Food Delivery Application Dataset

Exploratory Data Analysis and User Segmentation

By Maral Hamedian

Introduction

In this assignment the aim is to create a user segmentation for getting some insight of type of users of a service. After researching for a while it was found out that one of the best strategies for segmenting the customers is by running an RFM analysis. RFM stands for Recency, Frequency and and Monetary, and by running this kind of segmentation most and least valuable and loyal customers can be found. This could benefit the company in boosting marketing strategies, utilize promotional activities and as a result increase sales.

Data Exploration

Load the nessary packages and the data

In [1]:
```
import numpy as np
import pandas as pd
import warnings
import seaborn as sns
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import scale
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
from sklearn.decomposition import PCA
```

In [2]:
```
df = pd.read_csv("dataset_for_analyst_assignment_20201120.csv")
df.head()
```

Out[2]:

	REGISTRATION_DATE	REGISTRATION_COUNTRY	PURCHASE_COUNT	PURCHASE_COUNT_DELIVER
0	2019-09-01 00:00:00.000	DNK	0	Nal
1	2019-09-01 00:00:00.000	FIN	1	1.
2	2019-09-01 00:00:00.000	DNK	19	19.
3	2019-09-01 00:00:00.000	FIN	0	Nal
4	2019-09-01 00:00:00.000	GRC	0	Nal

5 rows × 30 columns

Let's find out what columns and datatypes do we have

In [3]:
```
df.dtypes
```

Out[3]:
```
REGISTRATION_DATE                        object
REGISTRATION_COUNTRY                     object
PURCHASE_COUNT                            int64
PURCHASE_COUNT_DELIVERY                 float64
PURCHASE_COUNT_TAKEAWAY                 float64
FIRST_PURCHASE_DAY                       object
LAST_PURCHASE_DAY                        object
USER_ID                                   int64
BREAKFAST_PURCHASES                     float64
LUNCH_PURCHASES                         float64
EVENING_PURCHASES                       float64
DINNER_PURCHASES                        float64
LATE_NIGHT_PURCHASES                    float64
TOTAL_PURCHASES_EUR                     float64
DISTINCT_PURCHASE_VENUE_COUNT           float64
MIN_PURCHASE_VALUE_EUR                  float64
MAX_PURCHASE_VALUE_EUR                  float64
AVG_PURCHASE_VALUE_EUR                  float64
PREFERRED_DEVICE                         object
IOS_PURCHASES                           float64
WEB_PURCHASES                           float64
ANDROID_PURCHASES                       float64
PREFERRED_RESTAURANT_TYPES               object
USER_HAS_VALID_PAYMENT_METHOD              bool
MOST_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE float64
MOST_COMMON_WEEKDAY_TO_PURCHASE         float64
AVG_DAYS_BETWEEN_PURCHASES              float64
MEDIAN_DAYS_BETWEEN_PURCHASES           float64
AVERAGE_DELIVERY_DISTANCE_KMS           float64
PURCHASE_COUNT_BY_STORE_TYPE             object
dtype: object
```

In [4]:
```
df.describe()
```

Out[4]:

	PURCHASE_COUNT	PURCHASE_COUNT_DELIVERY	PURCHASE_COUNT_TAKEAWAY	USER_I
count	21983.000000	12028.000000	12028.000000	21983.000(
mean	3.345358	5.741686	0.372464	10992.000(
std	8.523171	10.536220	1.416310	6346.089{
min	0.000000	0.000000	0.000000	1.000(
25%	0.000000	1.000000	0.000000	5496.500(
50%	1.000000	2.000000	0.000000	10992.000(
75%	3.000000	6.000000	0.000000	16487.500(
max	320.000000	320.000000	44.000000	21983.000(

8 rows × 22 columns

In [5]:

```
print(df.shape)
```

(21983, 30)

In [6]:

```
# make sure that none of the users is repeated more than once
df.USER_ID.unique().shape[0]
```

Out[6]: 21983

In [7]:

```
### Let's also check if the First purchase is greater than the last purchase - this
any(df['FIRST_PURCHASE_DAY'] > df['LAST_PURCHASE_DAY'])
```

Out[7]: False

Data Preprocessing

Convert features to suitable types

In [8]:

```
#convert all date and time columns from object type to datetime64[ns]
df["REGISTRATION_DATE"] = df["REGISTRATION_DATE"].astype('datetime64[ns]')
df["FIRST_PURCHASE_DAY"] = df["FIRST_PURCHASE_DAY"].astype('datetime64[ns]')
df["LAST_PURCHASE_DAY"] = df["LAST_PURCHASE_DAY"].astype('datetime64[ns]')

#convert purchase count from object ype to int64
df["PURCHASE_COUNT"] = df["PURCHASE_COUNT"].astype('int64')
```

Missing Values

In [9]:

```
#See the amount of missing values in each feature
df.isnull().sum(axis=0)
```

Out[9]:
```
REGISTRATION_DATE                          0
REGISTRATION_COUNTRY                       0
PURCHASE_COUNT                             0
PURCHASE_COUNT_DELIVERY                 9955
PURCHASE_COUNT_TAKEAWAY                 9955
FIRST_PURCHASE_DAY                     10019
LAST_PURCHASE_DAY                       9956
USER_ID                                    0
BREAKFAST_PURCHASES                     9955
LUNCH_PURCHASES                         9955
EVENING_PURCHASES                       9955
DINNER_PURCHASES                        9955
LATE_NIGHT_PURCHASES                    9955
TOTAL_PURCHASES_EUR                     9955
DISTINCT_PURCHASE_VENUE_COUNT           9955
MIN_PURCHASE_VALUE_EUR                  9955
MAX_PURCHASE_VALUE_EUR                  9955
AVG_PURCHASE_VALUE_EUR                  9955
PREFERRED_DEVICE                          73
IOS_PURCHASES                           9955
WEB_PURCHASES                           9955
ANDROID_PURCHASES                       9955
PREFERRED_RESTAURANT_TYPES             19289
USER_HAS_VALID_PAYMENT_METHOD              0
MOST_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE 9955
MOST_COMMON_WEEKDAY_TO_PURCHASE         9955
AVG_DAYS_BETWEEN_PURCHASES             14151
MEDIAN_DAYS_BETWEEN_PURCHASES          14151
AVERAGE_DELIVERY_DISTANCE_KMS           9955
PURCHASE_COUNT_BY_STORE_TYPE               0
dtype: int64
```

In [10]:
```
len(df[(df.PURCHASE_COUNT == 0)])
```

Out[10]:
```
9955
```

As it can be seen, in most of the features, the missing value equals to 9955,and with little inspection it is found out that this value corresponds to the number of users that have not made any purchases after registration (purchase count = 0). this is proven by printing out a condition where purchase count is 0 and any other attribute that has 9955 missing values, as shown below.

In [11]:
```
df[(df.PURCHASE_COUNT == 0) & (df.TOTAL_PURCHASES_EUR.isnull())]
```

Out[11]:

	REGISTRATION_DATE	REGISTRATION_COUNTRY	PURCHASE_COUNT	PURCHASE_COUNT_DEL
0	2019-09-01	DNK	0	
3	2019-09-01	FIN	0	
4	2019-09-01	GRC	0	
5	2019-09-01	FIN	0	
6	2019-09-01	DNK	0	
...	...	...	...	
21974	2019-09-30	GRC	0	
21977	2019-09-30	GRC	0	
21979	2019-09-30	GRC	0	
21980	2019-09-30	DNK	0	
21981	2019-09-30	DNK	0	

9955 rows × 30 columns

we can also see some features of higher values that are missing. Let's explore why more than 9555 values are missing from First purchase day and last purchase day.

In [12]:

```
# customers who has made purchases but there is no first purchase date
df[(df.PURCHASE_COUNT > 0) & (df.FIRST_PURCHASE_DAY.isnull())]
```

Out[12]:

	REGISTRATION_DATE	REGISTRATION_COUNTRY	PURCHASE_COUNT	PURCHASE_COUNT_DEL
151	2019-09-01	DNK	3	
193	2019-09-01	FIN	1	
400	2019-09-01	DNK	3	
552	2019-09-01	DNK	3	
555	2019-09-01	FIN	2	
...	...	...	...	
20978	2019-09-29	FIN	1	
21094	2019-09-29	DNK	3	
21214	2019-09-29	DNK	13	
21384	2019-09-29	DNK	1	
21451	2019-09-29	DNK	1	

64 rows × 30 columns

In [13]:

```
df[(df.PURCHASE_COUNT != 0) & (df.LAST_PURCHASE_DAY.isnull())]
```

Out[13]:

	REGISTRATION_DATE	REGISTRATION_COUNTRY	PURCHASE_COUNT	PURCHASE_COUNT_DEL
20978	2019-09-29	FIN	1	

1 rows × 30 columns

Average days between purchases and median days between purchases have missing values even if purchase count is greater than 0. It makes sense if among those are customers that ordered one time.

In [14]:

```
df[(df.PURCHASE_COUNT ==1)& (df.AVG_DAYS_BETWEEN_PURCHASES.isnull())]
```

Out[14]:

	REGISTRATION_DATE	REGISTRATION_COUNTRY	PURCHASE_COUNT	PURCHASE_COUNT_DEL
1	2019-09-01	FIN	1	
7	2019-09-01	FIN	1	
22	2019-09-01	FIN	1	
24	2019-09-01	FIN	1	
37	2019-09-01	FIN	1	
...	...	...	...	
21970	2019-09-30	DNK	1	
21973	2019-09-30	FIN	1	
21976	2019-09-30	DNK	1	
21978	2019-09-30	GRC	1	
21982	2019-09-30	GRC	1	

4179 rows × 30 columns

Let's check if there are rows with purchase count greater than 1 and see if any of those have missing values of average and median days between purchases.

In [15]:

```
df[(df.PURCHASE_COUNT > 1)& (df.AVG_DAYS_BETWEEN_PURCHASES.isnull())]
```

Out[15]:

	REGISTRATION_DATE	REGISTRATION_COUNTRY	PURCHASE_COUNT	PURCHASE_COUNT_DEL
707	2019-09-01	DNK	2	
1724	2019-09-03	DNK	2	
1991	2019-09-04	DNK	2	
5769	2019-09-08	DNK	2	
7108	2019-09-10	DNK	2	
7717	2019-09-11	DNK	2	
8164	2019-09-12	DNK	2	
8581	2019-09-13	DNK	2	
9216	2019-09-14	GRC	2	
9616	2019-09-14	DNK	2	
9636	2019-09-14	DNK	2	
11485	2019-09-16	DNK	2	
12650	2019-09-18	DNK	2	
18297	2019-09-25	DNK	2	
20178	2019-09-28	DNK	2	
20230	2019-09-28	DNK	2	
20327	2019-09-28	DNK	2	

17 rows × 30 columns

Seems that there are 17 users with missing values of average and median days between purchases,eventhough they have made purchases twice.

let's handle some of the missing values now

Delete the following rows:

- purchase count > 0 but first and last purchase days are missing
- purchase count > 1 but average and median days between purchases are missing

In [16]:
```
df.drop(df.index[(df.PURCHASE_COUNT > 0) & (df.FIRST_PURCHASE_DAY.isnull())], inpla
df.drop(df.index[(df.PURCHASE_COUNT > 0) & (df.LAST_PURCHASE_DAY.isnull())], inplac
```

In [17]:
```
df.drop(df.index[(df.PURCHASE_COUNT > 1)& (df.AVG_DAYS_BETWEEN_PURCHASES.isnull())]
df.drop(df.index[(df.PURCHASE_COUNT > 1)& (df.MEDIAN_DAYS_BETWEEN_PURCHASES.isnull(
```

for all the remaining rows that have missing values on Average and median days between purchases, substitute 0 (Nan -> 0.0).

In [18]:
```
### Replacing 'Nan' values in AVG_DAYS_BETWEEN_PURCHASES with 0
df['AVG_DAYS_BETWEEN_PURCHASES'][df['AVG_DAYS_BETWEEN_PURCHASES'].isnull()] = 0.0
### Replacing 'Nan' values in MEDIAN_DAYS_BETWEEN_PURCHASES with 0
df['MEDIAN_DAYS_BETWEEN_PURCHASES'][df['MEDIAN_DAYS_BETWEEN_PURCHASES'].isnull()] =
```

Do we have any rows with purchase count greater than 1, but average and median days between purchases equal to 0? Errors, they should be removed

In [19]:
```
df[(df.PURCHASE_COUNT > 1)& (df.AVG_DAYS_BETWEEN_PURCHASES == 0.0)]
```

Out[19]:

	REGISTRATION_DATE	REGISTRATION_COUNTRY	PURCHASE_COUNT	PURCHASE_COUNT_DEL
269	2019-09-01	GRC	2	
282	2019-09-01	DNK	2	
529	2019-09-01	DNK	2	
1214	2019-09-02	DNK	2	
1694	2019-09-03	DNK	2	
...	...	...	...	
19111	2019-09-27	GRC	2	
19437	2019-09-27	DNK	2	
19925	2019-09-28	FIN	2	
20247	2019-09-28	FIN	2	
20896	2019-09-29	FIN	2	

62 rows × 30 columns

In [20]:
```
df.drop(df.index[(df.PURCHASE_COUNT > 1)& (df.AVG_DAYS_BETWEEN_PURCHASES == 0.0)],
```

In [21]:
```
df[(df.PURCHASE_COUNT > 1)& (df.MEDIAN_DAYS_BETWEEN_PURCHASES == 0.0)]
```

Out[21]:

	REGISTRATION_DATE	REGISTRATION_COUNTRY	PURCHASE_COUNT	PURCHASE_COUNT_DEL
2101	2019-09-04	FIN	18	
4072	2019-09-06	GRC	176	
9185	2019-09-13	FIN	18	
11225	2019-09-16	DNK	10	
16922	2019-09-23	GRC	4	
21702	2019-09-30	FIN	14	

6 rows × 30 columns

In [22]:
```
df.drop(df.index[(df.PURCHASE_COUNT > 1)& (df.MEDIAN_DAYS_BETWEEN_PURCHASES == 0.0)
```

In [23]:
```
#confirm
print('# of 0 average days between purchases when purchase count > 1: '+ str(len(df
print('# of 0 Median days between purchases when purchase count > 1: '+ str(len(df[
```

```
# of 0 average days between purchases when purchase count > 1: 0
# of 0 Median days between purchases when purchase count > 1: 0
```

since purchase count is 0 when 9955 values of missing from different features, we substitute the missing values with 0. This action will take place in the following feature columns:

- PURCHASE_COUNT_DELIVERY
- PURCHASE_COUNT_TAKEAWAY
- BREAKFAST_PURCHASES
- LUNCH_PURCHASES
- EVENING_PURCHASES
- DINNER_PURCHASES
- LATE_NIGHT_PURCHASES
- TOTAL_PURCHASES_EUR
- DISTINCT_PURCHASE_VENUE_COUNT
- MIN_PURCHASE_VALUE_EUR
- MAX_PURCHASE_VALUE_EUR
- AVG_PURCHASE_VALUE_EUR
- IOS_PURCHASES
- WEB_PURCHASES
- ANDROID_PURCHASES
- MOST_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE
- MOST_COMMON_WEEKDAY_TO_PURCHASE

- AVERAGE_DELIVERY_DISTANCE_KMS

In [24]:
```
df['PURCHASE_COUNT_DELIVERY'][(df['PURCHASE_COUNT'] == 0) & (df['PURCHASE_COUNT_DEL
df['PURCHASE_COUNT_TAKEAWAY'][(df['PURCHASE_COUNT'] == 0) & (df['PURCHASE_COUNT_TAK
df['BREAKFAST_PURCHASES'][(df['PURCHASE_COUNT'] == 0) & (df['BREAKFAST_PURCHASES'].
df['LUNCH_PURCHASES'][(df['PURCHASE_COUNT'] == 0) & (df['LUNCH_PURCHASES'].isnull()
df['EVENING_PURCHASES'][(df['PURCHASE_COUNT'] == 0) & (df['EVENING_PURCHASES'].isnu
df['DINNER_PURCHASES'][(df['PURCHASE_COUNT'] == 0) & (df['DINNER_PURCHASES'].isnull
df['LATE_NIGHT_PURCHASES'][(df['PURCHASE_COUNT'] == 0) & (df['LATE_NIGHT_PURCHASES'
df['TOTAL_PURCHASES_EUR'][(df['PURCHASE_COUNT'] == 0) & (df['TOTAL_PURCHASES_EUR'].
df['DISTINCT_PURCHASE_VENUE_COUNT'][(df['PURCHASE_COUNT'] == 0) & (df['DISTINCT_PUR
df['MIN_PURCHASE_VALUE_EUR'][(df['PURCHASE_COUNT'] == 0) & (df['MIN_PURCHASE_VALUE_
df['MAX_PURCHASE_VALUE_EUR'][(df['PURCHASE_COUNT'] == 0) & (df['MAX_PURCHASE_VALUE_
df['AVG_PURCHASE_VALUE_EUR'][(df['PURCHASE_COUNT'] == 0) & (df['AVG_PURCHASE_VALUE_
df['IOS_PURCHASES'][(df['PURCHASE_COUNT'] == 0) & (df['IOS_PURCHASES'].isnull()) ]
df['WEB_PURCHASES'][(df['PURCHASE_COUNT'] == 0) & (df['WEB_PURCHASES'].isnull()) ]
df['ANDROID_PURCHASES'][(df['PURCHASE_COUNT'] == 0) & (df['ANDROID_PURCHASES'].isnu
df['MOST_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE'][(df['PURCHASE_COUNT'] == 0) & (df['MO
df['MOST_COMMON_WEEKDAY_TO_PURCHASE'][(df['PURCHASE_COUNT'] == 0) & (df['MOST_COMMO
df['AVERAGE_DELIVERY_DISTANCE_KMS'][(df['PURCHASE_COUNT'] == 0) & (df['AVERAGE_DELI
```

Handle Missing values on First purchase day and Last purchase day

It is important to also treat the missing dates in first and last purchases. AS we said before these values are missing because the customers have not made any purchases, so their purchase count is 0. One way to treat them is to set it to the time that has not occured yet, e.g. 21th March, 2025.

Before we change the non existing date values, let's save the most recent last purchase day into a variable. we are going to need it later.

In [25]:
```
Latest_purchase_day = df.LAST_PURCHASE_DAY.max()
Latest_purchase_day
```

Out[25]:
```
Timestamp('2020-10-31 00:00:00')
```

In [26]:
```
#set the non existing first and last purchase days to 21th March, 2025.
df['FIRST_PURCHASE_DAY'][(df['PURCHASE_COUNT'] == 0) & (df['FIRST_PURCHASE_DAY'].is
df['LAST_PURCHASE_DAY'][(df['PURCHASE_COUNT'] == 0) & (df['LAST_PURCHASE_DAY'].isnu
```

In [27]:
```
# confirm the correct data types of first and last purchase day
df['FIRST_PURCHASE_DAY'].dtypes
```

Out[27]:
```
dtype('<M8[ns]')
```

In [28]:
```
#confirm that the most recent last purchase day is now 21th March, 2025
print('Latest Last purchase day: '+ str(df.LAST_PURCHASE_DAY.max()))
```

```
Latest Last purchase day: 2025-03-21 00:00:00
```

In [29]:
```
#Reset index rows of the dataset
df.reset_index(drop=True, inplace=True)
```

There are other missing values as shown below, but we are not going to use those in our segmentation analysis... so let's skip dealing with those

some more exploration

Let's see if we find any correlation between our features

```
In [30]: df.corr()
```

Out[30]:

	PURCHASE_COUNT	PURCHASE_COUNT_DELIVER
PURCHASE_COUNT	1.000000	0.9922
PURCHASE_COUNT_DELIVERY	0.992290	1.0000
PURCHASE_COUNT_TAKEAWAY	0.270900	0.1495
USER_ID	0.003732	0.0037
BREAKFAST_PURCHASES	0.459978	0.4573
LUNCH_PURCHASES	0.886301	0.8831
EVENING_PURCHASES	0.516019	0.5202
DINNER_PURCHASES	0.853717	0.8404
LATE_NIGHT_PURCHASES	NaN	Na
TOTAL_PURCHASES_EUR	0.894808	0.8840
DISTINCT_PURCHASE_VENUE_COUNT	0.811502	0.7953
MIN_PURCHASE_VALUE_EUR	0.095353	0.0929
MAX_PURCHASE_VALUE_EUR	0.371204	0.3555
AVG_PURCHASE_VALUE_EUR	0.221162	0.2116
IOS_PURCHASES	0.636007	0.6263
WEB_PURCHASES	0.479758	0.4805
ANDROID_PURCHASES	0.612322	0.6096
USER_HAS_VALID_PAYMENT_METHOD	0.274512	0.2697
MOST_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE	0.267796	0.2572
MOST_COMMON_WEEKDAY_TO_PURCHASE	0.287727	0.2768
AVG_DAYS_BETWEEN_PURCHASES	0.061365	0.0554
MEDIAN_DAYS_BETWEEN_PURCHASES	0.025393	0.0210
AVERAGE_DELIVERY_DISTANCE_KMS	0.263942	0.2535

23 rows × 23 columns

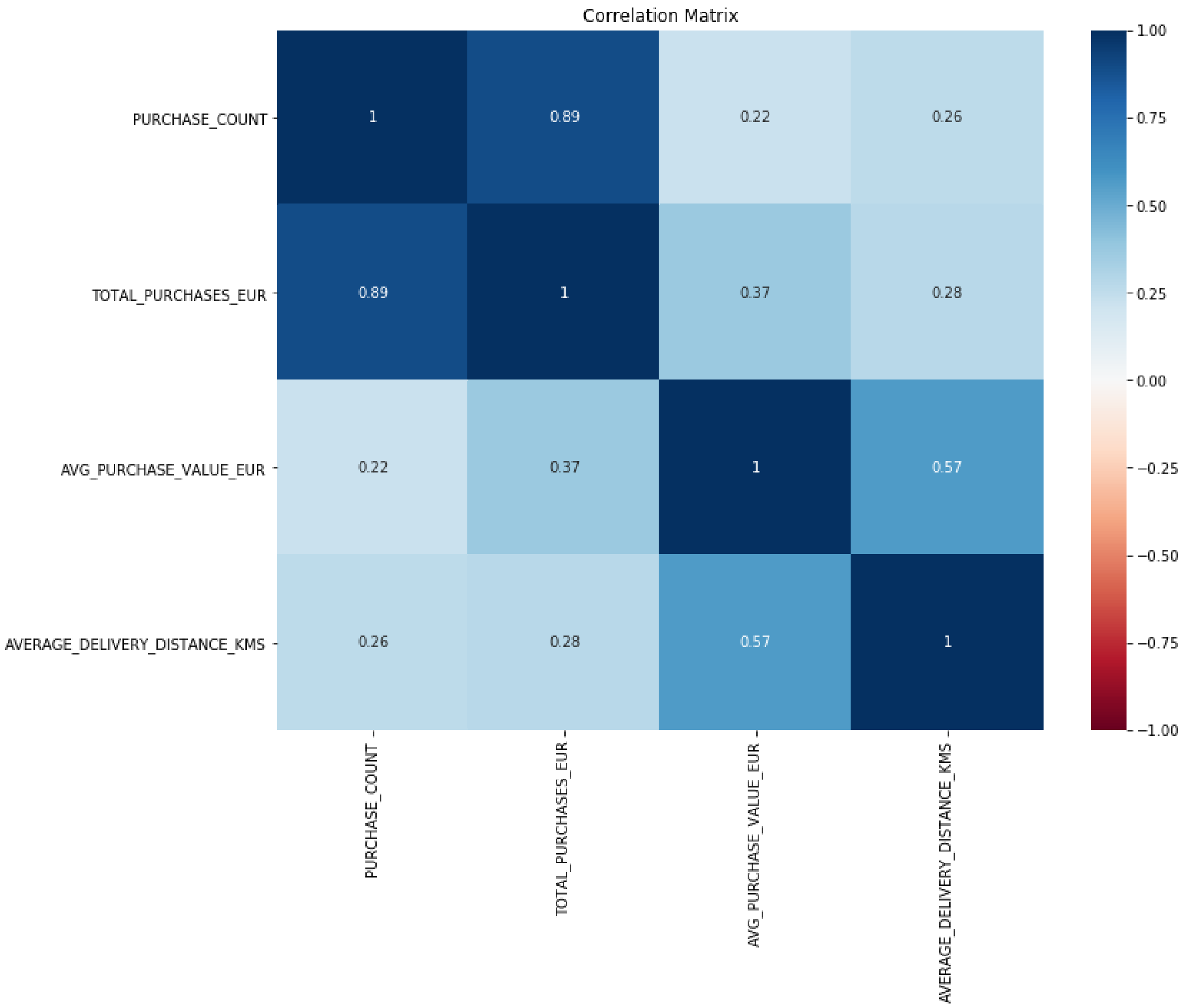
Let's narrow it down

```
In [31]: #correlation of purchase count, total purchases, average purchase values, most comm
df.iloc[:, [2, 13,17,28]].corr()
```

Out[31]:

	PURCHASE_COUNT	TOTAL_PURCHASES_EUR	AVG_PURCHASE_V
PURCHASE_COUNT	1.000000	0.894808	
TOTAL_PURCHASES_EUR	0.894808	1.000000	
AVG_PURCHASE_VALUE_EUR	0.221162	0.374336	
AVERAGE_DELIVERY_DISTANCE_KMS	0.263942	0.279410	

In [32]:
```
plt.figure(figsize=(12, 9))
s = sns.heatmap(df.iloc[:, [2, 13,17,28]].corr(),
                annot=True,
                cmap='RdBu',
                vmin=-1,
                vmax=1)
s.set_xticklabels(s.get_xticklabels(), rotation=90)
plt.title('Correlation Matrix')
plt.show()
```

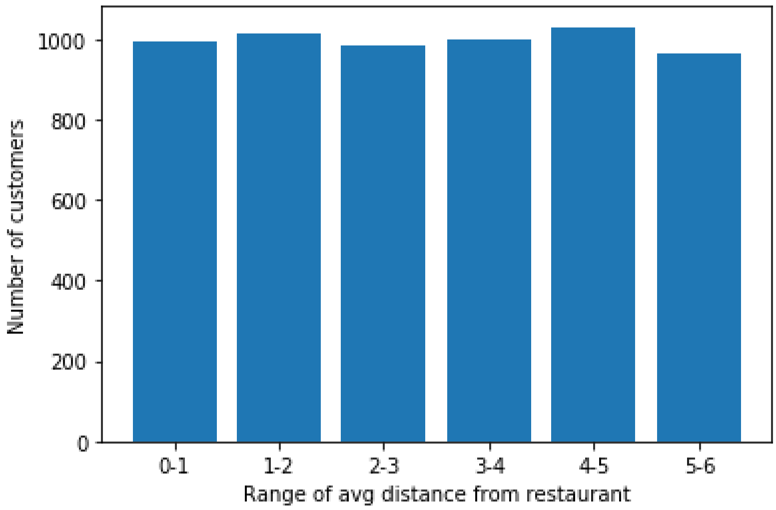


It seem that there is a high positive correlation between purchase count and total purcahses in terms of money. Also, it seems that there is a strong correlation between average delivery distance and average purchase value. Let's see at what distances from resturants most of the users are. Note: customers whose purchase count is 0 are excluded from this as we want to see the true value of distances to the resturants.

In [33]:
```
n_cust_dist = [len(df.USER_ID[(df.AVERAGE_DELIVERY_DISTANCE_KMS > 0) & (df.AVERAGE_
len(df.USER_ID[(df.AVERAGE_DELIVERY_DISTANCE_KMS >= 1) & (df.AVERAGE_DELIVERY_DISTA
```

```
len(df.USER_ID[(df.AVERAGE_DELIVERY_DISTANCE_KMS >= 2) & (df.AVERAGE_DELIVERY_DISTA
len(df.USER_ID[(df.AVERAGE_DELIVERY_DISTANCE_KMS >= 3) & (df.AVERAGE_DELIVERY_DISTA
len(df.USER_ID[(df.AVERAGE_DELIVERY_DISTANCE_KMS >= 4) & (df.AVERAGE_DELIVERY_DISTA
len(df.USER_ID[(df.AVERAGE_DELIVERY_DISTANCE_KMS >= 5) & (df.AVERAGE_DELIVERY_DISTA

avg_dist = ['0-1', '1-2', '2-3', '3-4', '4-5', '5-6']
plt.bar(avg_dist, n_cust_dist)
plt.xlabel('Range of avg distance from restaurant')
plt.ylabel('Number of customers')
plt.show()
```



Seems that the users are equally distributed between different range of delivery distance.

Let's now move on to segment the users.

RFM Analysis

AS explained in the beginning, RFM (Recency, Frequency, Monetary) segmentation method is chosen to find out the purchasing behaviour of the users.

Recency - How recent is the customer last purchase

Frequency - How often did customer purchase / how many times purchased in total

Monetary - How much money they have spend on the service in total

- most valuable customers are potentially the ones who purchased most recently, have purchased a lot and spent a lot

For segmenting the customers according to the RFM analysis, we are going to need to see the amount of purchases of each customer, the money they have spent in total and also calculate the recency from their last order.

In [34]:
```
df_rfm = df.loc[:, ['USER_ID', 'PURCHASE_COUNT', 'TOTAL_PURCHASES_EUR', 'LAST_PURCH
```

In [35]:
```
df_rfm.head()
```

Out[35]:

	USER_ID	PURCHASE_COUNT	TOTAL_PURCHASES_EUR	LAST_PURCHASE_DAY
0	1	0	0.000	2025-03-21
1	2	1	38.456	2020-09-02
2	3	19	631.488	2020-05-25
3	4	0	0.000	2025-03-21
4	5	0	0.000	2025-03-21

Recency

Calculate Racency last purchase. The most recent last purchase overall in the data is 31th October, 2020 and we have saved it as Latest_purchase_day. We'll use that day as a reference to be the most recent date and substract the last purchase days of each customer from it. the formula is: Absolute value (Latest purchase day - customer purchase day)

- Note: The smaller the resulting recency number is, the more recent the user purchase.

In [36]:

```
recency_of_last_purchases = []
for i in range(0, df_rfm.shape[0]):
    recency_of_last_purchases.append(abs((Latest_purchase_day - df_rfm['LAST_PURCHA

df_rfm['RECENCY'] = pd.Series(recency_of_last_purchases)
```

In [37]:

```
df_rfm.head()
```

Out[37]:

	USER_ID	PURCHASE_COUNT	TOTAL_PURCHASES_EUR	LAST_PURCHASE_DAY	RECENCY
0	1	0	0.000	2025-03-21	1602
1	2	1	38.456	2020-09-02	59
2	3	19	631.488	2020-05-25	159
3	4	0	0.000	2025-03-21	1602
4	5	0	0.000	2025-03-21	1602

In [38]:

```
df['RECENCY'] = (df_rfm['RECENCY']) #add recency to the main dataframe
```

Frequency and Monetary

The Frequency of this analysis translate to how many times the customer has ordered in total -> purchase count.

The Monetary term, as the name suggests, refers to the total amount of money that each customer has spent -> total purchases in EUR.

seems that we have everything we need so let's run an algorithm for segmenting or clustering the users and find out which ones are most valuable.

K-means algorithm to for RFM Segmentation

It is decided that K-means machine learning clustering algorithm is used since it's relatively simple to implement and it has shown that the resulting segmentation is accurate and reliable in a sense that it puts most similar characteristic data together.

How does it work?

Wee need to give the k means algorithm the number of clusters we want to segment our data. Afterwards the algirthm works in a way that it will randomly assign some points in data space, called the centroids, (one for each cluster), and calculates the sum of distances to of the group of data it has randomly made. This is an iterative process, where centroids move several times until suitable clusters are formed with minimum sum of distances of data points to their own cluster centroid, and within clusters data points have similar features.

More about k-means algorithm in this link:
https://www.analyticsvidhya.com/blog/2021/11/understanding-k-means-clustering-in-machine-learningwith-examples/

One important issue that we need to take into account when dealing with k-means algorithm is to standardized, here we'll use Scikit-learn StandardScaler method.

In [39]:
```
#first we need to standardize our data
scaler = StandardScaler()
df_rfm.iloc[:,[1,2,4]]=scaler.fit_transform(df_rfm.iloc[:,[1,2,4]])
```

In [40]:
```
df_rfm.head()
```

Out[40]:

	USER_ID	PURCHASE_COUNT	TOTAL_PURCHASES_EUR	LAST_PURCHASE_DAY	RECENCY
0	1	-0.394539	-0.414641	2025-03-21	1.077517
1	2	-0.276420	-0.248398	2020-09-02	-1.092241
2	3	1.849732	2.315248	2020-05-25	-0.951621
3	4	-0.394539	-0.414641	2025-03-21	1.077517
4	5	-0.394539	-0.414641	2025-03-21	1.077517

Finding suitbale number of k-cluster number to feed into algorithm

Elbow method - finding suitable number of clusters

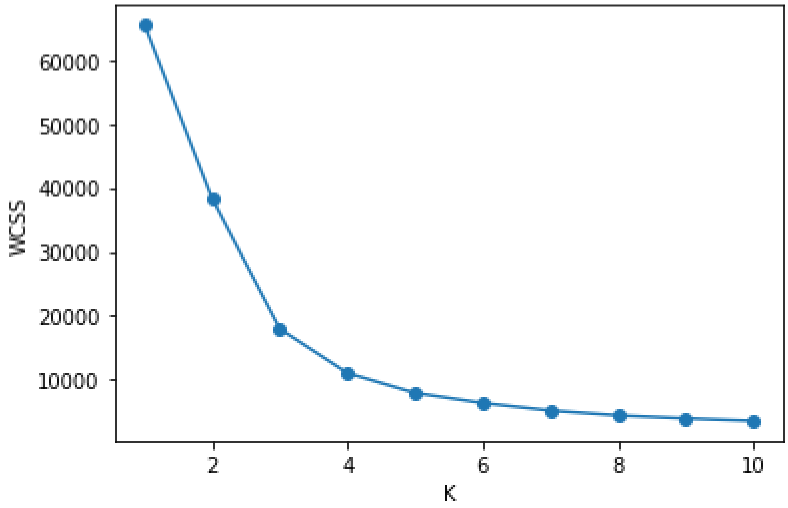
We need to find a suitable number of clusters to be feeded in k-means algorithm, for whcih will use the Elbow method,where through a plot we find the minimum number of clusters that could be meaningful enough to segment our data into, so that we avoid extra cluster formation that does not really reveal any variability in data.

The plot is of potential k number of clusters agaist sum of square within cluster (WCSS), where we can find the suitable number of cluster value when there is a abrupt change in the shape of the plot, like the shape of the elbow.

By using python inertia method when creating the elbow plot, the sum of squared errors within clusters would be minimum, while sum of squared errors between each cluster would be maximum.

In [41]:
```python
### Elbow Analysis to find number of clusters ###
n_clusters = range(1, 11)
cluster_sse = []
for i in n_clusters:
    kmeans = KMeans(n_clusters= i)
    kmeans.fit(df_rfm.iloc[:, [1, 2, 4]])
    cluster_sse.append(kmeans.inertia_)

plt.plot(n_clusters, cluster_sse, marker = "o")
plt.xlabel('K')
plt.ylabel('WCSS')
plt.show()
```



From the above plot we can see that the elbow shape is forming at the point where cluster number is between 3-4. so it's safe to have the cluster number at 4. Now let's use that value into the algorithm to segment our users based on Recency, purchase count and total purchase value.

In [42]:
```python
#Let's segment into clusters using k-means algorithm
km_cluster = KMeans(n_clusters = 4, random_state= 22)
df_rfm['CLUSTER'] = km_cluster.fit_predict(df_rfm.iloc[:,[1,2,4]])
```

In [43]:
```python
df_rfm.head()
```

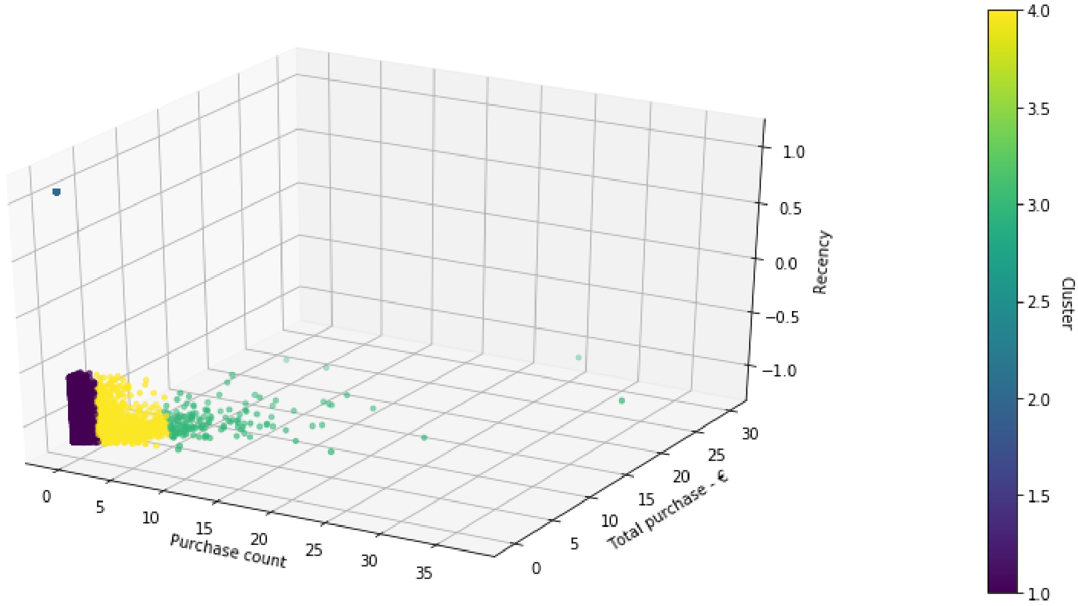
Out[43]:

	USER_ID	PURCHASE_COUNT	TOTAL_PURCHASES_EUR	LAST_PURCHASE_DAY	RECENCY	CLUSTE
0	1	-0.394539	-0.414641	2025-03-21	1.077517	
1	2	-0.276420	-0.248398	2020-09-02	-1.092241	
2	3	1.849732	2.315248	2020-05-25	-0.951621	
3	4	-0.394539	-0.414641	2025-03-21	1.077517	
4	5	-0.394539	-0.414641	2025-03-21	1.077517	

Let's plot the datapoints with their clusters

In [44]:
```python
fig = plt.figure(figsize= (15, 7))
ax = fig.add_subplot(111, projection='3d')
graph = ax.scatter(xs = df_rfm['PURCHASE_COUNT'], ys = df_rfm['TOTAL_PURCHASES_EUR'
ax.set_xlabel('Purchase count')
ax.set_ylabel('Total purchase - €')
```

```
ax.set_zlabel('Recency')
cb = fig.colorbar(graph, pad=0.1)
cb.set_label('Cluster', rotation= -90, va='bottom')
plt.show()
```

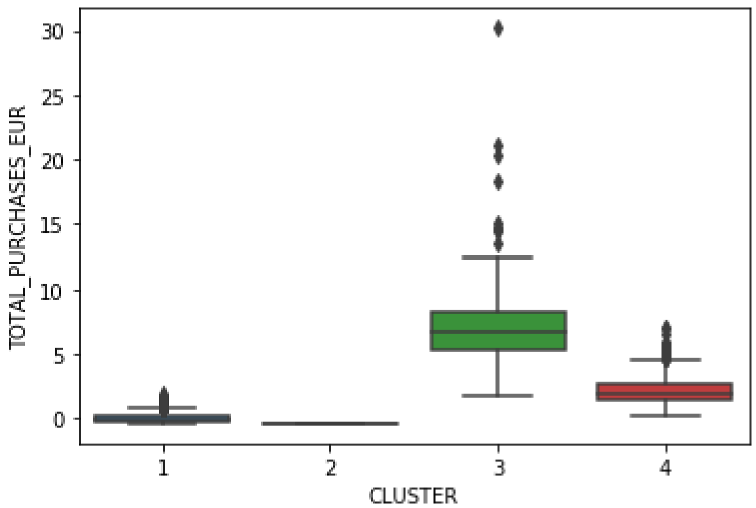


This 3-d graph shows that cluster 3 users are most valuable, since they have the lowest recency from their last purchase, and have the highest total purchase and purchase count.

Let's explore how each user segment present in each of the RFM values.

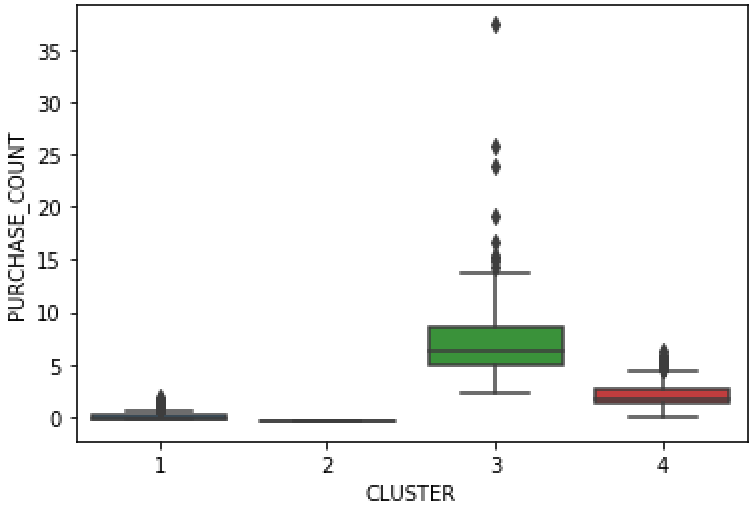
In [45]:
```
sns.boxplot(x=(df_rfm['CLUSTER'] + 1), y=df_rfm['TOTAL_PURCHASES_EUR'], data=df_rfm
```

Out[45]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2ec32b09948>
```



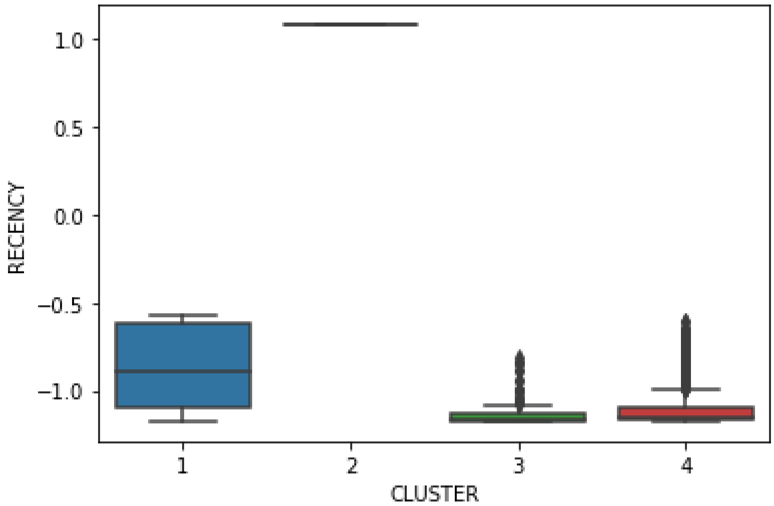
In [46]:
```
sns.boxplot(x=(df_rfm['CLUSTER'] + 1), y=df_rfm['PURCHASE_COUNT'], data=df_rfm)
```

Out[46]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2ec332a8a08>
```



In [47]:
```
sns.boxplot(x=(df_rfm['CLUSTER'] + 1), y=df_rfm['RECENCY'], data=df_rfm)
```

Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x2ec333662c8>



Aggregation of cluster information with main dataset.

In [48]:
```
df['CLUSTER'] = (df_rfm['CLUSTER'] + 1)    ### to account for cluster indices, as ir
```

In [49]:
```
df.head()
```

Out[49]:

	REGISTRATION_DATE	REGISTRATION_COUNTRY	PURCHASE_COUNT	PURCHASE_COUNT_DELIVER
0	2019-09-01	DNK	0	0.
1	2019-09-01	FIN	1	1.
2	2019-09-01	DNK	19	19.
3	2019-09-01	FIN	0	0.
4	2019-09-01	GRC	0	0.

5 rows × 32 columns

Let's form an informative table where we can have the user segments and their calculated mean of different features

In [50]:

```
round(df.iloc[:,].groupby('CLUSTER').mean(), 2)
```

Out[50]:

CLUSTER	PURCHASE_COUNT	PURCHASE_COUNT_DELIVERY	PURCHASE_COUNT_TAKEAWAY	USER_I
1	3.13	2.89	0.24	11034.6
2	0.00	0.00	0.00	10938.6
3	67.00	64.95	2.05	11223.7
4	20.34	19.25	1.09	11104.2

4 rows × 24 columns

From the table above and the box plot we can see some valuable information!

- As said earlier, the 3rd cluster/segment of users are standing out as the top users of the service, the mean purchase count and total purchase value is much higher than any other cluster, while there has not been much time gone from their last purchase.
- Apart from the users who has not purchased at all (cluster 2), cluster 1 is the least performing group, their purchase count and total purchase value is lower than cluster number 3 and 4.
- Most common purchases of cluster number 3 and 4 falls between lunch and dinner purchases

We have successfully been able to segment the users of the service based on their Recency, frequency and monetary purchase behaviou, by utilizing k-mean ML clustering algorithm.