items each merchant has using the **merchant\_item** dataframe. The **drop\_duplicates()** method is used to ensure that each item is only counted once per merchant. The resulting dataframe, **merchant\_item\_info**, has two columns: **merchant\_id** and **merchant\_item\_count**.

Secondly, it extracts the number of brands each merchant has using the **merchant\_brand** dataframe. Again, **drop\_duplicates()** is used to ensure that each brand is only counted once per merchant. The resulting dataframe, **merchant\_brand\_info**, has two columns: **merchant\_id** and **merchant\_brand\_count**.

Finally, the two resulting dataframes are merged with the original **data** dataframe on the **merchant\_id** column using the **pd.merge()** method. This adds two new columns to the **data** dataframe: **merchant\_item\_count** and **merchant\_brand\_count**, which represent the number of unique items and brands each merchant has, respectively.

### **4.7.3 Merchant to Category:**

Here we are calculating the number of unique categories (cat\_id) each merchant (merchant\_id) is associated with, and adding this information to the "data" dataframe by merging it with the "merchant\_cat\_info" dataframe based on the "merchant\_id" column.

First, a new dataframe "merchant\_cat" is created that only includes the "merchant\_id" and "cat\_id" columns from the "data" dataframe. Then, the "merchant\_cat\_info" dataframe is created by grouping "merchant\_cat" by "merchant\_id" and aggregating the count of unique "cat\_id" associated with each merchant.

Finally, the "merchant\_cat\_count" column is added to the "data" dataframe by merging it with the "merchant\_cat\_info" dataframe based on the "merchant\_id" column.

## **4.8 User Level Information:**

### **4.8.1 User to Merchant:**

In User seller relationship we are extracting unique pairs of **user\_id** and **merchant\_id** from the **data** dataframe, counts the number of unique **merchant\_id** per **user\_id**, and creates a new dataframe with **user\_id** and **user\_merchant\_count**. It then merges the new dataframe with **data** based on the **user\_id** column, adding the **user\_merchant\_count** column to the **data** dataframe.

### **4.8.2 User to Category:**

To create User and category relationship we are creating a dataframe **user\_cat** containing unique pairs of user and category IDs in the **data** dataframe. Then, it groups the **user\_cat** dataframe by the **user\_id** column and counts the number of unique categories associated with each user. The resulting dataframe **user\_cat\_info** has two columns: **user\_id** and **user\_cat\_count**, which represents the number of unique categories associated with each user. Finally, the **user\_cat\_info** dataframe is merged with the **data** dataframe using a left join on the **user\_id** column.

### **4.8.3 User to Brand:**

Now we are extracting the user and brand information from the main data and then creates a new data frame (**user\_brand**) containing only the **user\_id** and **brand\_id** columns. It then drops duplicate **brand\_id** values, meaning each **brand\_id** only appears once in the new data frame. It then groups the data by **user\_id** and counts the number of unique **brand\_id** values for each user. The resulting data frame (**user\_brand\_info**) has two columns: **user\_id** and **user\_brand\_count**, which shows the number of unique brands each user has interacted with. Finally, the **user\_brand\_info** data frame is merged with the main data frame (**data**) based on the **user\_id** column, so that the **user\_brand\_count** column is added to the main data frame.

## **4.9 User Seller Information:**

### **4.9.1 User Seller Brand Information:**

For User Merchant And Brand relationship we calculate the number of times each user has purchased from each merchant and each brand from the given transaction data.It first selects the relevant columns (user ID, merchant ID, and brand ID) from the **data** dataframe, then groups the data by user ID and merchant ID, and counts the number of unique brand IDs for each group. This count is then stored in a new column called **user\_merchant\_brand\_count** in the **user\_brand\_info** dataframe. Finally, the **user\_brand\_info** dataframe is merged back into the **data** dataframe on the columns **user\_id** and **merchant\_id** using a left join. This allows the **user\_merchant\_brand\_count** information to be added to the original **data** dataframe for each unique user and merchant combination.

### **4.9.2 User Seller Category Information:**

We are now calculating a distinct product categories that each user has purchased from each merchant and adds this information to the **data** DataFrame.

First, the code selects the columns **user\_id**, **merchant\_id**, and **cat\_id** from the **data** DataFrame and creates a new DataFrame called **user\_merchant\_cat**. Then, the **groupby** method is used to group the data by **user\_id** and **merchant\_id** and count the number of distinct **cat\_id** values for each group. This information is stored in a new DataFrame called **user\_merchant\_cat\_info**.

Next, the column name of the count is changed to **user\_merchant\_cat\_count**. Finally, the **user\_merchant\_cat\_info** DataFrame is merged with the **data** DataFrame based on the **user\_id** and **merchant\_id** columns, and the resulting merged DataFrame is stored in the **data** variable.

## **4.10 User Merchant Day Category Information:**

Lastly, we are creating a new dataframe **user\_merchant\_day\_cat\_info** by selecting four columns from the **data** dataframe, namely **user\_id**, **merchant\_id**, **Day**, and **cat\_id**. Then, it is grouping the new dataframe by these four columns and aggregating the **cat\_id** column by count. The resulting dataframe is then renamed to **user\_merchant\_day\_cat\_count**. Finally, the **data** dataframe is merged with the **user\_merchant\_day\_cat\_info** dataframe on the columns **user\_id**, **merchant\_id** using a left join. The printed output shows the first 10 rows of the **user\_merchant\_day\_cat\_info** dataframe.

## **4.11 Resulting Features:**

**‘data.columns’** will show the final columns which we generated after extracting useful features.

Index(['user\_id', 'item\_id', 'cat\_id', 'merchant\_id', 'brand\_id',

'action\_type', 'label', 'Day\_x', 'Month', 'activity\_count',

'click\_count', 'add\_to\_cart\_count', 'purchase\_count', 'favourite\_count',

'user\_gender\_female', 'user\_gender\_male', 'user\_gender\_unknown',

'user\_Click', 'user\_Add\_to\_cart', 'user\_purchase',

'user\_Add\_to\_favorite', 'age\_0\_18', 'age\_18\_24', 'age\_25\_29',

'age\_30\_34', 'age\_35\_39', 'age\_40\_49', 'merchant\_item\_count',

'merchant\_brand\_count', 'merchant\_cat\_count', 'user\_merchant\_count',

'user\_cat\_count', 'user\_brand\_count', 'user\_merchant\_brand\_count',

'user\_merchant\_cat\_count', 'Day\_y', 'user\_merchant\_day\_cat\_count'],

dtype='object')

# **Statistical Analysis and Feature Ranking**

Statistical summary is a method of summarizing and describing the numerical features of a dataset. It involves calculating and reporting various statistical measures that describe the central tendency, dispersion, and shape of the dataset.

To generate a statistical summary, we have typically focused on the numerical features of the dataset, which are the columns that contain quantitative data such as integers, floats or numeric values. We have used Python libraries such as Pandas or NumPy to calculate the following statistical measures for each of the numerical features:

1. Count: The number of non-missing values in each feature.

2. Mean: The average value of each feature.

3. Standard deviation: A measure of the spread of each feature around the mean.

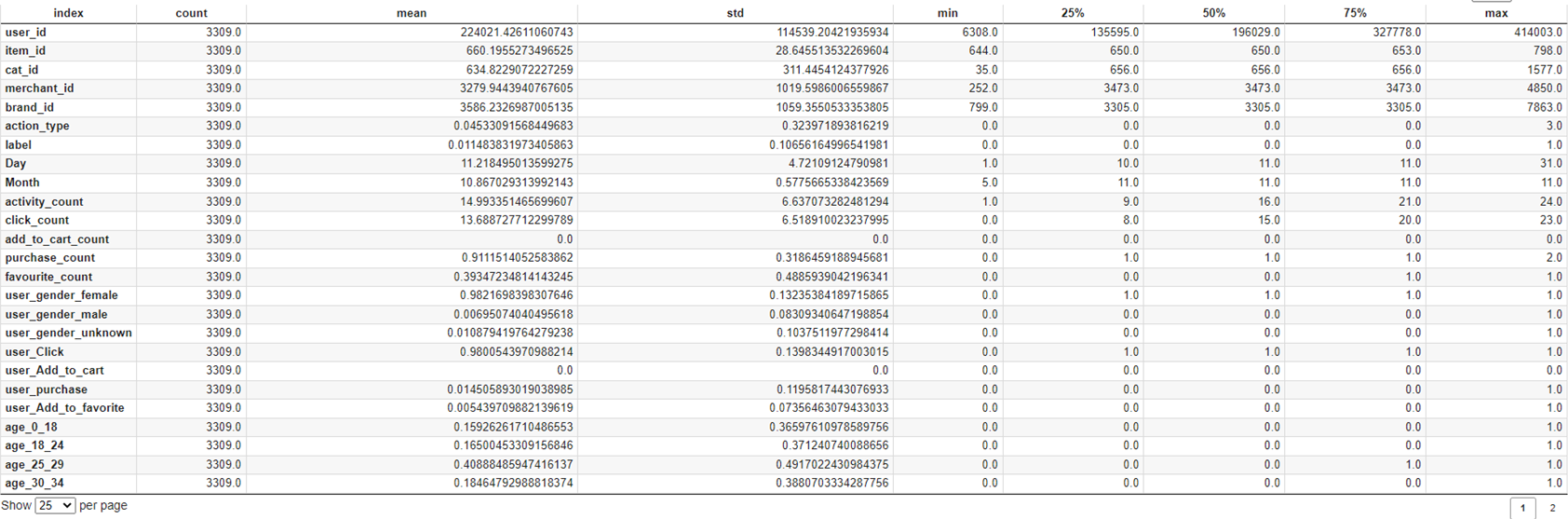
4. Minimum: The smallest value in each feature.

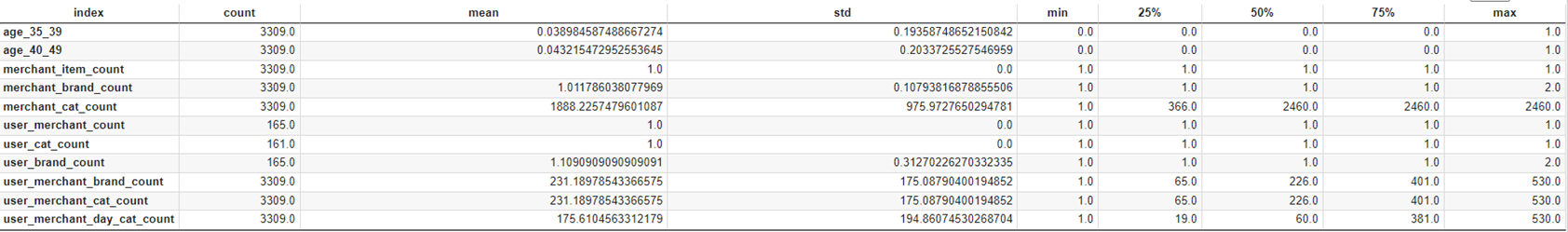
5. Maximum: The largest value in each feature.

6. 25th percentile: The value below which 25% of the data falls in each feature.

7. 50th percentile: The value below which 50% of the data falls in each feature, also known as the median.

8. 75th percentile: The value below which 75% of the data falls in each feature.





Feature ranking is a technique used in machine learning and data analysis to determine the importance or relevance of different features or variables that are used in a model or analysis. It involves assessing the impact or contribution of each feature to the overall performance or outcome of the model, and then ranking them in order of importance.

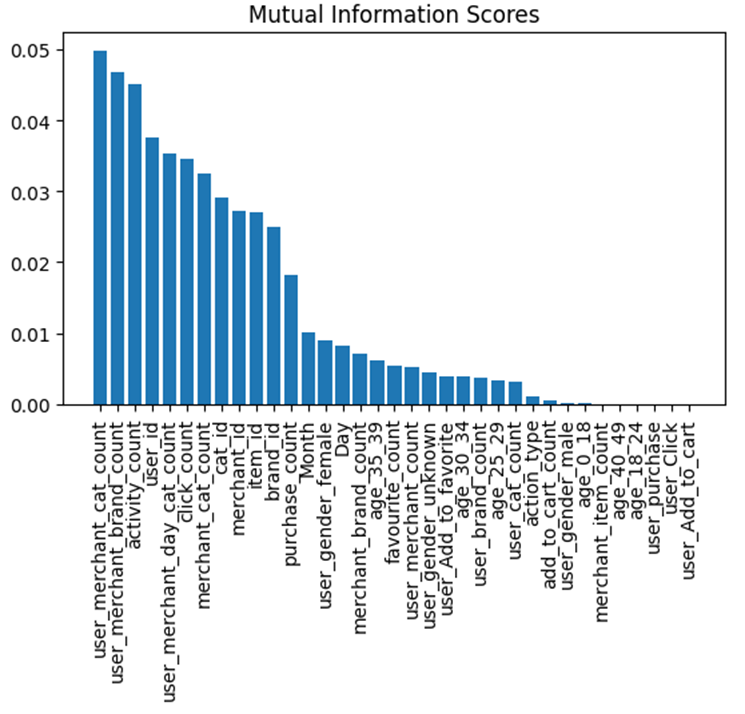
# Feature ranking is useful because it helps to identify the most important features that have the greatest impact on the outcome or performance of a model. This information can then be used to improve the model by focusing on the most important features and ignoring or reducing the importance of less important features. Feature ranking can also help to identify redundant or irrelevant features, which can be removed to simplify the model and improve its performance. Additionally, feature ranking can be used to gain insight into the underlying relationships between features and outcomes, which can help to guide further analysis or modeling efforts.

### **5.1 Feature ranking with Mutual Information Classifier**

Mutual Information (MI) is a measure of the statistical dependence between two variables. In the context of feature selection for classification tasks, Mutual Information Classification (MIC) is used to quantify the amount of information that a feature provides about the target variable. In other words, it measures how much knowing the value of a particular feature helps us to predict the class labels accurately.

MIC can be used for feature ranking by computing the mutual information between each feature and the target variable and then sorting the features based on their MI scores in descending order. The higher the MI score, the more relevant the feature is to the classification task. Therefore, the top-ranked features can be selected as the most informative ones and used for building a classification model.

In contrast to traditional statistical methods that assume linear relationships between variables, MIC can capture complex and nonlinear dependencies between features and the target variable. Additionally, MIC is robust to irrelevant features, making it a useful tool for high-dimensional feature selection problems.

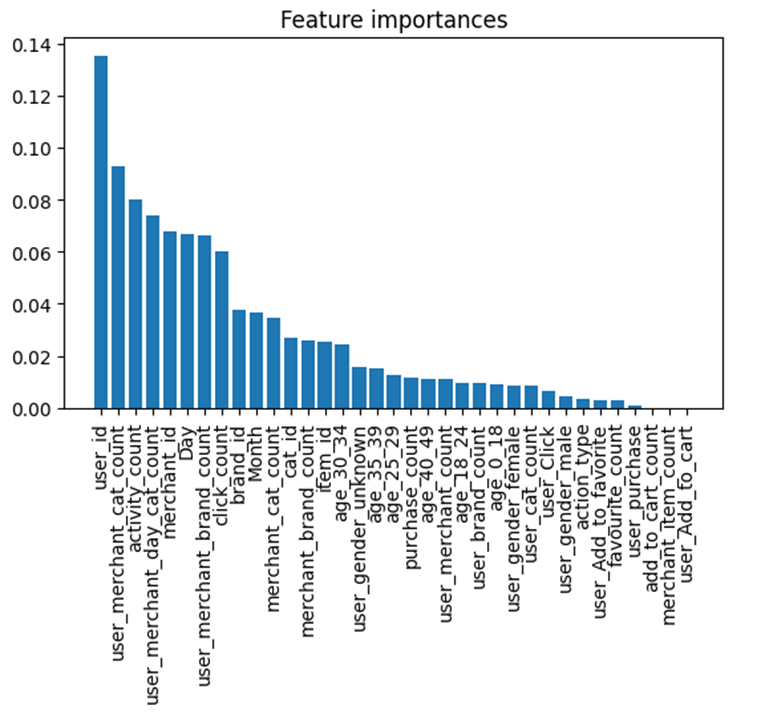


### **5.2 Feature ranking with Random Forest Classifier**

Random forest classifier is a popular ensemble learning algorithm used in machine learning for classification tasks. It is a combination of decision trees where each tree is built using a random subset of the input features and a random subset of the training data. The predictions of individual trees are combined to make the final prediction.

One of the benefits of random forest classifier is that it can be used to determine the feature importance in the dataset. This can be done by calculating the Gini importance or mean decrease impurity for each feature. The feature with the highest importance score is considered to be the most important in predicting the target variable.

To perform feature ranking using random forest classifier, the algorithm is first trained on the dataset using all the features. The feature importance scores are then calculated, and the features are ranked based on their scores. The least important features can be removed from the dataset, and the algorithm can be retrained using only the most important features. This process can be repeated until the desired level of accuracy is achieved.

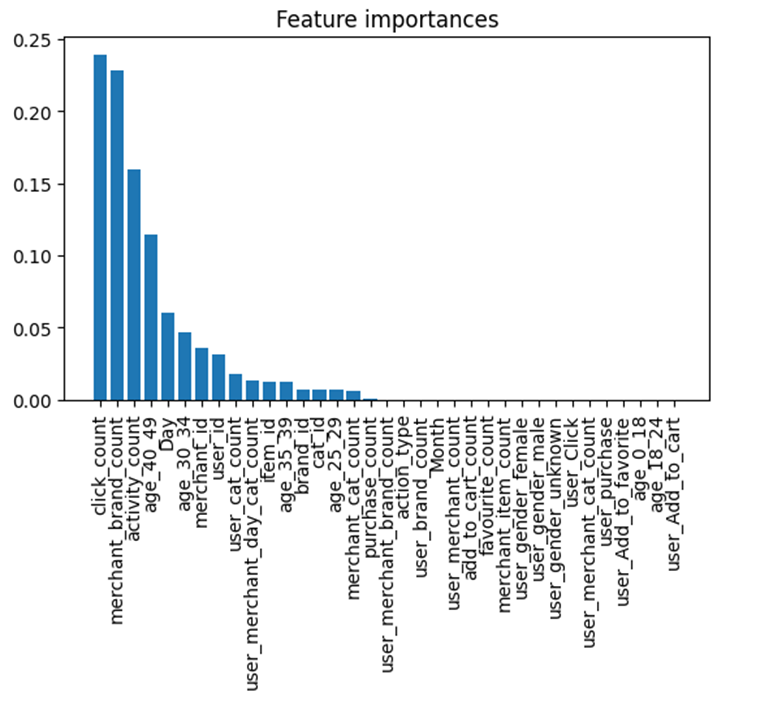


### **5.3 Feature ranking with XGB Classifier**

XGBoost (Extreme Gradient Boosting) is a popular machine learning library that implements gradient boosting algorithms. The XGBClassifier is a classification model provided by XGBoost that uses a tree-based boosting algorithm to classify data. The algorithm builds a series of decision trees iteratively, where each new tree is trained on the errors made by the previous trees.

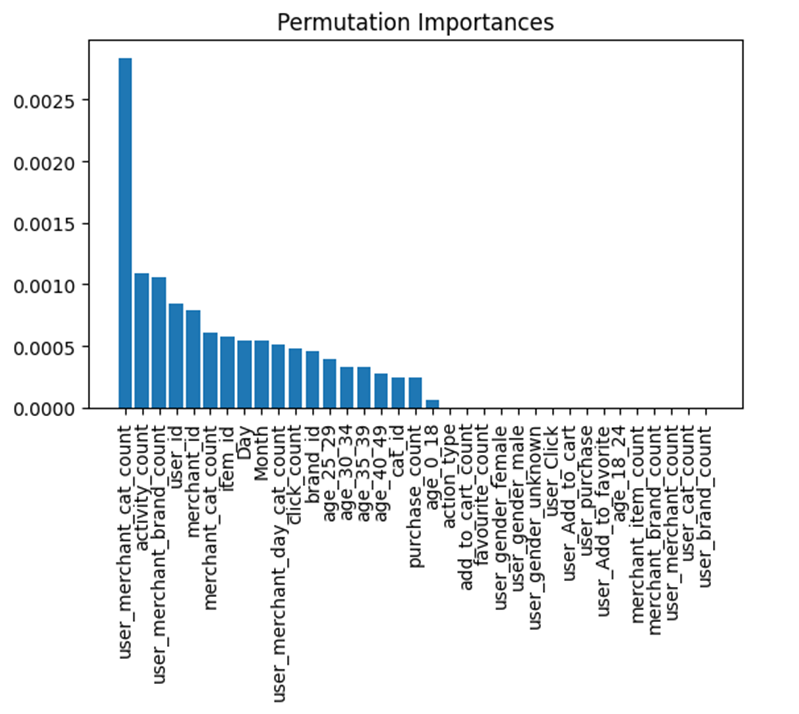
In addition to its powerful classification abilities, XGBClassifier can also be used for feature ranking. Feature ranking is the process of determining the importance of each feature in a dataset for making predictions. XGBClassifier provides a feature\_importances\_ attribute which can be used to calculate the importance of each feature in the dataset.

The feature\_importances\_ attribute assigns a score to each feature based on how useful it is in predicting the target variable. Features with higher scores are considered more important. This information can be used to select the most relevant features for a model, reducing the dimensionality of the dataset and potentially improving the accuracy and efficiency of the model.



### **5.4 Feature ranking with Permutation Feature Importance Classifier**

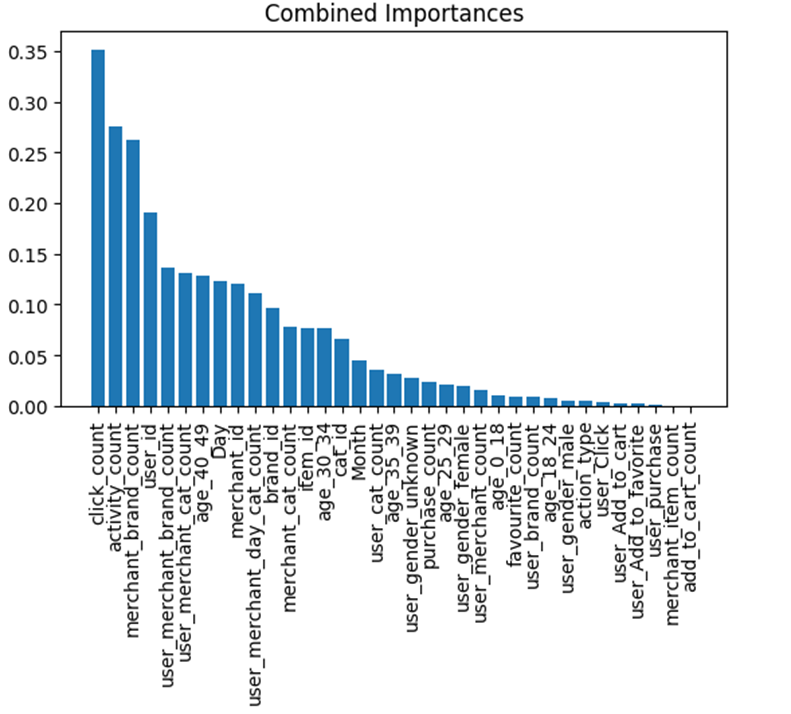
Permutation Feature Importance is a technique used for feature ranking in machine learning. It works by randomly permuting the values of a single feature in the test data and measuring the resulting decrease in the model's accuracy or performance metric. If permuting a feature results in a significant drop in the performance metric, it suggests that the feature is important for the model's predictive power. Conversely, if permuting a feature has little effect on the performance metric, it suggests that the feature is not important. This process is repeated for all features, and the features are ranked according to their impact on the performance metric. Thus, Permutation Feature Importance is a simple and effective way to identify the most important features in a machine learning model.



### **5.5 Feature ranking with Combined Feature Importance**

In this technique, the feature importance values obtained from each of the four techniques are added up, and only those features with a threshold value greater than 0.1 are selected for further analysis. This threshold value can be adjusted based on the specific requirements of the model.

By combining the results of multiple feature importance techniques, Combined Feature Importance helps to identify the most robust and relevant features for a given model, while also reducing the risk of overfitting. This approach can lead to more accurate and efficient models, with improved performance and generalization.



After doing the Combined Feature Importance we are getting 10 features which are:

['click\_count', 'activity\_count', 'merchant\_brand\_count', 'user\_id', 'user\_merchant\_brand\_count', 'age\_40\_49', 'Day', 'merchant\_id', 'user\_merchant\_day\_cat\_count', 'user\_merchant\_cat\_count']

### **5.6 PCA**

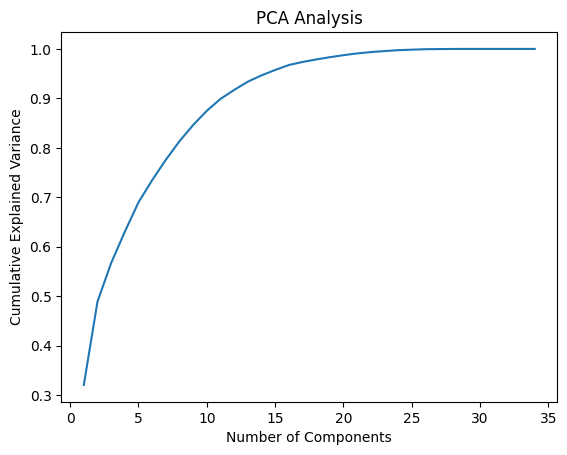
PCA (Principal Component Analysis) is a technique for reducing the dimensionality of data by finding the principal components that capture the maximum variance in the data. It is widely used in data analysis, machine learning, and pattern recognition.

Firstly,In PCA analysis we select the columns that will be used for the analysis. In this case, the 'user\_id' and 'label' columns are not considered for the PCA analysis, and hence they are removed from the dataset.

After selecting the relevant columns, the missing values in the dataset are imputed using the mean of each column. This is done to ensure that the PCA analysis can be performed on complete data. Next, the data is standardized using StandardScaler. Standardization involves scaling the data so that the mean is zero and the variance is one, which helps in comparing the importance of different features in the PCA analysis.

Finally, PCA is applied to the standardized data. The PCA algorithm is used to identify the most important features in the dataset by projecting the data onto a lower-dimensional space. The PCA algorithm calculates the eigenvalues and eigenvectors of the dataset and uses them to transform the original features into a new set of orthogonal features called principal components. These principal components are sorted in descending order of their explained variance, and a plot is generated to show how much variance is explained by each principal component.

Based on the plot generated, the optimal number of components is selected i.e., 15 , which in this case is determined as the number of components needed to explain 95% of the total variance in the data. Finally, the optimal number of components we found are used for further analysis or modeling.



# **6.Predicted Model:**

Firstly, We are reading the file and taking features from the previous work where we extracted main features and will going to use that in our model to train.

Then we are dividing data into given and predicition datasets.

## **6.1 Splitting into Train and Test Datasets:**

SimpleImputer is a class in the sklearn library that provides basic strategies for imputing missing values. In this case, the strategy used is 'mean', which replaces missing values with the mean of the available data.

After the missing values have been imputed, the features are then scaled using StandardScaler. This is done using the StandardScaler class in the sklearn library, which scales the features by removing the mean and scaling to unit variance.

The output of the imputer is passed as input to the scaler, and the transformed features are stored in the X\_train\_scaled variable. This variable contains the scaled feature values that can be used for training a machine learning model.

We are spliting the given dataset into training and testing sets using the **train\_test\_split** function from the scikit-learn library. The dataset is represented as two variables, **X** and **y**, where **X** contains the feature data and **y** contains the target variable.

The **test\_size** parameter specifies the fraction of the dataset that will be used for testing. In this case, the testing dataset will be 20% of the total dataset, and the remaining 80% will be used for training.

The **random\_state** parameter is used to ensure that the same random values are generated each time the code is run. This ensures that the split of the dataset is consistent across multiple runs of the program.

The **stratify** parameter is used to ensure that the target variable is evenly distributed across the training and testing datasets. This is important when dealing with imbalanced datasets where one class is much more frequent than the others. By setting **stratify=y**, the function will ensure that the same proportion of each class is present in both the training and testing datasets.

The resulting variables **X\_train**, **X\_test**, **y\_train**, and **y\_test** represent the training and testing sets of featurres and target variables, respectively. These can be used to train and evaluate machine learning models on the dataset.

## **6.2 PCA:**

Here, the PCA is performed using the PCA class from the sklearn library, without specifying the number of components.

The function then plots the cumulative explained variance ratio for each component, which shows the proportion of the total variance explained by each component. The number of components needed to explain a certain percentage of the total variance can be chosen based on this plot.

In this case, the function chooses the number of components that explain 95% of the variance by finding the index of the first cumulative explained variance ratio that is greater than or equal to 0.95, adding 1 to get the number of components, and storing it in the variable optimal\_components.

The function then prints the number of optimal components and can be used as a preprocessing step before training a machine learning model.

## **6.3 Calulate various scores:**

Now we will calculate and display different evaluation metrics for a given model. The function takes in several arguments:

* **accuracyScore**: a float representing the accuracy score of the model
* **precisionScore**: a float representing the precision score of the model
* **recallScore**: a float representing the recall score of the model
* **f1Score**: a float representing the F1 score of the model
* **confusion**: a confusion matrix object representing the confusion matrix of the model
* **roc**: a tuple representing the fpr, tpr and thresholds of the ROC curve of the model
* **roc\_auc**: a float representing the area under the ROC curve of the model
* **model\_name**: a string representing the name of the model
* **displayPlotsOutput**: a boolean value indicating whether or not to display the plots output
* **displayTableOutput**: a boolean value indicating whether or not to display the table output

The function first creates a dictionary with the different scores, using the provided arguments, and then creates a pandas dataframe from that dictionary. The dataframe is then displayed using the **to\_markdown()** method.

If **displayPlotsOutput** is True, the function also displays two plots: a ROC curve and a confusion matrix. The ROC curve is displayed using the **RocCurveDisplay** function from **sklearn.metrics**, while the confusion matrix is displayed using the **ConfusionMatrixDisplay** function, also from **sklearn.metrics**.

## **6.4 Fit dataset into model:**

We are fitting the values of X and Y into our model. So for that we created one function called **model\_fit(),** which will take four arguments: **model**, **X\_train**, **X\_test**, **y\_train**, and **y\_test**. It fits the **model** to the training data (**X\_train**, **y\_train**), and then calls the **predict()** function to generate predictions on the test data (**X\_test**).

## **6.5 Data Prediction:**

Now after fitting the data into the model, we will be going to predict the actual values and will get predicted values for Y.

This **predict()** function takes one argument: **model**. It generates predictions (**y\_pred**) and probability estimates (**y\_proba**) on the test data using the **model**. It then calculates various evaluation metrics such as accuracy, precision, recall, and F1-score, and returns these metrics to the **calculate\_model\_scores()** function for display. Finally, it plots the ROC curve and confusion matrix using the **calculate\_model\_scores()** function.

## **6.6 Models Used:**

Firstly, We have used **RandomForestClassifier** to train our data.

**GaussianNB** is a good choice when there is a large number of features, and the dataset is not too large. So, For the second model we have chose this for training and predict the values.

Thirdly, We have selected **Categorical\_Naive\_Bayes** for our Model.

And The last model which we have used is **Bernoulli\_Naive\_Bayes.**

## **6.7 Scores:**

For each of the previous model, We are generating different scores to evaluate the performance of the model.

Here we have calculated Accuracy\_Score,Precision\_Score,Recall\_Score,F1 Score,ROC AUC Score.

## **6.8 KNN:**

Now we are performing hyperparameter tuning for the K-Nearest Neighbors (KNN) algorithm using grid search with cross-validation.

The KNN algorithm is a type of instance-based learning or lazy learning, where the model stores the training instances and makes predictions based on the similarity between new instances and the stored instances. The KNN algorithm works by finding the K nearest training instances to the new instance and using the majority class among the K nearest neighbors as the predicted class for the new instance.

The code creates a range of K values from 1 to 15 and a list of different distance metrics (euclidean, manhattan, chebyshev, minkowski, and mahalanobis) that will be used to calculate the distances between the instances.

The GridSearchCV class from scikit-learn is used to perform a grid search over the combinations of the K values and the distance metrics. The cv parameter specifies the number of folds used for cross-validation, and the scoring parameter specifies the evaluation metric used to select the best parameters. In this case, the evaluation metric is set to 'accuracy', which is a common metric for classification problems.

After fitting the grid search object on the training data, the code prints the best parameter combination (best\_params\_) and the best score (best\_score\_) obtained during the grid search. These values can be used to select the best hyperparameters for the KNN model to achieve the highest accuracy on the test data.

## **6.9 Optimal KNN:**

Now we are using optimal KNN by adding some parameters **n\_neighbors=3** and **metric='manhattan'**. This means that the model will use the Manhattan distance metric to calculate the distances between the instances and will consider the 3 nearest neighbors to make predictions.

By fitting and evaluating the KNN model on the test data, we can assess how well the model generalizes to new, unseen data. This is important to ensure that the model is not overfitting to the training data and is able to make accurate predictions on real-world data.

## **6.10 Neural Network Model:**

We used a simple neural network using the Keras Sequential API. Here's what the code does line by line:

1. **model\_S = Sequential()**: Creates a Sequential model object that will represent a neural network.
2. **model\_S.add(Dense(64, input\_dim=X\_train.shape[1], activation='relu'))**: Adds a fully connected layer with 64 units, using the ReLU activation function. The **input\_dim** parameter specifies the number of input features in the dataset.
3. **model\_S.add(Dense(32, activation='relu'))**: Adds another fully connected layer with 32 units and ReLU activation function.
4. **model\_S.add(Dense(1, activation='sigmoid'))**: Adds a final output layer with a single unit and sigmoid activation function. Since this is a binary classification problem (0 or 1), the sigmoid function is used to obtain probabilities of the positive class.
5. **model\_S.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])**: Compiles the model by specifying the loss function, optimizer, and evaluation metric. The **binary\_crossentropy** loss function is used for binary classification problems, and the **adam** optimizer is a popular choice for gradient descent. The **accuracy** metric is used to evaluate the performance of the model during training.
6. **model\_S.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_data=(X\_test, y\_test))**: Trains the model for 10 epochs using the training data **X\_train** and **y\_train**. The **batch\_size** parameter specifies the number of samples used in each gradient update. The **validation\_data** parameter specifies the validation set to be used for evaluation after each epoch.
7. **y\_pred = model\_S.predict(X\_test)**: Makes predictions on the test set using the trained model.
8. **y\_pred = np.round(y\_pred)**: Rounds the predicted probabilities to either 0 or 1, to obtain binary predictions.
9. **accuracy = accuracy\_score(y\_test, y\_pred)**: Calculates the accuracy of the model on the test set, by comparing the predicted labels with the true labels.
10. **print('Accuracy: %.2f%%' % (accuracy \* 100))**: Prints the accuracy of the model on the test set as a percentage.

## **6.11 Parzen Window:**

Now we are fitting a Parzen window (Kernel Density Estimation) on the training data (**X\_train\_pca**) and then computing the log-density of the test data (**X\_test\_pca**) using the **score\_samples** method. Finally, the plot is filled with the exponential of the log-density values. This plot can help visualize the density distribution of the test data in comparison to the training data.

# **7. Model Evaluation:**

## **7.1 Performance Evaluation:**

For Model Evaluation, we are evaluating the performance of each model and plotting a bar chart that compares the scores (accuracy, precision, recall, F1 score, and ROC AUC) for six different models. The x-axis shows the names of the models, and the y-axis shows the scores.

The models being compared are:

- PCA

- RandomForest

- GaussianNB

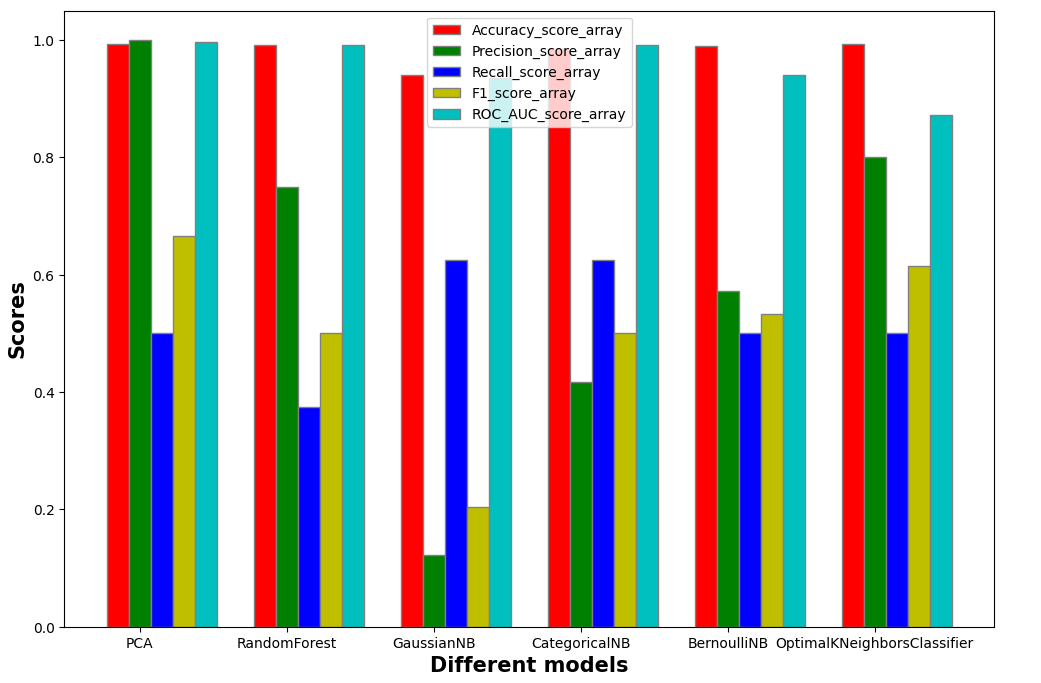
- CategoricalNB

- BernoulliNB

- OptimalKNeighborsClassifier

Each model is represented by a set of five bars, one for each score. The `Accuracy\_score\_array`, `Precision\_score\_array`, `Recall\_score\_array`, `F1\_score\_array`, and `ROC\_AUC\_score\_array` arrays contain the respective scores for each model. The `barWidth` variable sets the width of each bar, and the `fig` variable sets the size of the figure.

The `plt.bar()` function is used to plot the bars for each model and score. The `edgecolor` parameter sets the color of the edge of the bars, and the `label` parameter sets the label for each set of bars in the legend. The `plt.xlabel()` and `plt.ylabel()` functions set the x-axis and y-axis labels, respectively. The `plt.xticks()` function sets the names of the models on the x-axis. Finally, the `plt.legend()` function displays the legend.



## **7.2 Discussion:**

In conclusion, we can say that PCA is the best Model we can use to perform for our problem.