In [616…

```
import pandas as pd
import numpy as np
import missingno as msno
import matplotlib.pyplot as plt
import ast
import plotly.express as px
import seaborn as sns
from datetime import date as dt
import calendar
import datetime as dt

import warnings
warnings.filterwarnings('ignore')
import plotly.express as px
```

In [617…

```
#Read  Customer data input
cus_data_raw = pd.read_csv("C:\\Users\\Namana\\OneDrive\\Desktop\\Projects\\Custome
```

In [618…

```
cus_data_raw['PREFERRED_RESTAURANT_TYPES'] = cus_data_raw['PREFERRED_RESTAURANT_TYP
cus_data_raw['PREFERRED_RESTAURANT_TYPES'] = cus_data_raw['PREFERRED_RESTAURANT_TYP
cus_data_raw["preferred_American_restaurant"] = cus_data_raw["PREFERRED_RESTAURANT_
cus_data_raw["preferred_Japanese_restaurant"] = cus_data_raw["PREFERRED_RESTAURANT_
cus_data_raw["preferred_Italian_restaurant"] = cus_data_raw["PREFERRED_RESTAURANT_T
cus_data_raw["preferred_Mexican_restaurant"] = cus_data_raw["PREFERRED_RESTAURANT_T
cus_data_raw["preferred_Indian_restaurant"] = cus_data_raw["PREFERRED_RESTAURANT_TY
cus_data_raw["preferred_Middleeastern_restaurant"] = cus_data_raw["PREFERRED_RESTAU
cus_data_raw["preferred_Korean_restaurant"] = cus_data_raw["PREFERRED_RESTAURANT_TY
cus_data_raw["preferred_Thai_restaurant"] = cus_data_raw["PREFERRED_RESTAURANT_TYPE
cus_data_raw["preferred_Vietnamese_restaurant"] = cus_data_raw["PREFERRED_RESTAURAN
cus_data_raw["preferred_Hawaiian_restaurant"] = cus_data_raw["PREFERRED_RESTAURANT_
cus_data_raw["preferred_Greek_restaurant"] = cus_data_raw["PREFERRED_RESTAURANT_TYP
cus_data_raw["preferred_Spanish_restaurant"] = cus_data_raw["PREFERRED_RESTAURANT_T
cus_data_raw["preferred_Nepalese_restaurant"] = cus_data_raw["PREFERRED_RESTAURANT_
cus_data_raw["preferred_Chinese_restaurant"] = cus_data_raw["PREFERRED_RESTAURANT_T

#I would like to validate that any preffered resturatnt types  are  not left by cho
cus_data_raw['PREFERRED_RESTAURANT_TYPES'] = cus_data_raw['PREFERRED_RESTAURANT_TYP
cus_data_raw['PREFERRED_RESTAURANT_TYPES'] = cus_data_raw['PREFERRED_RESTAURANT_TYP
cus_data_raw['PREFERRED_RESTAURANT_TYPES'] = cus_data_raw['PREFERRED_RESTAURANT_TYP
cus_data_raw['PREFERRED_RESTAURANT_TYPES'] = cus_data_raw['PREFERRED_RESTAURANT_TYP
cus_data_raw['PREFERRED_RESTAURANT_TYPES'] = cus_data_raw['PREFERRED_RESTAURANT_TYP
cus_data_raw['PREFERRED_RESTAURANT_TYPES'] = cus_data_raw['PREFERRED_RESTAURANT_TYP
cus_data_raw['PREFERRED_RESTAURANT_TYPES'] = cus_data_raw['PREFERRED_RESTAURANT_TYP
cus_data_raw['PREFERRED_RESTAURANT_TYPES'] = cus_data_raw['PREFERRED_RESTAURANT_TYP
cus_data_raw['PREFERRED_RESTAURANT_TYPES'] = cus_data_raw['PREFERRED_RESTAURANT_TYP
cus_data_raw['PREFERRED_RESTAURANT_TYPES'] = cus_data_raw['PREFERRED_RESTAURANT_TYP
cus_data_raw['PREFERRED_RESTAURANT_TYPES'] = cus_data_raw['PREFERRED_RESTAURANT_TYP
cus_data_raw['PREFERRED_RESTAURANT_TYPES'] = cus_data_raw['PREFERRED_RESTAURANT_TYP
cus_data_raw['PREFERRED_RESTAURANT_TYPES'] = cus_data_raw['PREFERRED_RESTAURANT_TYP
cus_data_raw['PREFERRED_RESTAURANT_TYPES'] = cus_data_raw['PREFERRED_RESTAURANT_TYP
#cus_data_raw.PREFERRED_RESTAURANT_TYPES.value_counts()

#droping the column
cus_data_raw.drop('PREFERRED_RESTAURANT_TYPES', axis=1,inplace=True)
```

In [619…

```
# Transforming keys and values in PURCHASE_COUNT_BY_STORE_TYPE into new columns in
store_columns = []                   # creating an empty data to store new data col
#cus_data_raw=cus_data_raw.reset_index()
for i in cus_data_raw.PURCHASE_COUNT_BY_STORE_TYPE:
    store_columns.append(eval(i))  # using eval() to transform the type of data int
```

```
df = pd.DataFrame(store_columns)    # turning it into a new dataframe
cus_data_1 = pd.concat([cus_data_raw,df],axis=1)  # concatenating two dataframes wi
```

In [620…

```
# Renaming some columns having spaces between words and making it easir for usage l
cus_data_1.rename(columns ={'General merchandise':'GENERAL_MERCHANDISE','Pet suppli
                            'Retail store':'RETAIL_STORE','Grocery':'GROCERY','Res
cus_data_raw=cus_data_1.drop(columns=['PURCHASE_COUNT_BY_STORE_TYPE'])
```

Data Understanding

In [621…

```
cus_data_raw.shape
```

Out[621]:

```
(21983, 47)
```

In [622…

```
cus_data_raw .info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21983 entries, 0 to 21982
Data columns (total 47 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   REGISTRATION_DATE                     21983 non-null  object
 1   REGISTRATION_COUNTRY                  21983 non-null  object
 2   PURCHASE_COUNT                        21983 non-null  int64
 3   PURCHASE_COUNT_DELIVERY               12028 non-null  float64
 4   PURCHASE_COUNT_TAKEAWAY               12028 non-null  float64
 5   FIRST_PURCHASE_DAY                    11964 non-null  object
 6   LAST_PURCHASE_DAY                     12027 non-null  object
 7   USER_ID                               21983 non-null  int64
 8   BREAKFAST_PURCHASES                   12028 non-null  float64
 9   LUNCH_PURCHASES                       12028 non-null  float64
 10  EVENING_PURCHASES                     12028 non-null  float64
 11  DINNER_PURCHASES                      12028 non-null  float64
 12  LATE_NIGHT_PURCHASES                  12028 non-null  float64
 13  TOTAL_PURCHASES_EUR                   12028 non-null  float64
 14  DISTINCT_PURCHASE_VENUE_COUNT         12028 non-null  float64
 15  MIN_PURCHASE_VALUE_EUR                12028 non-null  float64
 16  MAX_PURCHASE_VALUE_EUR                12028 non-null  float64
 17  AVG_PURCHASE_VALUE_EUR                12028 non-null  float64
 18  PREFERRED_DEVICE                      21910 non-null  object
 19  IOS_PURCHASES                         12028 non-null  float64
 20  WEB_PURCHASES                         12028 non-null  float64
 21  ANDROID_PURCHASES                     12028 non-null  float64
 22  USER_HAS_VALID_PAYMENT_METHOD         21983 non-null  bool
 23  MOST_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE  12028 non-null  float64
 24  MOST_COMMON_WEEKDAY_TO_PURCHASE       12028 non-null  float64
 25  AVG_DAYS_BETWEEN_PURCHASES            7832 non-null   float64
 26  MEDIAN_DAYS_BETWEEN_PURCHASES         7832 non-null   float64
 27  AVERAGE_DELIVERY_DISTANCE_KMS         12028 non-null  float64
 28  preferred_American_restaurant         21983 non-null  bool
 29  preferred_Japanese_restaurant         21983 non-null  bool
 30  preferred_Italian_restaurant          21983 non-null  bool
 31  preferred_Mexican_restaurant          21983 non-null  bool
 32  preferred_Indian_restaurant           21983 non-null  bool
 33  preferred_Middleeastern_restaurant    21983 non-null  bool
 34  preferred_Korean_restaurant           21983 non-null  bool
 35  preferred_Thai_restaurant             21983 non-null  bool
 36  preferred_Vietnamese_restaurant       21983 non-null  bool
 37  preferred_Hawaiian_restaurant         21983 non-null  bool
 38  preferred_Greek_restaurant            21983 non-null  bool
 39  preferred_Spanish_restaurant          21983 non-null  bool
 40  preferred_Nepalese_restaurant         21983 non-null  bool
 41  preferred_Chinese_restaurant          21983 non-null  bool
 42  GENERAL_MERCHANDISE                   21983 non-null  int64
 43  GROCERY                               21983 non-null  int64
 44  PET_SUPPLIES                          21983 non-null  int64
 45  RESTAURANT                            21983 non-null  int64
 46  RETAIL_STORE                          21983 non-null  int64
dtypes: bool(15), float64(20), int64(7), object(5)
memory usage: 5.7+ MB
```

cus_data_raw['REGISTRATION_DATE']=pd.to_datetime(cus_data_raw['REGISTRATION_DATE'])
cus_data_raw['FIRST_PURCHASE_DAY']=pd.to_datetime(cus_data_raw['FIRST_PURCHASE_DAY'])
cus_data_raw['LAST_PURCHASE_DAY']=pd.to_datetime(cus_data_raw['LAST_PURCHASE_DAY'])

In [623…

```
# Creating new "USER_TYPE" column with the following conditions:

cus_data_raw.loc[(cus_data_raw.PURCHASE_COUNT > 1) & (cus_data_raw.AVG_DAYS_BETWEEN
                 'USER_TYPE' ] = "Churning users"
cus_data_raw.loc[(cus_data_raw.PURCHASE_COUNT > 1) & (cus_data_raw.AVG_DAYS_BETWEEN
                 'USER_TYPE'] = "Active users"
```

```
cus_data_raw.loc[cus_data_raw.PURCHASE_COUNT == 1, 'USER_TYPE'] = "First-time or or
cus_data_raw.loc[cus_data_raw.PURCHASE_COUNT == 0, 'USER_TYPE'] = "Inactive users"
```

In [624…

```
cus_data_raw.duplicated().sum()
```

Out[624]:

```
0
```

In [625…

```
datacheck1=cus_data_raw[cus_data_raw['PURCHASE_COUNT']==0]
len(datacheck1)
```

Out[625]:

```
9955
```

In [626…

```
#Missing values of each column
print(datacheck1.isna().sum())
```

```
REGISTRATION_DATE                           0
REGISTRATION_COUNTRY                        0
PURCHASE_COUNT                              0
PURCHASE_COUNT_DELIVERY                  9955
PURCHASE_COUNT_TAKEAWAY                  9955
FIRST_PURCHASE_DAY                       9955
LAST_PURCHASE_DAY                        9955
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                           72
IOS_PURCHASES                            9955
WEB_PURCHASES                            9955
ANDROID_PURCHASES                        9955
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               9955
MEDIAN_DAYS_BETWEEN_PURCHASES            9955
AVERAGE_DELIVERY_DISTANCE_KMS            9955
preferred_American_restaurant               0
preferred_Japanese_restaurant               0
preferred_Italian_restaurant                0
preferred_Mexican_restaurant                0
preferred_Indian_restaurant                 0
preferred_Middleeastern_restaurant          0
preferred_Korean_restaurant                 0
preferred_Thai_restaurant                   0
preferred_Vietnamese_restaurant             0
preferred_Hawaiian_restaurant               0
preferred_Greek_restaurant                  0
preferred_Spanish_restaurant                0
preferred_Nepalese_restaurant               0
preferred_Chinese_restaurant                0
GENERAL_MERCHANDISE                         0
GROCERY                                     0
PET_SUPPLIES                                0
RESTAURANT                                  0
RETAIL_STORE                                0
USER_TYPE                                   0
dtype: int64
```

Comments:

-User ID, Registration Date, Registation Country, Purchase Count, Preferred devide, User has valid payement, Purchase count by valid type are completely populated.
-We have very less data for preferred restaurant type. /n -Rest of the columns are mostly missing 9955 values.
-There are no duplicates

- I do think that there are alot of redundancy with respect to the columns (features) like min and max purchase euros, lunch, dinner, breakfast purchases and alot more. It can be derived out of some other columns as well. I still kept the column as of now just in case it makes anything easier later. But can be avoided to save some space.

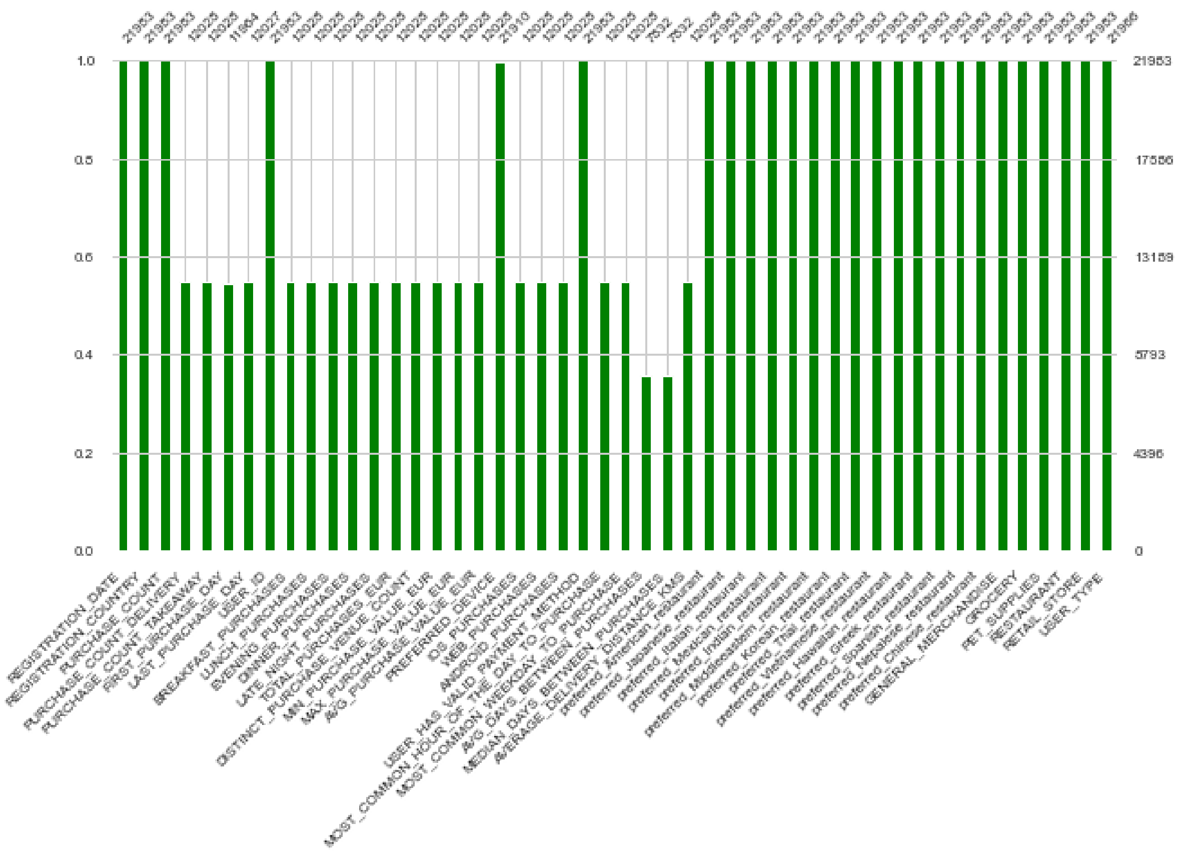
How do we handle it ?

In [627…

```
# Identiy the columns with non- missing values
msno.bar(cus_data_raw,color='#008000',figsize=(10,5), fontsize=8)
```

Out[627]:

```
<AxesSubplot:>
```



If you observe the data there are PURCHASE_COUNT =0 and they have only user id and preferred device data is populated mostly. All the above columns where 9955 rows missing values belong to these rows(by observation) This could mean the user has installed the app and not made any order. These customers anyway are a different target. let us consider this segment is already at our hand. To attract potential customers to make a purchase, offering incentives such as discounts, free gifts, or free DELIVERY on their first purchase can help them save money and try OUR SERVICE, thereby get used to the comfort. On account of the above explanation, we an get rid of the users whose purchse count is zero for now, and treat them as a seperate segment

In [628…

```
cus_data=cus_data_raw[cus_data_raw['PURCHASE_COUNT']>0]
len(cus_data)
```

Out[628]:

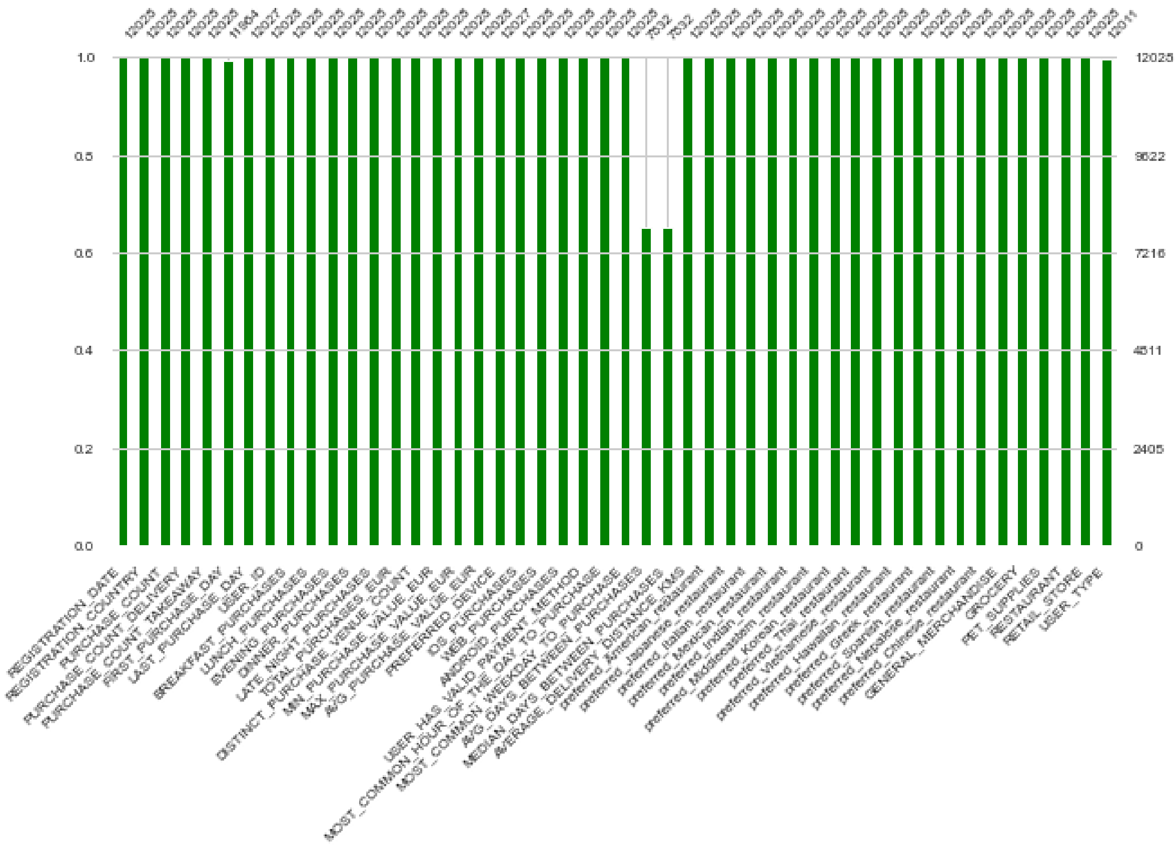
```
12028
```

In [629…

```
# Identiy the columns with non- missing values
msno.bar(cus_data,color='#008000',figsize=(10,5), fontsize=8)
```

Out[629]:

```
<AxesSubplot:>
```



In [630…
```
#the columns avg days between purchases and median days between purchases are null
#and both the columns are missing on the same rows
cus_data[cus_data['AVG_DAYS_BETWEEN_PURCHASES'].isnull()].equals(cus_data[cus_data[
```
Out[630]:
```
True
```
In [631…
```
#It could be null because there may be only one purchase.Check the difference betwe
cus_data['total_days']= pd.to_datetime(cus_data['LAST_PURCHASE_DAY'])- pd.to_dateti
cus_data['TOTAL_DAYS_BETWEEN_PURCHASES']=cus_data['total_days'].dt.days
cus_data.drop(columns=['total_days'],inplace=True)
```
In [632…
```
data_check1=cus_data[(cus_data['AVG_DAYS_BETWEEN_PURCHASES'].isnull()) & (cus_data[
len(data_check1)
```
Out[632]:
```
4196
```
In [633…
```
data_check1['TOTAL_DAYS_BETWEEN_PURCHASES'].value_counts()
```
Out[633]:
```
0.0      4150
77.0        1
27.0        1
370.0       1
360.0       1
28.0        1
39.0        1
37.0        1
211.0       1
63.0        1
Name: TOTAL_DAYS_BETWEEN_PURCHASES, dtype: int64
```
In [634…
```
data_check1['PURCHASE_COUNT'].value_counts()
```
Out[634]:
```
1    4179
2      17
Name: PURCHASE_COUNT, dtype: int64
```

This says that most of the null rows in avg days between purchases and median days between purchases are from rows with purchase count = 1. We can populate the AVG_DAYS_BETWEEN_PURCHASES and MEDIAN_DAYS_BETWEEN_PURCHASES 4179 of rows with purchase count 1 with 0 We can get rid of the 17 rows with purchase count 2 and having null values in the days between purchases or keep them (<1% of data) as it might not add much value Hence, It is safe to induce 0 into the cells where AVG_DAYS_BETWEEN_PURCHASES and MEDIAN_DAYS_BETWEEN_PURCHASES is null

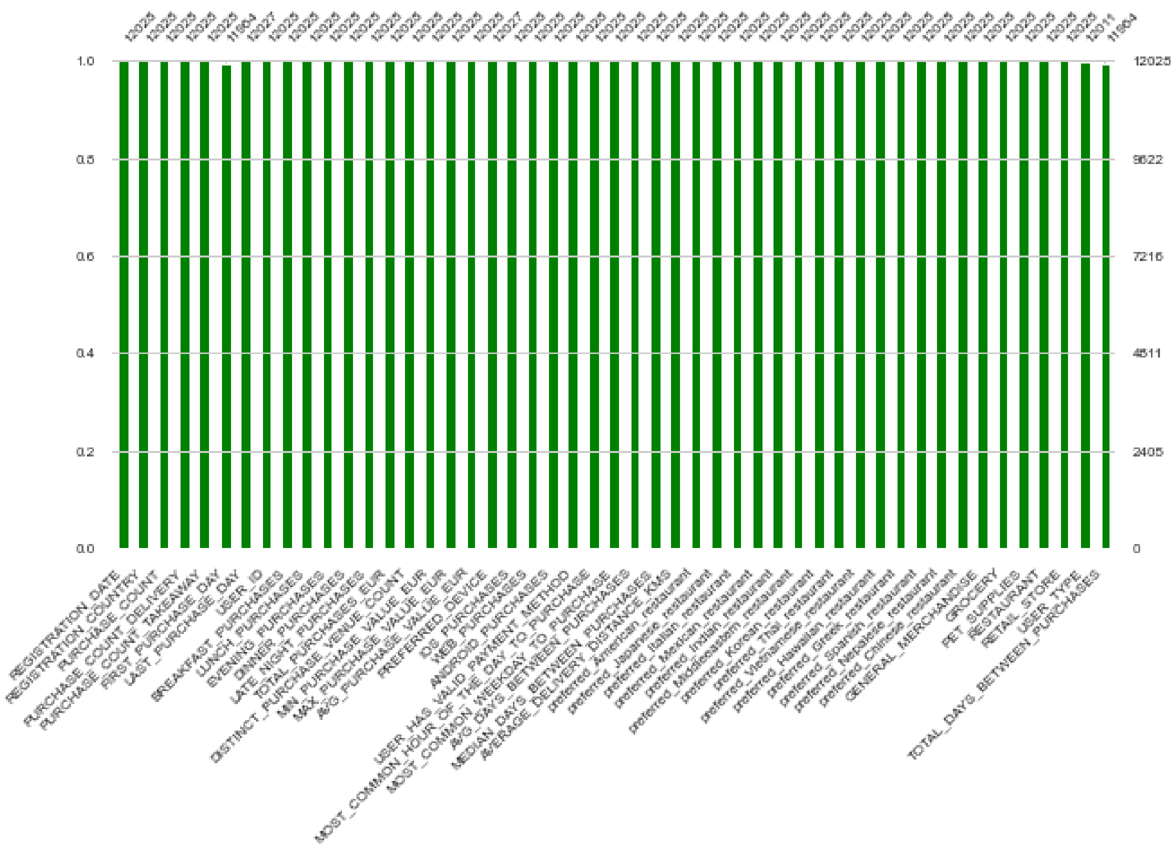
In [635…
```
cus_data['AVG_DAYS_BETWEEN_PURCHASES'].fillna(0, inplace=True)
cus_data['MEDIAN_DAYS_BETWEEN_PURCHASES'].fillna(0, inplace=True)
```

In [636…
```
len(cus_data)
```

Out[636]:
```
12028
```

In [637…
```
msno.bar(cus_data,color='#008000',figsize=(10,5), fontsize=8)
```

Out[637]:
```
<AxesSubplot:>
```



In [638…
```
len(cus_data)
```

Out[638]:
```
12028
```

Exploratory Data Analysis

In [639…
```
cus_data_raw.USER_TYPE.value_counts(normalize=True).mul(100).round(0)
```

Out[639]:
```
Inactive users                 45.0
Churning users                 21.0
First-time or one time users   19.0
Active users                   15.0
Name: USER_TYPE, dtype: float64
```

In [640…
```
fig = px.pie(cus_data_raw.USER_TYPE.value_counts(), values='USER_TYPE',
             names=['Inactive users','Churning users','First-time users','Active us
             title="User's device preferences",width=700, height=400)
fig.update_traces(textposition='inside', textinfo='percent+label')
fig.show()
```

There are alot of inactive users 45%. We really need to get lure them into a first time experience. There is a lot of potential in this area

In [641…
```
cus_data.describe()     #ignore user id
```

Out[641]:

	PURCHASE_COUNT	PURCHASE_COUNT_DELIVERY	PURCHASE_COUNT_TAKEAWAY	USER_
count	12028.000000	12028.000000	12028.000000	12028.000(
mean	6.114150	5.741686	0.372464	11036.1339
std	10.763064	10.536220	1.416310	6383.387
min	1.000000	0.000000	0.000000	2.000(
25%	1.000000	1.000000	0.000000	5529.750(
50%	3.000000	2.000000	0.000000	11038.000(
75%	6.000000	6.000000	0.000000	16520.250(
max	320.000000	320.000000	44.000000	21983.000(

8 rows × 28 columns

Average Purchase count is ~6 per person. Mean POV : Average delivery is ~5 which is a large part and a very less Take away deliveries

PERSPECTIVE - PREFERRED DEVICE

In [642…

```
cus_data.groupby(['PREFERRED_DEVICE']).agg({
    'USER_ID':'count',
    'PURCHASE_COUNT': 'sum',
    'TOTAL_PURCHASES_EUR': 'sum',
    'IOS_PURCHASES': 'sum',
    'WEB_PURCHASES': 'sum',
    'ANDROID_PURCHASES': 'sum'
})
```

Out[642]:

PREFERRED_DEVICE	USER_ID	PURCHASE_COUNT	TOTAL_PURCHASES_EUR	IOS_PURCHASES	WEB_I
android	4108	24884	646563.764	934.0	
ios	5328	31716	932438.584	30019.0	
web	2591	16940	540433.300	3934.0	

In [643…

```
cus_data.PREFERRED_DEVICE.value_counts(normalize=1).mul(100).round(0)
```

Out[643]:

```
ios        44.0
android    34.0
web        22.0
Name: PREFERRED_DEVICE, dtype: float64
```

In [644…

```
fig = px.pie(cus_data_raw.PREFERRED_DEVICE.value_counts(), values='PREFERRED_DEVICE
             title="User's device preferences",width=700, height=300)
fig.update_traces(textposition='inside', textinfo='percent+label')
fig.show()
```

In [645…

```
cus_data_raw.groupby(['PREFERRED_DEVICE','USER_HAS_VALID_PAYMENT_METHOD']).size()
```

Out[645]:

```
PREFERRED_DEVICE  USER_HAS_VALID_PAYMENT_METHOD
android           False                            6597
                  True                             1851
ios               False                            6543
                  True                             3204
web               False                            1204
                  True                             2511
dtype: int64
```

In [646…

```
#PASSIVE USERS BASED ON DEVICE - TO MAKE NECESSARY IMPROVEMENTS
cus_data_raw.groupby(['PREFERRED_DEVICE','USER_TYPE']).size()
```

Out[646]:

```
PREFERRED_DEVICE  USER_TYPE
android           Active users                   1163
                  Churning users                 1517
                  First-time or one time users   1424
                  Inactive users                 4340
ios               Active users                   1414
                  Churning users                 2089
                  First-time or one time users   1817
                  Inactive users                 4419
web               Active users                    631
                  Churning users                 1018
                  First-time or one time users    937
                  Inactive users                 1124
dtype: int64
```

INSIGHT: Wolt app orders predominantly stem from three primary devices: ~44% from iOS, ~39% from Android, and ~17% from the website. Given the prevalent use of mobile phones for food ordering due to their convenience, it is advisable for Wolt to customize user acquisition campaigns and banner designs to align with various phone sizes. Specially, preference could be given to iphone features, if any.

PERSPECTIVE - COUNTRY

In [647…

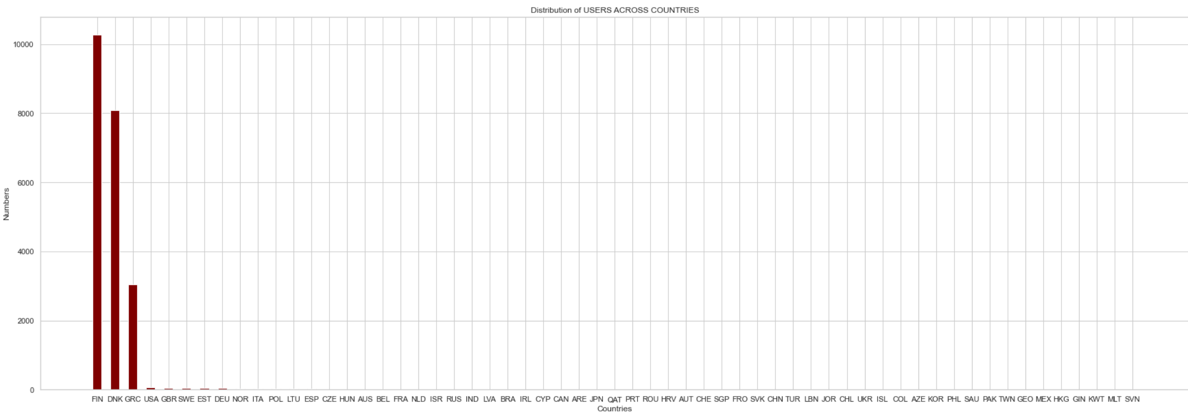
```
cus_data.REGISTRATION_COUNTRY.value_counts(normalize=1).mul(100).round(0)
```

Out[647]:

```
FIN    45.0
DNK    41.0
GRC    13.0
NOR     0.0
EST     0.0
HUN     0.0
CZE     0.0
SWE     0.0
POL     0.0
ISR     0.0
LVA     0.0
GBR     0.0
FRA     0.0
LTU     0.0
CAN     0.0
DEU     0.0
HRV     0.0
CYP     0.0
ARE     0.0
Name: REGISTRATION_COUNTRY, dtype: float64
```

In [648…

```python
countries= cus_data_raw.REGISTRATION_COUNTRY.value_counts().rename_axis('Countries'
fig = plt.figure(figsize =(30, 10))
plt.bar(countries['Countries'], countries['Numbers'], color ='maroon', width = 0.5)
plt.xlabel('Countries'),
plt.ylabel('Numbers')
plt.title('Distribution of USERS ACROSS COUNTRIES')
plt.show()
```



In [649…

```python
cus_data.describe()
```

Out[649]:

	PURCHASE_COUNT	PURCHASE_COUNT_DELIVERY	PURCHASE_COUNT_TAKEAWAY	USER_
count	12028.000000	12028.000000	12028.000000	12028.000(
mean	6.114150	5.741686	0.372464	11036.1339
std	10.763064	10.536220	1.416310	6383.387
min	1.000000	0.000000	0.000000	2.000(
25%	1.000000	1.000000	0.000000	5529.7500
50%	3.000000	2.000000	0.000000	11038.000(
75%	6.000000	6.000000	0.000000	16520.2500
max	320.000000	320.000000	44.000000	21983.000(

8 rows × 28 columns

In [650…

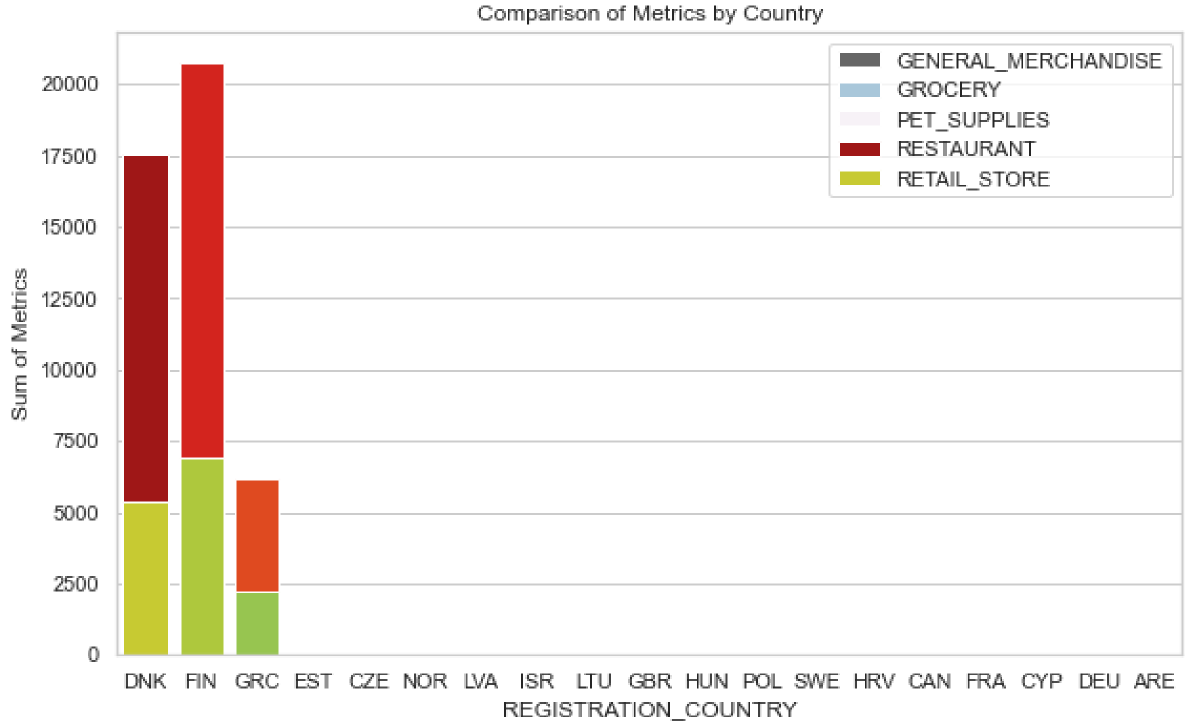
```
grouped_data=cus_data.groupby(['REGISTRATION_COUNTRY']).agg({
    'GENERAL_MERCHANDISE': 'sum',
    'GROCERY': 'sum',
    'PET_SUPPLIES': 'sum',
    'RESTAURANT': 'sum',
    'RETAIL_STORE': 'sum',
    'PURCHASE_COUNT':'sum',
    'TOTAL_PURCHASES_EUR': 'sum'        # measure for total purchase in EUR
}).sort_values(by="TOTAL_PURCHASES_EUR", ascending=False).reset_index()
grouped_data.head()
```

Out[650]:

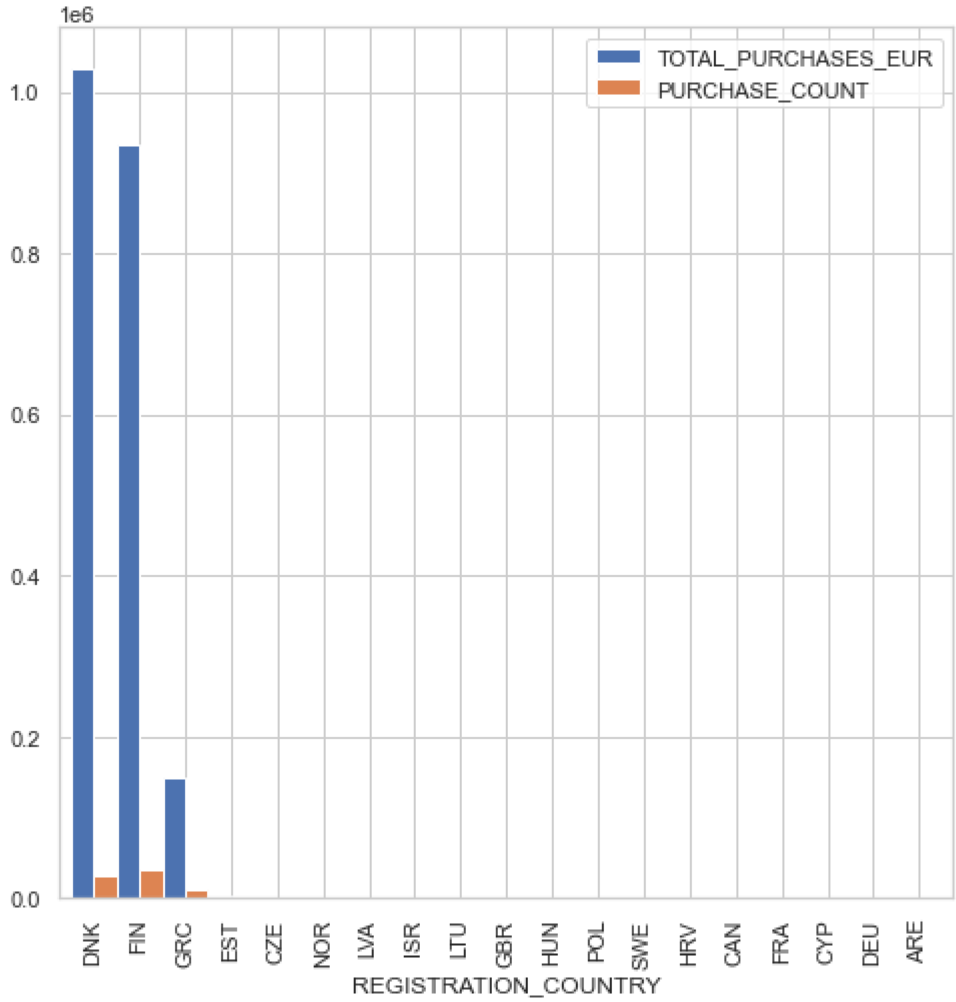
	REGISTRATION_COUNTRY	GENERAL_MERCHANDISE	GROCERY	PET_SUPPLIES	RESTAURANT	RI
0	DNK	589	4683	61	17531	
1	FIN	687	6184	84	20784	
2	GRC	160	1782	28	6182	
3	EST	3	13	0	49	
4	CZE	0	17	1	25	

In [651…

```
# Plotting using seaborn
sns.set(style="whitegrid")
plt.figure(figsize=(10, 6))

metrics = ['GENERAL_MERCHANDISE','GROCERY','PET_SUPPLIES','RESTAURANT','RETAIL_STOR

# Set a custom color palette for each metric
color_palette = {'GENERAL_MERCHANDISE':'Accent_r','GROCERY':'Blues_d','PET_SUPPLIES

for metric in metrics:
    sns.barplot(x='REGISTRATION_COUNTRY', y=metric, data=grouped_data, label=metric

plt.title('Comparison of Metrics by Country')
plt.ylabel('Sum of Metrics')
plt.legend()
plt.show()
```




```
In [652…  grouped_data.plot(x="REGISTRATION_COUNTRY", y=["TOTAL_PURCHASES_EUR","PURCHASE_COUN

          # print bar graph
          plt.show()
```




```
In [653…  #COUNTRY WISE PASSIVE USERS TO TAKE NECCESSARY ACTION
          cus_data_top3cntry=cus_data_raw[cus_data_raw['REGISTRATION_COUNTRY'].isin(['FIN','D
          grouped =cus_data_top3cntry.groupby(['REGISTRATION_COUNTRY','USER_TYPE']).size()
```

In [654…

```
percentage = round(grouped / len(cus_data_top3cntry) * 100)
percentage
```

Out[654]:

```
REGISTRATION_COUNTRY  USER_TYPE
DNK                   Active users                   6.0
                      Churning users                10.0
                      First-time or one time users   8.0
                      Inactive users                14.0
FIN                   Active users                   7.0
                      Churning users                10.0
                      First-time or one time users   9.0
                      Inactive users                23.0
GRC                   Active users                   2.0
                      Churning users                 2.0
                      First-time or one time users   3.0
                      Inactive users                 7.0
dtype: float64
```

In [655…

```
cus_data_top3cntry.groupby(['REGISTRATION_COUNTRY']).agg({
    'DISTINCT_PURCHASE_VENUE_COUNT': 'mean',
    'BREAKFAST_PURCHASES': 'sum',
    'LUNCH_PURCHASES': 'sum',
    'EVENING_PURCHASES': 'sum',
    'DINNER_PURCHASES': 'sum',
    'LATE_NIGHT_PURCHASES': 'sum' }).reset_index()
```

Out[655]:

	REGISTRATION_COUNTRY	DISTINCT_PURCHASE_VENUE_COUNT	BREAKFAST_PURCHASES	LUNCH
0	DNK	3.435507	327.0	
1	FIN	3.106769	1264.0	
2	GRC	3.697781	735.0	

INSIGHT: Most of the business 99% are highly located in Finland, Denmark, Greece
Restaurant segment gets the highest business across all the countries
followed by Retail Stores and then Groceries
Interestingly, though Finland leads in no. of orders, Denmark leads in terms of purchase values i.e.in terms of Euros(Monetary)
Denmark users mostly order Dinner where as Finland and Greece users order Lunch slightly more than dinner

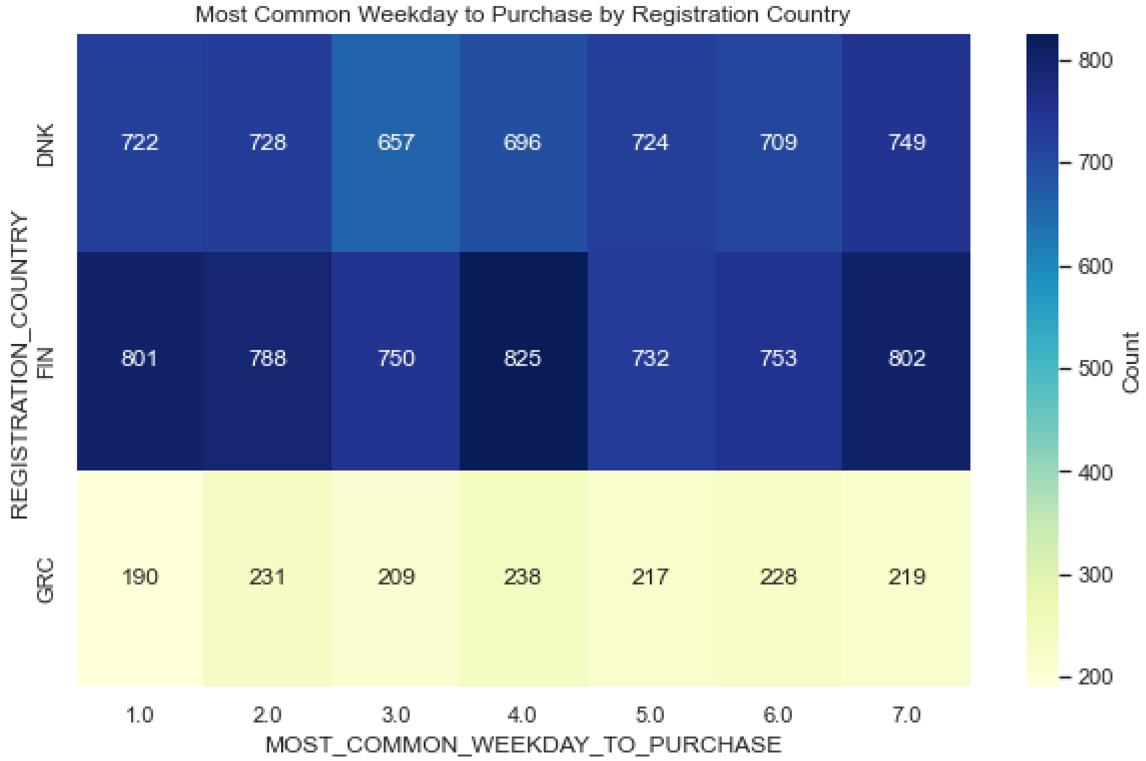
MOST_COMMON_WEEKDAY_TO_PURCHASE PATTERN

In [656…

```
cus_data.MOST_COMMON_WEEKDAY_TO_PURCHASE.value_counts(normalize=1).mul(100).round(
# Group by two columns and calculate the size
grouped = cus_data_top3cntry.groupby(['REGISTRATION_COUNTRY', 'MOST_COMMON_WEEKDAY_
```

In [657…

```
# Create a pivot table for heatmap
heatmap_data = grouped.pivot('REGISTRATION_COUNTRY', 'MOST_COMMON_WEEKDAY_TO_PURCHA

# Plotting the heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(heatmap_data, annot=True, cmap='YlGnBu', fmt='g', cbar_kws={'label': '
plt.title('Most Common Weekday to Purchase by Registration Country')
plt.show()
```



There is no significant demarcation here. Tuesday is comparatively less preferred by people for ordering

MOST_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE PATTERN

In [658…

```
# Creating new "USER_TYPE" column with the following conditions:
cus_data.loc[(cus_data.MOST_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE >= 0)
             & (cus_data.MOST_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE < 5), 'time'] = "M
cus_data.loc[(cus_data.MOST_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE >= 5)
             & (cus_data.MOST_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE < 11), 'time'] = '
cus_data.loc[(cus_data.MOST_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE >= 11)
             & (cus_data.MOST_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE < 16), 'time'] = '
cus_data.loc[(cus_data.MOST_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE >= 16)
             & (cus_data.MOST_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE <= 23), 'time'] =
```

In [659…

```
cus_data.time.value_counts(normalize=1).mul(100).round(0)
```

Out[659]:

```
Night        34.0
Morning      25.0
Mid_Night    21.0
Lunch        21.0
Name: time, dtype: float64
```

In [660…

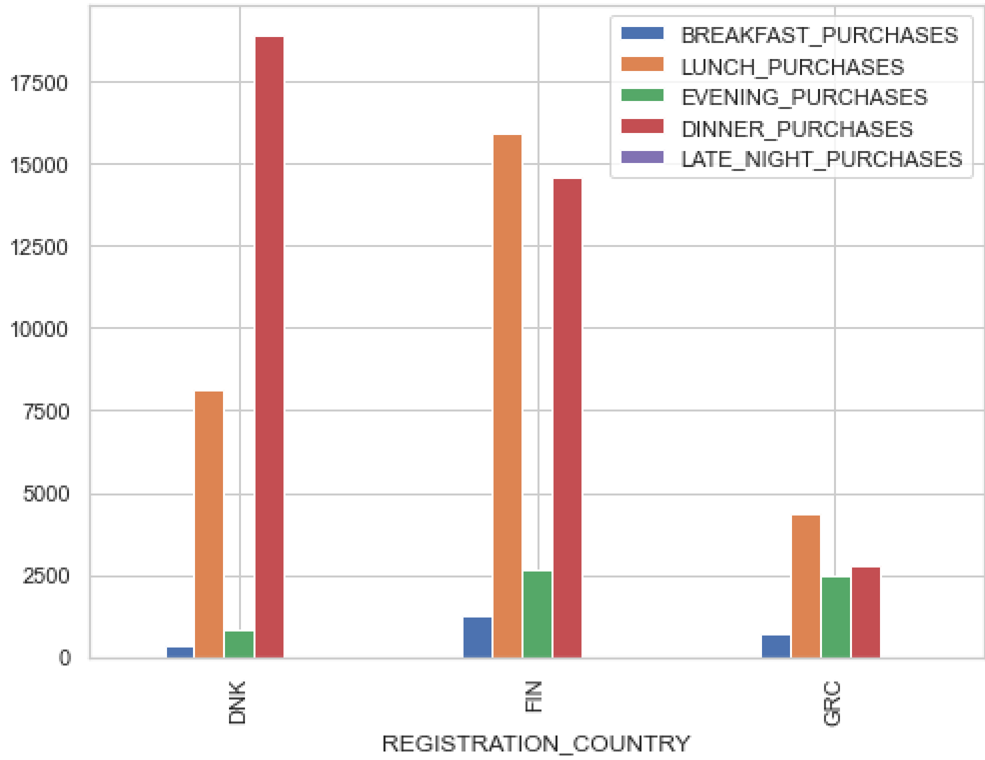
```
a=  cus_data_top3cntry.groupby(['REGISTRATION_COUNTRY']).agg({
'BREAKFAST_PURCHASES': 'sum',
'LUNCH_PURCHASES': 'sum',
'EVENING_PURCHASES': 'sum',
'DINNER_PURCHASES': 'sum',
'LATE_NIGHT_PURCHASES': 'sum'   }).reset_index()
a.head()
```

Out[660]:

	REGISTRATION_COUNTRY	BREAKFAST_PURCHASES	LUNCH_PURCHASES	EVENING_PURCHASES
0	DNK	327.0	8157.0	812.0
1	FIN	1264.0	15915.0	2635.0
2	GRC	735.0	4365.0	2449.0

In [661…

```
a.plot(x="REGISTRATION_COUNTRY", y=["BREAKFAST_PURCHASES","LUNCH_PURCHASES","EVENIN

# print bar graph
plt.show()
```



INSIGHT: Most of the orders come at night between 4 PM to 11PM

Denmark users mostly order Dinner where as Finland and Greece users order Lunch slightly more than dinner

PERSPECTIVE - USER AND PURCHASE COUNT

In [662…

```
cus_data_raw.PURCHASE_COUNT.value_counts()
```

Out[662]:

```
0      9955
1      4179
2      1821
3      1148
4       835
       ...
205       1
85        1
132       1
106       1
71        1
Name: PURCHASE_COUNT, Length: 107, dtype: int64
```

In [663…

```
cus_data_raw.PURCHASE_COUNT.value_counts(normalize=1).mul(100).round(0)
```

Out[663]:
```
0      45.0
1      19.0
2       8.0
3       5.0
4       4.0
       ...
205     0.0
85      0.0
132     0.0
106     0.0
71      0.0
Name: PURCHASE_COUNT, Length: 107, dtype: float64
```

In [664…
```
cus_data.head()
```

Out[664]:

	REGISTRATION_DATE	REGISTRATION_COUNTRY	PURCHASE_COUNT	PURCHASE_COUNT_DELIVE
1	2019-09-01 00:00:00.000	FIN	1	
2	2019-09-01 00:00:00.000	DNK	19	1
7	2019-09-01 00:00:00.000	FIN	1	
12	2019-09-01 00:00:00.000	FIN	19	1
13	2019-09-01 00:00:00.000	FIN	2	[illegible]

5 rows × 50 columns

In [665…
```
cus_data.groupby(['USER_TYPE']).agg({
    'USER_ID': 'count',
    'PURCHASE_COUNT': 'sum',
    'TOTAL_PURCHASES_EUR': 'sum',
    'IOS_PURCHASES' :'sum',
    'WEB_PURCHASES': 'sum',
    'ANDROID_PURCHASES':'sum',
    'PURCHASE_COUNT_DELIVERY':'sum',
    'PURCHASE_COUNT_TAKEAWAY':'sum'
})
```

Out[665]:

USER_TYPE	USER_ID	PURCHASE_COUNT	TOTAL_PURCHASES_EUR	IOS_PURCHASES	WEB_PURCHA
Active users	3208	47031	1270162.212	22008.0	78
Churning users	4624	22297	714307.044	11066.0	38
First-time or one time users	4179	4179	133548.580	1796.0	9

In [666…

```
### We just calculate based on the users who ordered the services at least one time
total_sales = cus_data['TOTAL_PURCHASES_EUR'].sum()
total_orders = cus_data['PURCHASE_COUNT'].sum()
AOV = total_sales/total_orders
PF = total_orders/len(cus_data[cus_data.PURCHASE_COUNT >0])
print(f"Avg Order Value: {AOV:0.1f} euros and Purchase Frequency: {PF:0.1f} purchas
```

```
Avg Order Value: 28.8 euros and Purchase Frequency: 6.1 purchase times
```

Comments: There are 04 groups of customers at glance to be analysed & assessed:

1. Non-users. [9955 - 45%] - Users who installed the app and never made orders
2. Once-users. [4179 - 19%] - Users whose purchase count was one and never came back
3. Repeated-users (Pareto principle: 80% sales figures coming from 20% of this customer group) [7849 - 37%]
   Delivery is highly likely than take away in all kind of user type

IMMEDIATE USERS </SPAN>

In [667…

```
###
registers = cus_data.groupby(['REGISTRATION_DATE'])['USER_ID'].count().reset_index(
registers.rename(columns = {'USER_ID' : 'registers'}, inplace = True)

## Calculate the users who have registered & immediately used the service
users = cus_data[(cus_data.PURCHASE_COUNT > 0) & (cus_data['REGISTRATION_DATE']
                                                   == cus_data['FIRST_PURCHASE_DAY']
users.rename(columns = {'USER_ID' : 'immediate_users'}, inplace = True)

group_registers = pd.merge(registers, users, on='REGISTRATION_DATE', how='left')
group_registers['%immediate_users'] = (group_registers['immediate_users'] / group_r
###
#group_registers = str_to_date(group_registers, col = 'REGISTRATION_DATE' )
group_registers.head()
```

Out[667]:

	REGISTRATION_DATE	registers	immediate_users	%immediate_users
0	2019-09-01 00:00:00.000	527	250	47.438330
1	2019-09-02 00:00:00.000	280	129	46.071429
2	2019-09-03 00:00:00.000	221	108	48.868778
3	2019-09-04 00:00:00.000	318	162	50.943396
4	2019-09-05 00:00:00.000	446	222	49.775785

In [668…

```
group_registers['%immediate_users'].mean()

print(f"Avg percentage of registers immediately using the Wolt services: {group_reg
```

```
Avg percentage of registers immediately using the Wolt services: 47.90%.
```

RESTAURANT TYPE </SPAN>

In [669…

```
def generate_value_counts_for_restaurant_style_preferance(df,column_list):
    pieces=[]
    for col in column_list:
        tmp_series = df[col].value_counts()
```

```
        tmp_series.name = col
        pieces.append(tmp_series)
    df_value_counts = pd.concat(pieces, axis=1)
    df_value_counts = df_value_counts .rename_axis('yes_I_prefer')
    return df_value_counts
```

In [670…

```
preferred_restaurants=["preferred_American_restaurant", "preferred_Italian_restaura
                                "preferred_Indian_restaurant","preferred_Middleeast
                                 ,"preferred_Vietnamese_restaurant","preferred_Hawa
                                 "preferred_Nepalese_restaurant","preferred_Chinese

value_counts_res = generate_value_counts_for_restaurant_style_preferance(cus_data_r
#value_counts_res = value_counts_res.rename_axis('yes_I_prefer')

value_counts_res
```

Out[670]:

	preferred_American_restaurant	preferred_Italian_restaurant	preferred_Japanese_restaura
yes_I_prefer			
False	20672	21054	213
True	1311	929	6

In [671…

```
#general
#restaurant preference
#User type angle
#country type angle
fig = px.bar(value_counts_res.iloc[1:, :],  y=["preferred_American_restaurant", "pr
                                "preferred_Indian_restaurant","preferred_Middleeast
                                 ,"preferred_Vietnamese_restaurant","preferred_Hawa
                                 "preferred_Nepalese_restaurant","preferred_Chinese
fig.update_layout(barmode='group')
fig.show()
```

People mostly prefer American Restuarants followed by Italian and then Japanese.
Spanish and Nepalese Restaurants are very less preferred by people

RFM Analysis

Customer Value RFM Model: We aim to initiate a straightforward analysis.
To achieve this, we will employ the RFM (Recency, Frequency, and Monetary Value) model for customer segmentation.
The RFM model involves assessing each customer's transactions to derive three key attributes:
Recency: Indicates the time elapsed since a customer's most recent purchase.
Frequency: Measures the regularity of a customer's transactions.
Monetary Value: Quantifies the total monetary worth of all customer transactions.
This approach allows us to distill complex customer data into actionable insights, providing a foundation for targeted strategies based on customer behavior.

In [672…

```
def return_date(df, col):

    df[col]= pd.to_datetime(df[col])
    df[col] = df[col].apply(lambda x: x.date()) # return a column for certain date

    return df
customers = return_date(cus_data, col = 'LAST_PURCHASE_DAY')
```

In [673…

```
cus_data['days_last_PO'] = dt.date(2020,10,31) - cus_data['LAST_PURCHASE_DAY']
customers['days_last_PO'] = customers['days_last_PO'].apply(lambda x: x.days)
len(cus_data)
```

Out[673]:

```
12028
```

In [674…

```
cus_RFM =cus_data[['USER_ID','days_last_PO', 'PURCHASE_COUNT', 'TOTAL_PURCHASES_EUR
                   , 'AVG_PURCHASE_VALUE_EUR']]
cus_RFM.columns = ['user_id','recency', 'frequency',  'TOTAL_PURCHASES_EUR','moneta
```

In [675…

```
cus_RFM.head()
```

Out[675]:

	user_id	recency	frequency	TOTAL_PURCHASES_EUR	monetary
1	2	59.0	1	38.456	38.456
2	3	159.0	19	631.488	33.396
7	8	17.0	1	19.228	19.228
12	13	4.0	19	587.972	31.372
13	14	52.0	2	118.404	59.708

In [676…

```
for col in cus_RFM.columns:
    print(col)
```

```
user_id
recency
frequency
TOTAL_PURCHASES_EUR
monetary
```

In [677…

```
cus_RFM.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 12028 entries, 1 to 21982
Data columns (total 5 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   user_id              12028 non-null  int64  
 1   recency              12027 non-null  float64
 2   frequency            12028 non-null  int64  
 3   TOTAL_PURCHASES_EUR  12028 non-null  float64
 4   monetary             12028 non-null  float64
dtypes: float64(3), int64(2)
memory usage: 563.8 KB
```

In [678…

```
#top cstomers with respect to frequency  and see if they have gone out
quantiles = cus_RFM.quantile(q=[0.25,0.5,0.75])
quantiles= quantiles.to_dict()

# Function arguments (x = value, p = RECENCY, MONETARY, FREQUENCY, d = quartiles di
def RScoring(x,p,d):
    if x <= d[p][0.25]:
        return 4
    elif x <= d[p][0.50]:
        return 3
    elif x <= d[p][0.75]:
        return 2
    else:
        return 1
# Function arguments (x = value, p = RECENCY, MONETARY, FREQUENCY, d = quartiles di
```

```python
#to create Wolt RFM segments in FREQUENCY AND MONETARY
def FMScoring(x,p,d):
    if x <= d[p][0.25]:
        return 1
    elif x <= d[p][0.50]:
        return 2
    elif x <= d[p][0.75]:
        return 3
    else:
        return 4

# Creating a Wolt_RFMScores segmentation table so we can evaluate the analysis
Wolt_RFMSegment = cus_RFM.copy()
Wolt_RFMSegment['R_Score'] = Wolt_RFMSegment['recency'].apply(RScoring, args=('rece
Wolt_RFMSegment['F_Score'] = Wolt_RFMSegment['frequency'].apply(FMScoring, args=('f
Wolt_RFMSegment['M_Score'] = Wolt_RFMSegment['monetary'].apply(FMScoring, args=('mo

Wolt_RFMSegment.head()
```

Out[678]:

	user_id	recency	frequency	TOTAL_PURCHASES_EUR	monetary	R_Score	F_Score	M_Score
1	2	59.0	1	38.456	38.456	3	1	3
2	3	159.0	19	631.488	33.396	3	4	3
7	8	17.0	1	19.228	19.228	4	1	1
12	13	4.0	19	587.972	31.372	4	4	3
13	14	52.0	2	118.404	59.708	3	2	4

In [679…

```python
Wolt_RFMSegment[Wolt_RFMSegment['R_Score']==1].sort_values(by='TOTAL_PURCHASES_EUR'
```

Out[679]:

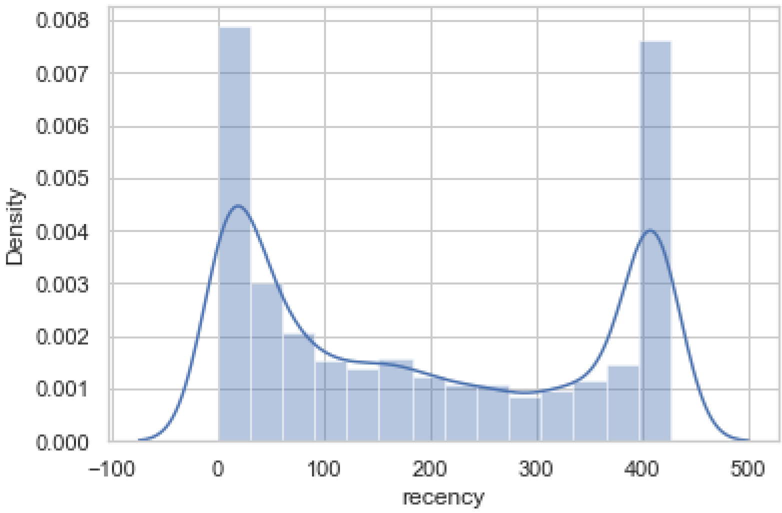
	user_id	recency	frequency	TOTAL_PURCHASES_EUR	monetary	R_Score	F_Score	M_Scor
3176	3177	417.0	1	657.800	657.800	1	1	4
2177	2178	401.0	23	575.828	25.300	1	4	2
862	863	402.0	3	475.640	158.884	1	2	4
14645	14646	387.0	6	384.560	63.756	1	3	4
904	905	402.0	3	380.512	126.500	1	2	4
11995	11996	397.0	4	340.032	85.008	1	3	4
19096	19097	396.0	1	338.008	338.008	1	1	4
11886	11887	408.0	2	329.912	164.956	1	2	4
1132	1133	425.0	1	297.528	297.528	1	1	4
17829	17830	392.0	3	284.372	95.128	1	2	4

As a non Data Scientist, I could evaluate different quantiles for recency, frequency and monetary values and bucket them into diffrent groups manually like High value but churned customers, Low value Less recent Customers and so on As a Data Scientist I would use Machine Learning Algorithm - K Means Clustering since it is much more scalable and Interpretable

Univariate Analysis

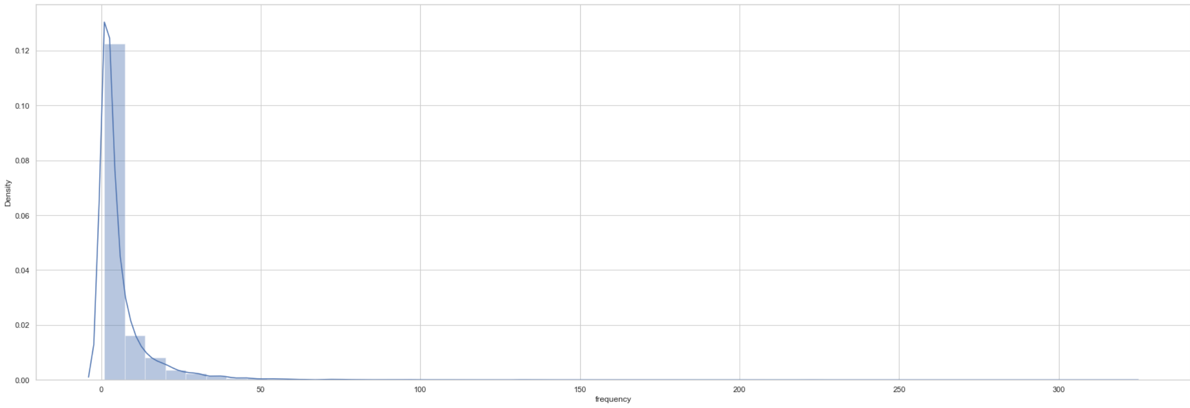
In [680…

```
x=cus_RFM['recency']
ax=sns.distplot(x)
```



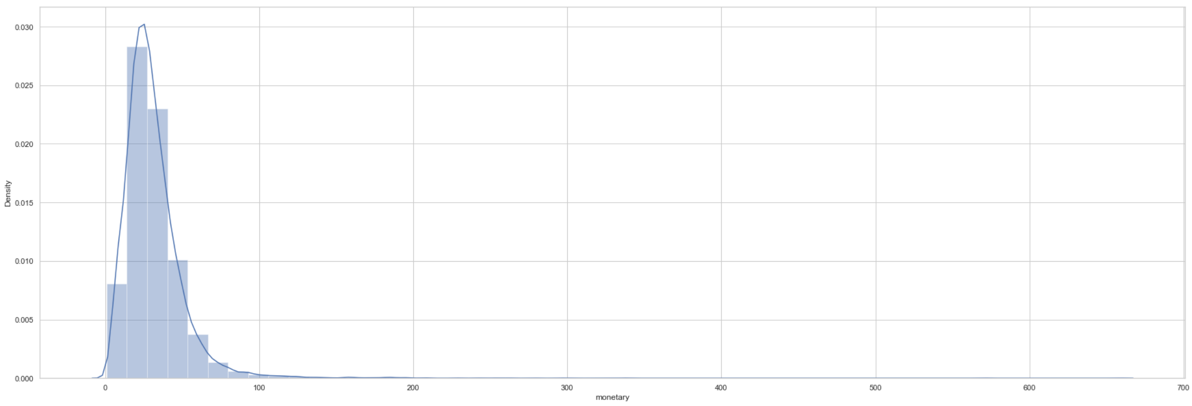
In [681…

```
fig = plt.figure(figsize =(30, 10))
x=cus_RFM['frequency']
ax=sns.distplot(x)
```



In [682…

```
fig = plt.figure(figsize =(30, 10))
x=cus_RFM['monetary']
ax=sns.distplot(x)
```

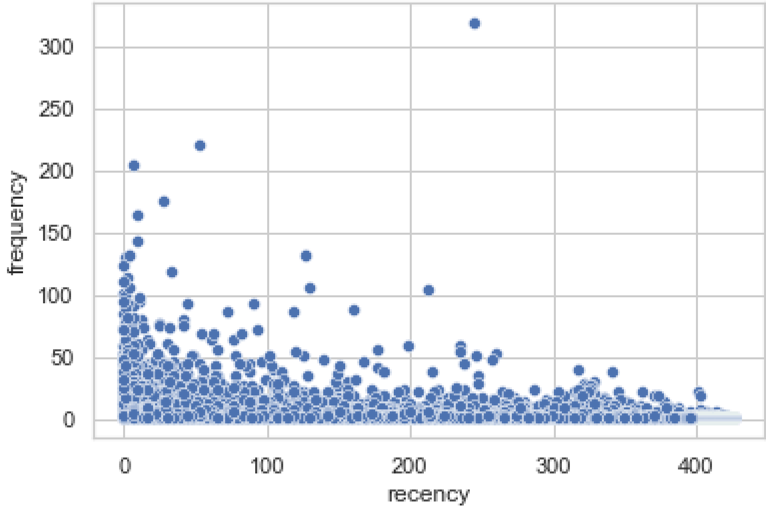


Bivariate Analysis

In [683…

```
sns.scatterplot(data=cus_RFM, x='recency',y='frequency' )
```

Out[683]: <AxesSubplot:xlabel='recency', ylabel='frequency'>



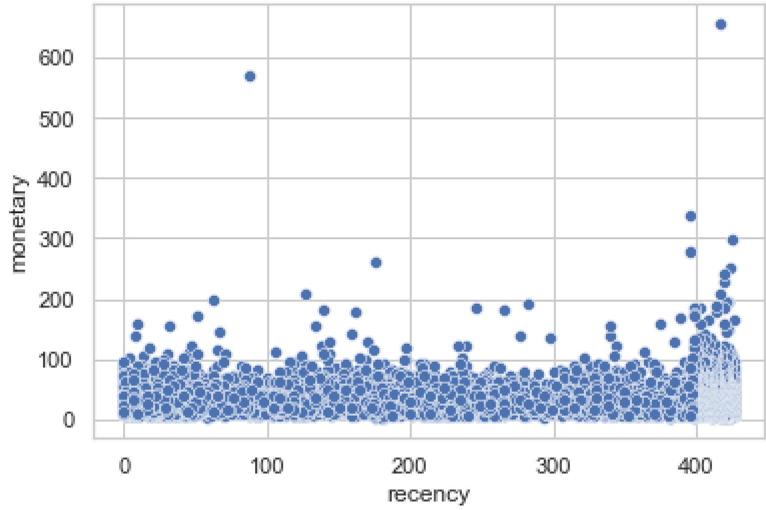
In [684…
```
cus_RFM[cus_RFM['frequency'] > 300]          #We might have lost this Golden Customer
```

Out[684]:

	user_id	recency	frequency	TOTAL_PURCHASES_EUR	monetary
79	80	245.0	320	4335.408	13.156

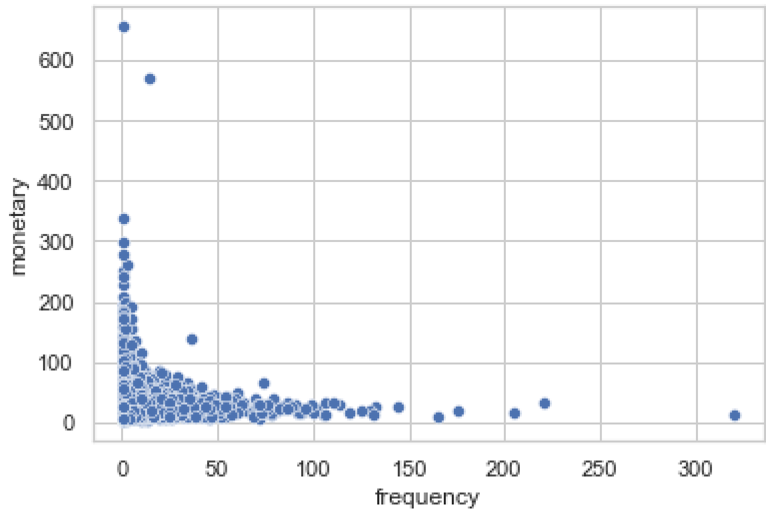
In [685…
```
sns.scatterplot(data=cus_RFM, x='recency',y='monetary' )
```

Out[685]: <AxesSubplot:xlabel='recency', ylabel='monetary'>



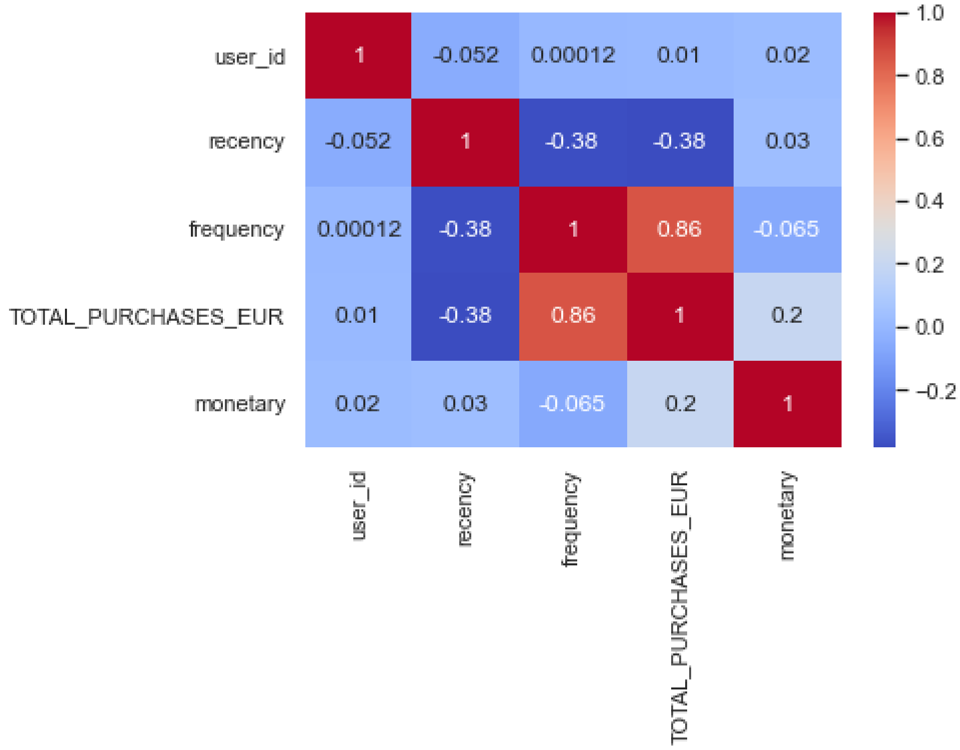
In [686…
```
sns.scatterplot(data=cus_RFM, x='frequency',y='monetary' )
```

Out[686]: <AxesSubplot:xlabel='frequency', ylabel='monetary'>



In [687…
```
sns.heatmap( cus_RFM.corr(),annot=True,cmap='coolwarm')
```

Out[687]:
```
<AxesSubplot:>
```



Frequency and Total Purchase Euros are highly corelated. Hence to avoid reduendancy we shall drop Total purchase Euros and rather consider Avg purchase value as a monetary parameter

K Means Clustering

In [688…
```
from sklearn.preprocessing import StandardScaler
# create new dataframe with transformed values
df_t =  cus_RFM.copy()

ss = StandardScaler()
df_t['recency_t'] = ss.fit_transform( cus_RFM['recency'].values.reshape(-1,1))
df_t['frequency_t'] = ss.fit_transform( cus_RFM['frequency'].values.reshape(-1,1))
df_t['monetary_t'] = ss.fit_transform( cus_RFM['monetary'].values.reshape(-1,1))
#df_t['avg_monetary_t'] = ss.fit_transform( cus_RFM['TOTAL_PURCHASES_EUR'].values.r
```

In [689…
```
df_t.info(10)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 12028 entries, 1 to 21982
Data columns (total 8 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   user_id              12028 non-null  int64  
 1   recency              12027 non-null  float64
 2   frequency            12028 non-null  int64  
 3   TOTAL_PURCHASES_EUR  12028 non-null  float64
 4   monetary             12028 non-null  float64
 5   recency_t            12027 non-null  float64
 6   frequency_t          12028 non-null  float64
 7   monetary_t           12028 non-null  float64
dtypes: float64(6), int64(2)
memory usage: 845.7 KB
```

In [690…
```
df_t.to_csv("C:\\Users\\Namana\\OneDrive\\Desktop\\Projects\\Customer_Segmentation\
```

In [691…
```
# Get rid of the row where recency is null
df_t=df_t[df_t['recency'].notnull()].reset_index()
```

In [692…
```
len(df_t)
```

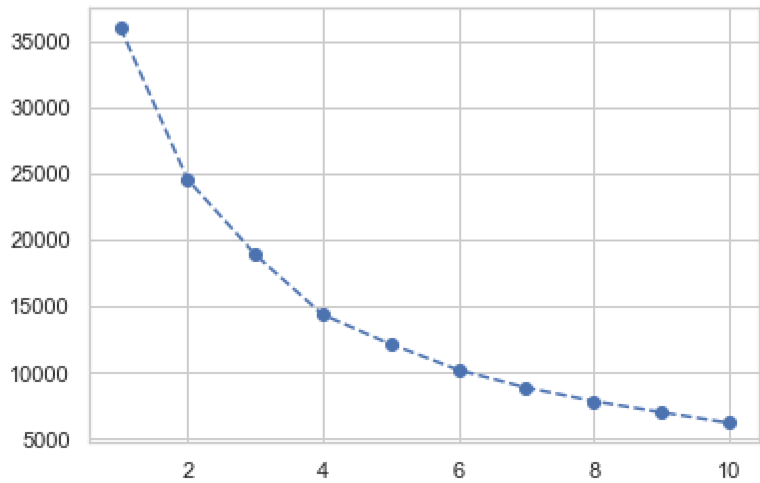
Out[692]:
```
12027
```

In [693…
```
from sklearn.cluster import KMeans
```

Elbow method to find the K value

In [694…
```
intertia_scores=[]
for i in range(1,11):
    kmeans=KMeans(n_clusters=i)
    kmeans.fit(df_t[['recency_t','frequency_t','monetary_t']])
    intertia_scores.append(kmeans.inertia_)
plt.plot(range(1,11),intertia_scores,'--bo')
```

Out[694]:
```
[<matplotlib.lines.Line2D at 0x1be58647b50>]
```



In [695…
```
#  Looks like K=4 should be our choice
```

In [696…
```
clustering2 = KMeans(n_clusters=4)
identified_clusters=clustering2.fit(df_t[['recency_t','frequency_t','monetary_t']])
df_t['Kmeans_segmentation'] =clustering2.labels_
df_t.head()
```

Out[696]:

	index	user_id	recency	frequency	TOTAL_PURCHASES_EUR	monetary	recency_t	frequency_t
0	1	2	59.0	1	38.456	38.456	-0.847385	-0.475177
1	2	3	159.0	19	631.488	33.396	-0.224478	1.197279
2	7	8	17.0	1	19.228	19.228	-1.109006	-0.475177
3	12	13	4.0	19	587.972	31.372	-1.189984	1.197279
4	13	14	52.0	2	118.404	59.708	-0.890989	-0.382263

In [697…

```
centers =pd.DataFrame(clustering2.cluster_centers_)
centers.columns = ['recency_t','frequency_t','monetary_t']
```

In [698…

```
df_t.groupby(['Kmeans_segmentation']).agg({
'recency': 'mean',
'frequency': 'mean',
'monetary': 'mean',
'user_id': 'count'}).sort_values(by="frequency", ascending=False)
```

Out[698]:

Kmeans_segmentation	recency	frequency	monetary	user_id
2	33.303191	41.792553	25.940574	564
0	68.631623	6.428808	29.336942	6040
1	278.989583	2.401042	81.990448	768
3	364.796992	1.996778	25.860676	4655

In [699…

```
cus_segment=cus_data_raw.merge(df_t.rename({'user_id': 'user_id_r'}, axis=1),
               left_on='USER_ID', right_on='user_id_r', how='left')
cus_segment['Kmeans_segmentation'].fillna('4',inplace = True)
```

In [700…

```
cus_segment['Segment']=np.where(cus_segment['Kmeans_segmentation']==2,'Loyal Custom
                                    np.where(cus_segment['Kmeans_segmentation']==0,'Pot
                                         np.where(cus_segment['Kmeans_segmentation']
                                             np.where(cus_segment['Kmeans_segmenta
```

In [701…

```
cus_segment.to_csv("C:\\Users\\Namana\\OneDrive\\Desktop\\Projects\\Customer_Segmer
```

In [702…

```
fig = px.scatter_3d(df_t, x='recency', y='frequency', z='monetary',color='Kmeans_se
                    height=700,width=700,opacity=0.5,size_max=0.1,template='plotly_
                   )
fig.update_layout(title_text='Customers across RFM Clusters', title_x=0.5)
fig.show()
```

1 33.303191 41.792553 25.940574 564 ---Loyal Customers 3 68.555132 6.431769 29.292458 6031 ---Average customers but need to keep them in check they might leave soon 2 277.241645 2.424165 81.691033 778 ---High value Churning customers but they are big whales. We need them. 0 364.799957 1.996777 25.854491 4654 ---We have lost them. We need to brink them back