WOLT CUSTOMER SEGMENTATION WITH PYTHON

In this analysis I will explore data set on Wolt customers.

Customer segmentation is useful in understanding what demographic and psychographic sub-populations there are within customers.

Need of Customer Segmentation:

- It helps identifying the most potential customers.
- It helps managers to easily communicate with a targetted group of the audience.
- It improves the quality of service, loyalty, and retention.
- Improves customer relationship via better understanding needs of segments.
- It provides opportunities for upselling and cross-selling.
- It will help managers to design special offers for targetted customers, to encourage them to buy more products.
- It helps companies to stay a step ahead of competitors.
- It also helps in identifying new products that customers could be interested in.

Types of Segmentation:

- Demographic (Gender, Age, Marital Status…)
- Geographic (Location, Region, …)
- Behavioral (Spending, Consumption, Habits, Products, Services…)
- Psychographic (social status, Lifestyle, Personality, etc…)

With the data in our data set I will focus on behavioral segmentation, since we don't have demographic, geographic or psychographic information about customer in this data set.

Setup

First, I imported all the libraries I will need and loaded the csv file.
With functions ".head()" , ".T" and ".shape" I previewed the data and checked the amount of columns and rows.

```
#viewing the dataframe to see the columns and values, data types, etc.
customers_df.head(5)
```

	REGISTRATION_DATE	REGISTRATION_COUNTRY	PURCHASE_COUNT	PURCHASE_COUNT_DELIVERY	PURCHASE_COUNT_TAKEAWAY	FIRST_PURCHASE_DA
0	2019-09-01 00:00:00.000	DNK	0	NaN	NaN	Nal
1	2019-09-01 00:00:00.000	FIN	1	1.00	0.00	2020-09-02 00:00:00.00
2	2019-09-01 00:00:00.000	DNK	19	19.00	0.00	2019-12-10 00:00:00.00
3	2019-09-01 00:00:00.000	FIN	0	NaN	NaN	Nal
4	2019-09-01 00:00:00.000	GRC	0	NaN	NaN	Nal

5 rows × 30 columns

```
#better view, since it has too many columns
customers_df.T
```

	0	1	2	3	4	5	6	
REGISTRATION_DATE	2019-09-01 00:00:00.000	2019-09-01 00:00:00.000	2019-09-01 00:00:00.000	2019-09-01 00:00:00.000	2019-09-01 00:00:00.000	2019-09-01 00:00:00.000	2019-09-01 00:00:00.000	20 00:(
REGISTRATION_COUNTRY	DNK	FIN	DNK	FIN	GRC	FIN	DNK	
PURCHASE_COUNT	0	1	19	0	0	0	0	
PURCHASE_COUNT_DELIVERY	NaN	1.00	19.00	NaN	NaN	NaN	NaN	
PURCHASE_COUNT_TAKEAWAY	NaN	0.00	0.00	NaN	NaN	NaN	NaN	
FIRST_PURCHASE_DAY	NaN	2020-09-02 00:00:00.000	2019-12-10 00:00:00.000	NaN	NaN	NaN	NaN	20 00:(
LAST_PURCHASE_DAY	NaN	2020-09-02 00:00:00.000	2020-05-25 00:00:00.000	NaN	NaN	NaN	NaN	20 00:(
USER_ID	1	2	3	4	5	6	7	
BREAKFAST_PURCHASES	NaN	0.00	0.00	NaN	NaN	NaN	NaN	
LUNCH_PURCHASES	NaN	1.00	4.00	NaN	NaN	NaN	NaN	
EVENING_PURCHASES	NaN	0.00	1.00	NaN	NaN	NaN	NaN	
DINNER_PURCHASES	NaN	0.00	14.00	NaN	NaN	NaN	NaN	
LATE_NIGHT_PURCHASES	NaN	0.00	0.00	NaN	NaN	NaN	NaN	
TOTAL_PURCHASES_EUR	NaN	38.46	631.49	NaN	NaN	NaN	NaN	
DISTINCT_PURCHASE_VENUE_COUNT	NaN	1.00	9.00	NaN	NaN	NaN	NaN	
MIN_PURCHASE_VALUE_EUR	NaN	38.53	20.28	NaN	NaN	NaN	NaN	
MAX_PURCHASE_VALUE_EUR	NaN	38.61	43.69	NaN	NaN	NaN	NaN	
AVG_PURCHASE_VALUE_EUR	NaN	38.46	33.40	NaN	NaN	NaN	NaN	
PREFERRED_DEVICE	ios	android	android	android	android	android	ios	
IOS_PURCHASES	NaN	0.00	0.00	NaN	NaN	NaN	NaN	
WEB_PURCHASES	NaN	0.00	19.00	NaN	NaN	NaN	NaN	
ANDROID_PURCHASES	NaN	1.00	0.00	NaN	NaN	NaN	NaN	
PREFERRED_RESTAURANT_TYPES	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
USER_HAS_VALID_PAYMENT_METHOD	False	False	True	False	False	False	False	
MOST_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE	NaN	23.00	21.00	NaN	NaN	NaN	NaN	
MOST_COMMON_WEEKDAY_TO_PURCHASE	NaN	2.00	2.00	NaN	NaN	NaN	NaN	
AVG_DAYS_BETWEEN_PURCHASES	NaN	NaN	9.00	NaN	NaN	NaN	NaN	
MEDIAN_DAYS_BETWEEN_PURCHASES	NaN	NaN	3.00	NaN	NaN	NaN	NaN	
AVERAGE_DELIVERY_DISTANCE_KMS	NaN	6.85	6.56	NaN	NaN	NaN	NaN	
	{\n "General	{\n "General	{\n "General	{\n "General	{\n "General	{\n "General	{\n "General	{\n

Some data are not in the correct format.

The data of the last column **PURCHASE_COUNT_BY_STORE_TYPE** are in JSON, so I have to modify it to separate columns and remove the old column. I used ".replace()" function to remove unneeded characters from the string and ".split()" to split the column into separated columns.

```
customers_df['PURCHASE_COUNT_BY_STORE_TYPE'].replace(r'\s+|\\n', ' ', regex=True, inplace=True)
```

```
customers_df['PURCHASE_COUNT_BY_STORE_TYPE'].replace('}',' ', regex=True)
```

```
0        { "General merchandise": 0, "Grocery": 0, "Pet...
1        { "General merchandise": 0, "Grocery": 0, "Pet...
2        { "General merchandise": 1, "Grocery": 9, "Pet...
3        { "General merchandise": 0, "Grocery": 0, "Pet...
4        { "General merchandise": 0, "Grocery": 0, "Pet...
                              ...
21978    { "General merchandise": 0, "Grocery": 0, "Pet...
21979    { "General merchandise": 0, "Grocery": 0, "Pet...
21980    { "General merchandise": 0, "Grocery": 0, "Pet...
21981    { "General merchandise": 0, "Grocery": 0, "Pet...
21982    { "General merchandise": 0, "Grocery": 0, "Pet...
Name: PURCHASE_COUNT_BY_STORE_TYPE, Length: 21983, dtype: object
```

```
customers_df['PURCHASE_COUNT_BY_STORE_TYPE']=customers_df['PURCHASE_COUNT_BY_STORE_TYPE'].replace('{',' ', regex=True)
```

```
GROCERY','PET SUPPLIES','RESTAURANT','RETAIL STORE']]=customers_df['PURCHASE_COUNT_BY_STORE_TYPE'].str.split(',', 4, expand=True)
```

```
customers_df[['GENERAL_MERCH','GROCERY','PET SUPPLIES','RESTAURANT','RETAIL STORE']]
```

	GENERAL_MERCH	GROCERY	PET SUPPLIES	RESTAURANT	RETAIL STORE
0	"General merchandise": 0	"Grocery": 0	"Pet supplies": 0	"Restaurant": 0	"Retail store": 0 }
1	"General merchandise": 0	"Grocery": 0	"Pet supplies": 0	"Restaurant": 1	"Retail store": 0 }
2	"General merchandise": 1	"Grocery": 9	"Pet supplies": 0	"Restaurant": 9	"Retail store": 0 }
3	"General merchandise": 0	"Grocery": 0	"Pet supplies": 0	"Restaurant": 0	"Retail store": 0 }
4	"General merchandise": 0	"Grocery": 0	"Pet supplies": 0	"Restaurant": 0	"Retail store": 0 }
...	...	...	...	...	...
21978	"General merchandise": 0	"Grocery": 0	"Pet supplies": 0	"Restaurant": 1	"Retail store": 0 }
21979	"General merchandise": 0	"Grocery": 0	"Pet supplies": 0	"Restaurant": 0	"Retail store": 0 }
21980	"General merchandise": 0	"Grocery": 0	"Pet supplies": 0	"Restaurant": 0	"Retail store": 0 }
21981	"General merchandise": 0	"Grocery": 0	"Pet supplies": 0	"Restaurant": 0	"Retail store": 0 }
21982	"General merchandise": 0	"Grocery": 0	"Pet supplies": 0	"Restaurant": 1	"Retail store": 0 }

21983 rows × 5 columns

Also data in the PREFERRED RESTSURANT TYPE are in wrong lists. I use the eval function to convert the string into list of strings.

```
#column PREFERRED_RESTAURANT_TYPES has values as lists
customers_df['PREFERRED_RESTAURANT_TYPES'].head()
```

```
13    [\n  "american"\n]
16    [\n  "american"\n]
17    [\n  "japanese"\n]
26     [\n  "italian"\n]
31    [\n  "american"\n]
Name: PREFERRED_RESTAURANT_TYPES, dtype: object
```

```
#converting string into list of strings
customers_df['PREFERRED_RESTAURANT_TYPES'] = customers_df['PREFERRED_RESTAURANT_TYPES'].apply(eval)
```

```
customers_df['PREFERRED_RESTAURANT_TYPES'].head()
```

```
13    [american]
16    [american]
17    [japanese]
26     [italian]
31    [american]
Name: PREFERRED_RESTAURANT_TYPES, dtype: object
```

There is a lot of missing values in the data set. I decided to remove them, so they wont mess up the calculations.

```
customers_df.dropna(how='any', inplace=True)
```

Next I checked, if the data types of the values are correct. To view the datatypes I used ".dtypes" function. I convert the float values to 2 decimal, because they were rounded to no decimal places and it can cause discrepancies in the avg and max counts.
The columns that we created with the split function have wrong data type (object) the have to be converted to integers with function ".to_numberic". Also the date columns have to be converted to dates with function ".to_datetime"

```
customers_df.dtypes
```

```
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
GENERAL_MERCH                            object
GROCERY                                  object
PET SUPPLIES                             object
RESTAURANT                               object
RETAIL STORE                             object
dtype: object
```

```
#corecting the data types
```

```
# chagning float to 2 decimals, because it caoused discrepancies in the avg and max purchases
pd.options.display.float_format = '{:,.2f}'.format
```

```
ANT','RETAIL STORE']] =customers_df[['GENERAL_MERCH','GROCERY','PET SUPPLIES','RESTAURANT','RETAIL STORE']].apply(pd.to_numeric)
```

The column MOST COMMON HOUR OF THE DAY TO PURCHASE had values as float, it must be converted to time values and format only for hour "%H", since we don't have the minute and seconds.

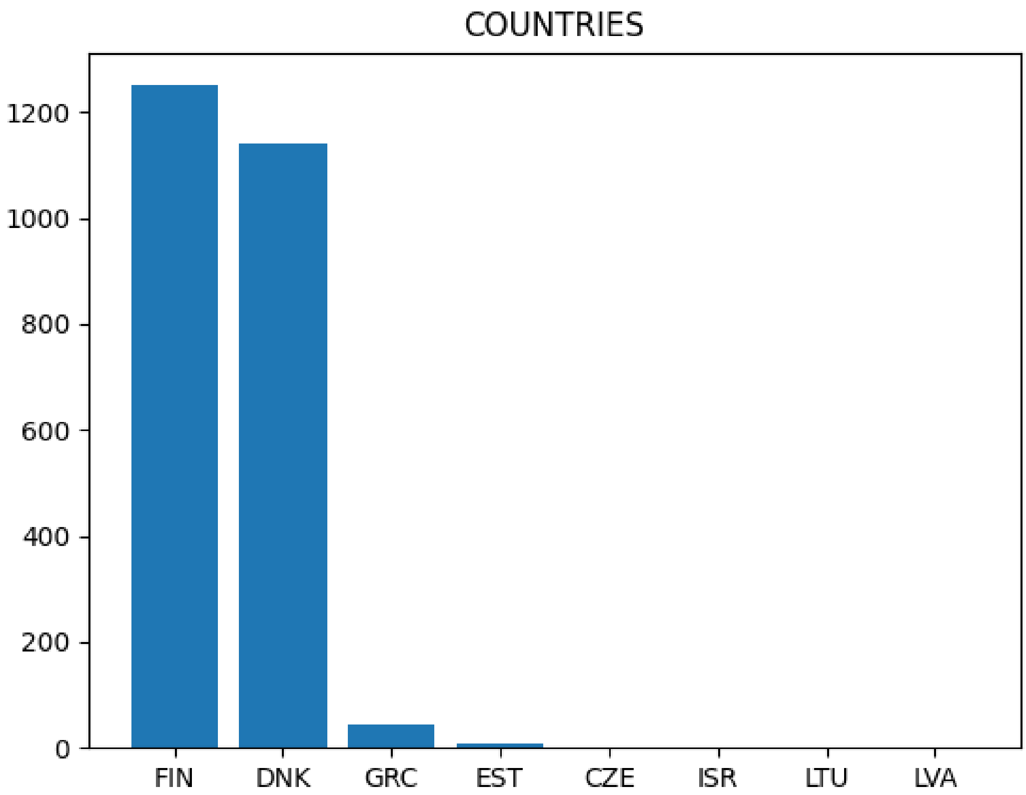
```
pe from float to time
_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE'] = pd.to_datetime(customers_df.MOST_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE, format='%H').dt.time
```

```
customers_df['MOST_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE']
```

```
13       05:00:00
16       10:00:00
17       07:00:00
26       00:00:00
31       23:00:00
           ...
21921    22:00:00
21923    20:00:00
21929    16:00:00
21947    10:00:00
21969    00:00:00
Name: MOST_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE, Length: 2447, dtype: object
```

The dataset includes data from different countries.

'FIN': 1250,
'DNK': 1139,
'GRC': 44,
'EST': 7,
'CZE': 2,
'ISR': 2,
'LTU': 2,
'LVA': 1



For the next data analysis, I will explore only data for Finland.

```
#filter data only for Finland
fin_data=customers_df[customers_df.REGISTRATION_COUNTRY=='FIN']
```

Exploring the data

With describe() I can see the statistics for each variable. We can see that none of the values are negative. (as they shouldn't be)

```
fin_data.describe()
#the average 'PURCHASE_COUNT'is 3, Avg purchases in eur is 171, avg delivery distance 6
```

	PURCHASE_COUNT	PURCHASE_COUNT_DELIVERY	PURCHASE_COUNT_TAKEAWAY	USER_ID	BREAKFAST_PURCHASES	LUNCH_PURCHASES	EVENIN
count	1,250.00	1,250.00	1,250.00	1,250.00	1,250.00	1,250.00	
mean	12.27	11.47	0.80	10,654.61	0.43	5.52	
std	15.28	14.91	2.81	6,509.69	1.27	9.07	
min	2.00	0.00	0.00	14.00	0.00	0.00	
25%	4.00	4.00	0.00	5,083.50	0.00	1.00	
50%	8.00	7.00	0.00	10,320.50	0.00	3.00	
75%	15.00	14.00	0.00	16,404.00	0.00	6.00	
max	221.00	221.00	44.00	21,970.00	17.00	110.00	

8 rows × 26 columns

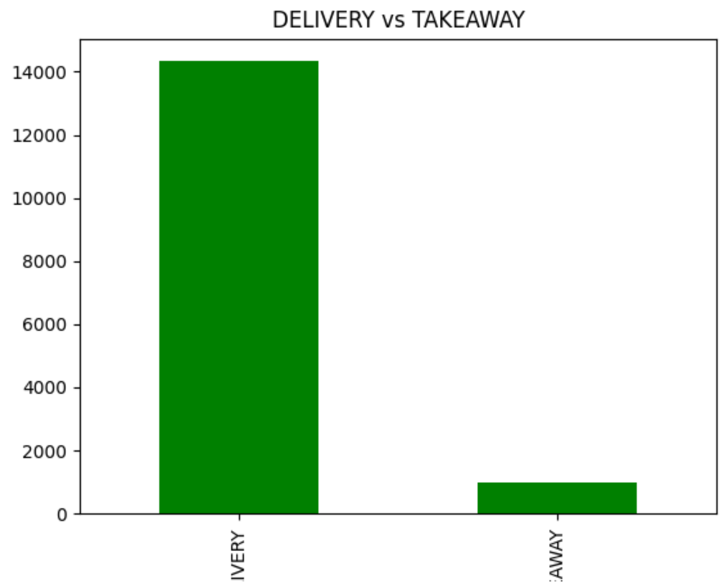
To see more detailed purchase values for each customer:

```
# the purchase values for each customer
fin_data[['USER_ID','TOTAL_PURCHASES_EUR','MIN_PURCHASE_VALUE_EUR','MAX_PURCHASE_VALUE_EUR','AVG_PURCHASE_VALUE_EUR']].dropna()
```

	USER_ID	TOTAL_PURCHASES_EUR	MIN_PURCHASE_VALUE_EUR	MAX_PURCHASE_VALUE_EUR	AVG_PURCHASE_VALUE_EUR
13	14	118.40	57.80	60.96	59.71
16	17	284.37	22.31	58.93	40.48
26	27	145.73	19.27	39.62	24.29
39	40	46.55	15.21	16.26	15.18
58	59	91.08	19.27	38.61	30.36
...	...	...	...	...	...
21830	21831	208.47	33.46	36.58	34.41
21886	21887	713.46	14.20	59.94	22.26
21905	21906	160.91	13.18	25.40	18.22
21923	21924	50.60	16.22	17.27	17.20
21969	21970	115.37	14.20	34.54	19.23

DELIVERY vs TAKEAWAY:

```
fin_data[['PURCHASE_COUNT_DELIVERY','PURCHASE_COUNT_TAKEAWAY']].sum().plot.bar (color='GREEN')
plt.title('DELIVERY vs TAKEAWAY')
plt.show()
fin_data[['PURCHASE_COUNT_DELIVERY','PURCHASE_COUNT_TAKEAWAY']].sum()
```



PURCHASE_COUNT_DELIVERY   14,337.00
PURCHASE_COUNT_TAKEAWAY   1,004.00

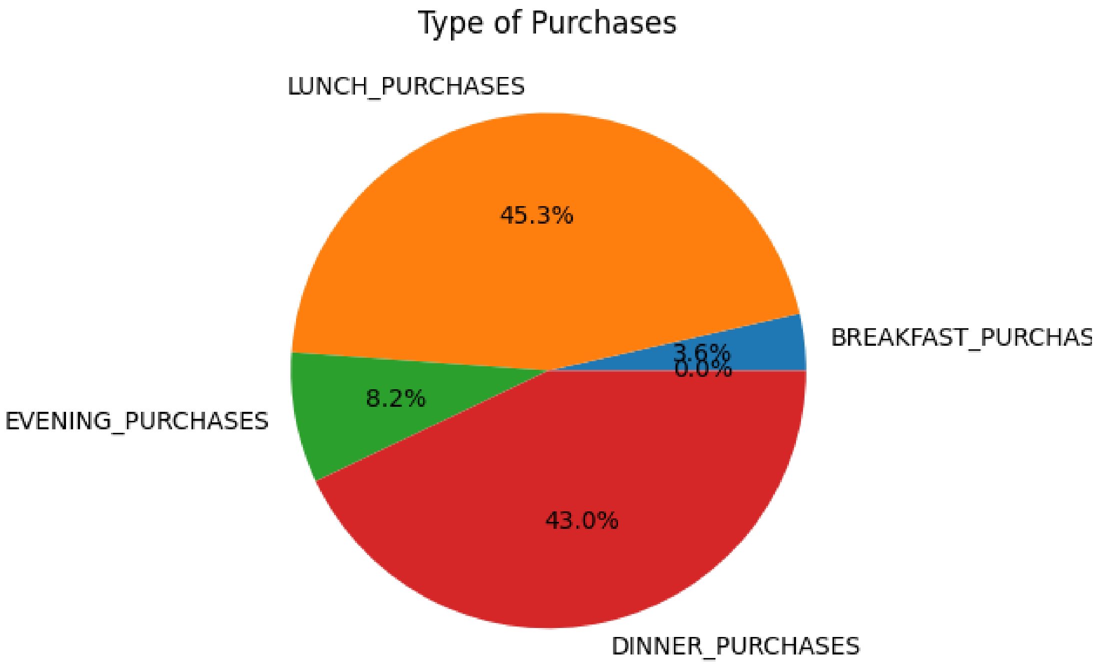
Customers strongly prefer delivery between takeaways.

TYPES OF PURCHASES:

```
fin_data[['BREAKFAST_PURCHASES','LUNCH_PURCHASES','EVENING_PURCHASES','DINNER_PURCHASES','LATE_NIGHT_PURCHASES']].sum().plot(kind

fin_data[['BREAKFAST_PURCHASES','LUNCH_PURCHASES','EVENING_PURCHASES','DINNER_PURCHASES','LATE_NIGHT_PURCHASES']].sum()
```

```
BREAKFAST_PURCHASES       541.00
LUNCH_PURCHASES         6,902.00
EVENING_PURCHASES       1,249.00
DINNER_PURCHASES        6,544.00
LATE_NIGHT_PURCHASES        0.00
```



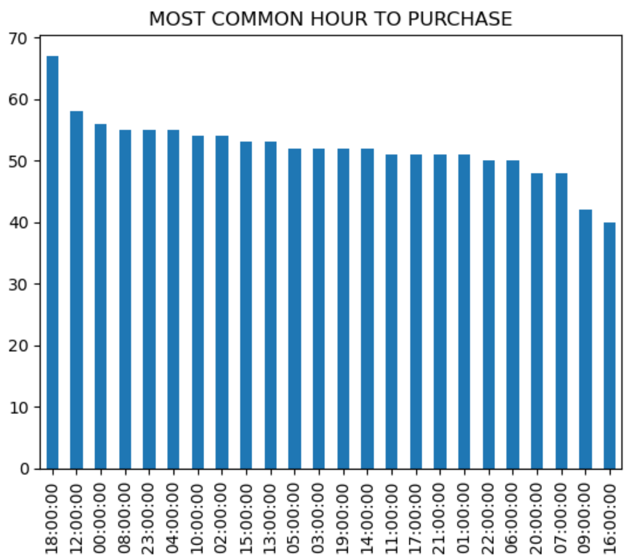
Customers purchase mostly lunch and dinner. Rarely in the evenings and for breakfast and none of the customers purchased late at night

MOST COMMON HOUR TO PURCHASE:

```
fin_data['MOST_COMMON_HOUR_OF_THE_DAY_TO_PURCHASE'].value_counts().plot.bar()
plt.title('MOST COMMON HOUR TO PURCHASE')

#as mentioned before, customers prefer to purchase in the evening and lunch (18:00 and 12:00)
```

```
Text(0.5, 1.0, 'MOST COMMON HOUR TO PURCHASE')
```



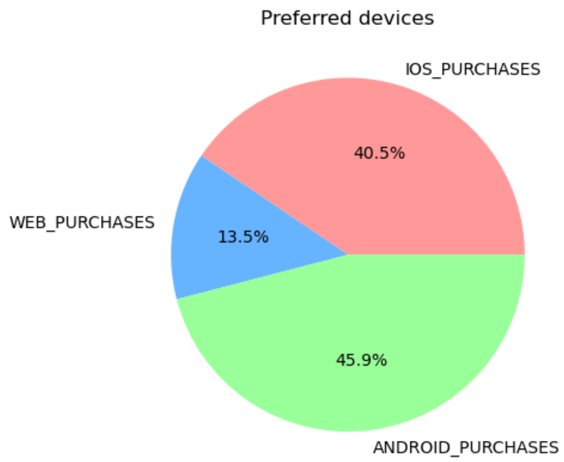
The most common hours to purchase are 18:00 and 12:00 as mentioned before that most purchases are done for dinner and lunch. But there are also purchase hours from middle of the night which disagrees with the previous data. It could be caused by the way the questionnaire for customers was created or filled.

PREFERRED DEVICES:

```
#PREFERRED DEVICES customers use for orders
colors = ['#ff9999','#66b3ff','#99ff99','#ffcc99']
fin_data[['IOS_PURCHASES','WEB_PURCHASES','ANDROID_PURCHASES']].sum().plot(kind="pie", autopct='%1.1f%%',colors=colors, ylabel=''
fin_data[['IOS_PURCHASES','WEB_PURCHASES','ANDROID_PURCHASES']].sum()

#customers prefer to order from phones Android purchases are slighting more used than IOS but alltogether customers order usuall
```

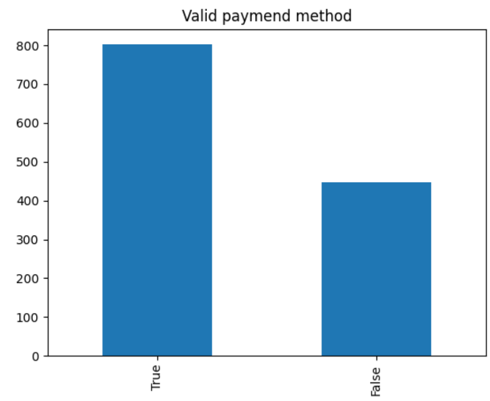
```
IOS_PURCHASES        6,220.00
WEB_PURCHASES        2,072.00
ANDROID_PURCHASES    7,049.00
dtype: float64
```

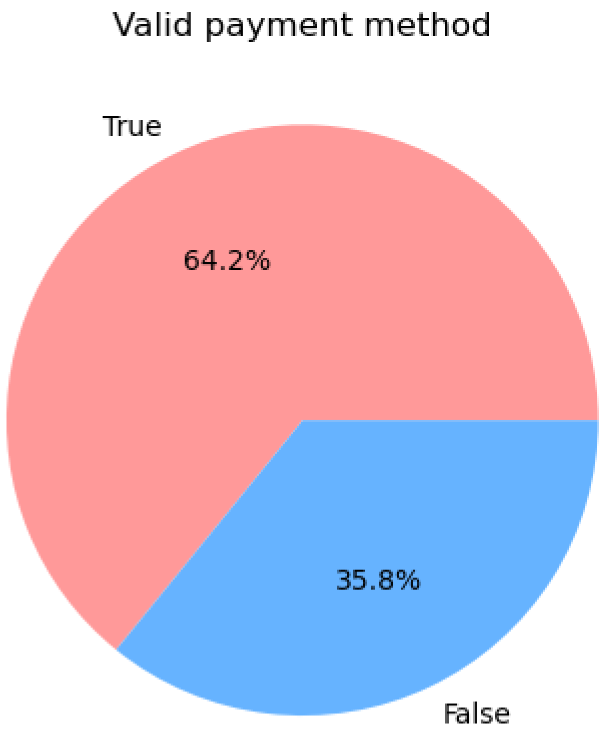


Customers purchase from their phones and rarely from the web. The percentage of using IOS and Android is almost the same.

VALID PAYMENT METHOD:

```
#it seems like more users have valid payment methon than not valid
fin_data['USER_HAS_VALID_PAYMENT_METHOD'].value_counts().plot.bar()
fin_data['USER_HAS_VALID_PAYMENT_METHOD'].value_counts()
plt.title('VALID PAYMENT METHOD')
```





Most customers have no problems with payment, but there still occur almost 36 % of customers with no valid payment method.

MOST COMMON PURCHASE WEEKDAY:

The weekdays were in the dataset marked as numbers. I decided to change the s to weekday names, so it would be easier to read the preferred days. The function **dayNameFromWeekday()** will return the correct day name to particular number. With that function I could convert the numbers to the name by using the function **changeToWeekday** and finally use the converted values in the graph.

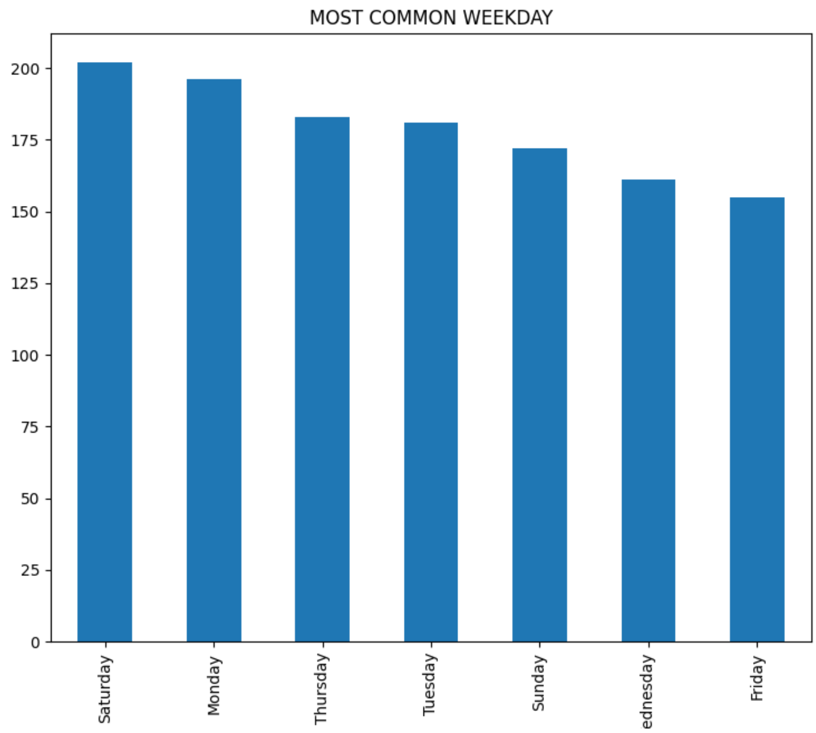
The results show that customers purchase the most on Saturday, than Monday, followed by Thursday, then Tuesday. They purchase the least on Fridays.

```python
#the week days were in integers, I decided to convert them to week day names
def dayNameFromWeekday(weekday):
    if weekday == 1.00:
        return "Monday"
    if weekday == 2.00:
        return "Tuesday"
    if weekday == 3.00:
        return "Wednesday"
    if weekday == 4.00:
        return "Thursday"
    if weekday == 5.00:
        return "Friday"
    if weekday == 6.00:
        return "Saturday"
    if weekday == 7.00:
        return "Sunday"
```

```python
def changeToWeekday(fin_data):
    data=fin_data['MOST_COMMON_WEEKDAY_TO_PURCHASE'].apply(lambda x: dayNameFromWeekday(x))
    return data
```

```python
changeToWeekday(fin_data).value_counts().plot.bar()
changeToWeekday(fin_data).value_counts()
#customers purchase mostly on Saturday and Monday
```

```
Saturday     202
Monday       196
Thursday     183
Tuesday      181
Sunday       172
Wednesday    161
Friday       155
Name: MOST_COMMON_WEEKDAY_TO_PURCHASE, dtype: int64
```



PREFERRED RESTAURANT TYPE:

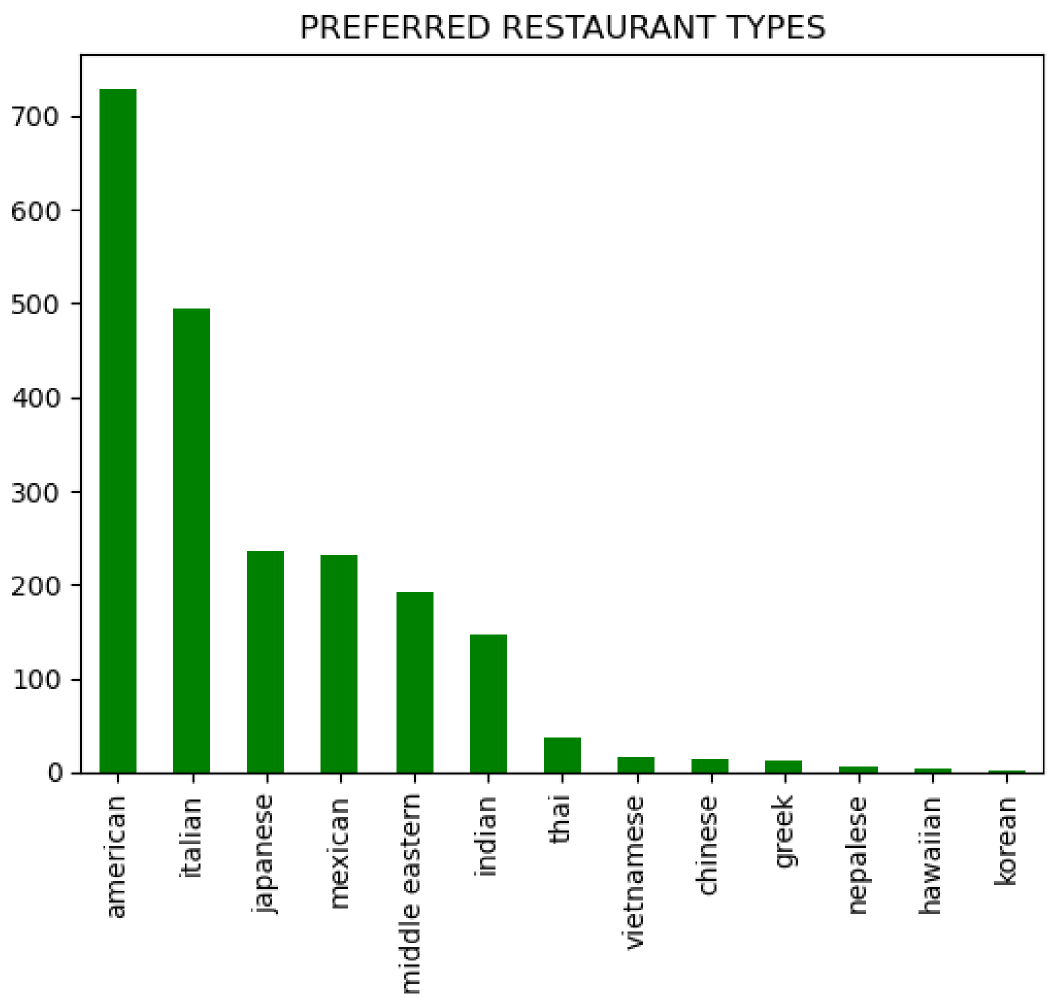
The data for restaurant types were in lists, which I converted to list of strings before. Now I had to reduce the lists dimension from 2 to 1.

```
#If we conceptualize the column as a 2D array, reducing its dimensions from 2 to 1 would allow
def to_1D(series):
 return pd.Series([x for _list in series for x in _list])
```

```
to_1D(fin_data['PREFERRED_RESTAURANT_TYPES']).value_counts()
```

```
american         729
italian          495
japanese         235
mexican          232
middle eastern   192
indian           147
thai              37
vietnamese        16
chinese           14
greek             13
nepalese           6
hawaiian           3
korean             2
dtype: int64
```

```
#customers prefer to order from american, italian and japanese restaurants the mosts.
to_1D(fin_data['PREFERRED_RESTAURANT_TYPES']).value_counts().plot(kind='bar',color='GREEN',title='PREFERRED RESTAURANT TYPES')
```



729 purchases were from American restaurants, 495 from Italian, 235 from Japanese, 232 from Mexican and 192 from middle eastern…Least purchases were from Korean restaurant.

AVERAGE DELIVERY DISTANCE:

```
fin_data['AVERAGE_DELIVERY_DISTANCE_KMS'].value_counts()
```

```
4.41     3
1.99     3
10.36    2
10.38    2
0.95     2
        ..
6.80     1
3.89     1
10.90    1
11.91    1
3.32     1
Name: AVERAGE_DELIVERY_DISTANCE_KMS, Length: 1180, dtype: int64
```

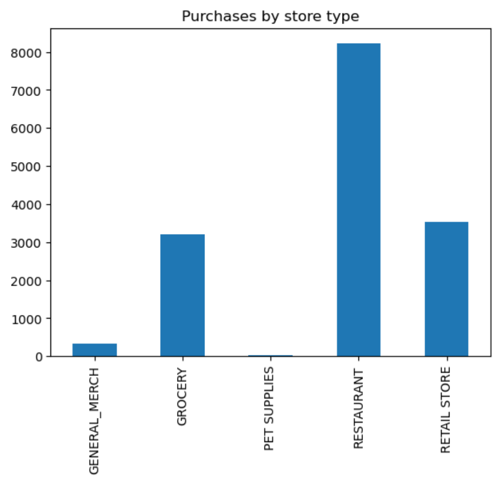
AVG delivery distance is mostly between 4,41 and 1,99 km. From the describe function view we can see that the overall avg distance is 5.94 km.

AVERAGE_DELIVERY_DISTANCE_KMS
1,250.00
5.94
3.51
0.00
2.92

PURCHASE COUNT BY STORE TYPE:

```
fin_data[['GENERAL_MERCH','GROCERY','PET SUPPLIES','RESTAURANT','RETAIL STORE']].sum().plot(kind='bar',title='Purchases by store
```

```
<AxesSubplot:title={'center':'Purchases by store type'}>
```



```
fin_data[['GENERAL_MERCH','GROCERY','PET SUPPLIES','RESTAURANT','RETAIL STORE']].sum()
```

```
9]: GENERAL_MERCH     339
    GROCERY          3206
    PET SUPPLIES       39
    RESTAURANT       8218
    RETAIL STORE     3539
    dtype: int64
```

The result says that clearly most of the purchases are from restaurants, then retail stores and groceries. 339 purchases were from general merch stores and only 39 purchases from pet supplies stores.

Using RFM ANALYSIS

RFM (Recency, Frequency, Monetary) analysis is a behaviour-based approach grouping customers into segments. It groups the customers based on their previous purchase transactions.

- Recency (R): Who have purchased recently. Number of days since last purchase.
- Frequency (F): Who has purchased frequently. It means the total number of purchases.
- Monetary Value(M): Who have high purchase amount. It means the total money customer spent.

First I created a dataframe with columns that will need for the calculations.

```
fin_data=fin_data[['USER_ID','PURCHASE_COUNT','FIRST_PURCHASE_DAY','LAST_PURCHASE_DAY','TOTAL_PURCHASES_EUR']]
```

Setting the recent day needed for the recency calculations. It is the latest purchase day. Recency will be calculated as the interval of the last purchase and today(recent day in this case).
Frequency is the number of purchases, PURCHASE COUNT from our dataset.
Monetary value is the TOTAL PURCHASE PRICE of the customer.

Then I renamed the columns to recency, frequency, monetary , kept the name for first purchase day and removed the missing values.

```
recent_date = fin_data['LAST_PURCHASE_DAY'].max()
```

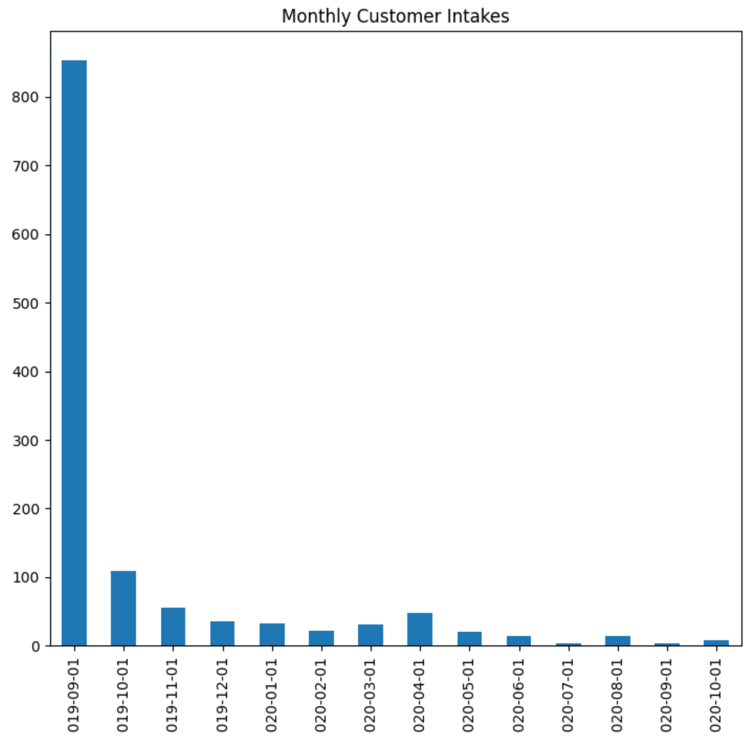
```
rfm= fin_data.groupby('USER_ID').agg({'LAST_PURCHASE_DAY': lambda date: (recent_date - date.max()).days,
                                      'PURCHASE_COUNT': lambda num: num*1,
                                      'TOTAL_PURCHASES_EUR': lambda price: price*1,
                                  'FIRST_PURCHASE_DAY':lambda date:date })
```

```
#rename columns
rfm.columns=['recency','frequency','monetary', 'FIRST_PURCHASE_DAY']
```

```
#removing missing values
rfm=rfm.dropna()
```

From the FIRST PURCHASE DAY I calculated the FIRST PURCHASE MONTH and with that we can look at the monthly customer intake

```
rfm["FIRST_PURCHASE_MONTH"] = rfm["FIRST_PURCHASE_DAY"].apply(lambda x: x.replace(day=1))
rfm.groupby(["FIRST_PURCHASE_MONTH"]).count()["FIRST_PURCHASE_DAY"].plot(kind="bar")
plt.title("Monthly Customer Intakes")
```



Calculate the R, F, M scores:

Ranking Customer's based upon their recency, frequency, and monetary score First I normalize the rank of the customers.

```
rfm['R_rank'] = rfm['recency'].rank(ascending=False)
rfm['F_rank'] = rfm['frequency'].rank(ascending=True)
rfm['M_rank'] = rfm['monetary'].rank(ascending=True)

# normalizing the rank of the customers
rfm['R_rank_norm'] = (rfm['R_rank']/rfm['R_rank'].max())*100
rfm['F_rank_norm'] = (rfm['F_rank']/rfm['F_rank'].max())*100
rfm['M_rank_norm'] = (rfm['M_rank']/rfm['M_rank'].max())*100
```

RFM score is calculated based upon recency, frequency, monetary value normalized ranks. Based upon this score we divide our customers. Here I rate them on a scale of 5.

Formula used for calculating rfm score is :
0.15*Recency score + 0.28*Frequency score + 0.57 *Monetary score

```
rfm['RFM_Score'] = 0.15*rfm['R_rank_norm']+0.28 * \
    rfm['F_rank_norm']+0.57*rfm['M_rank_norm']
rfm['RFM_Score'] *= 0.05
rfm = rfm.round(2)
```

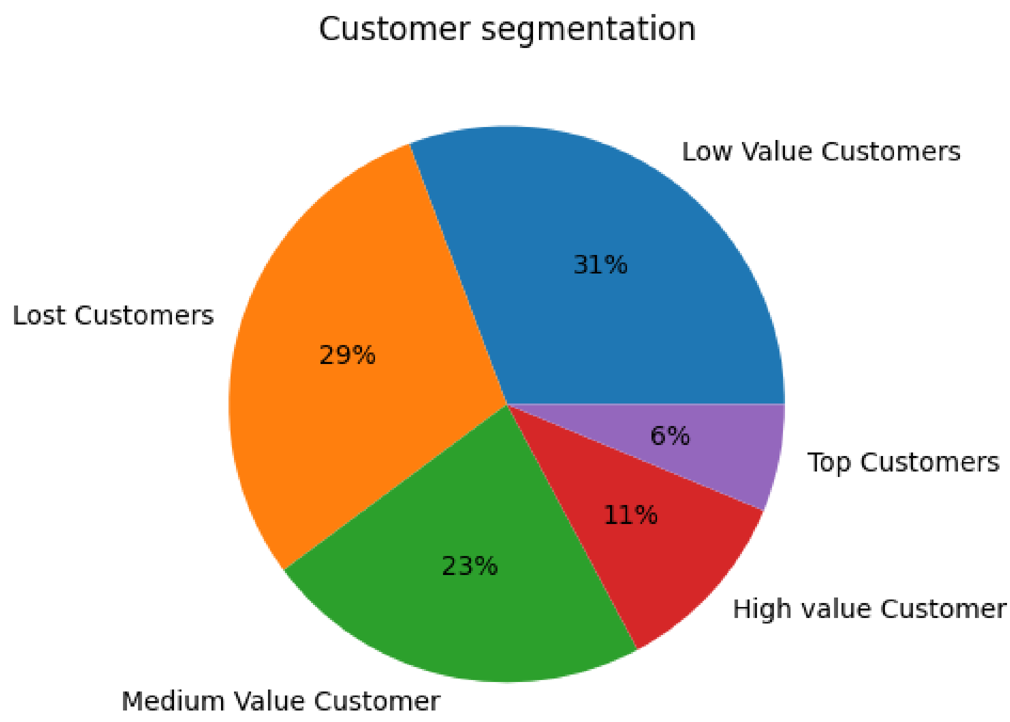
Rating Customer based upon the RFM score:
rfm score >4.5 : Top Customer
4.5 > rfm score > 4 : High Value Customer
4>rfm score >3 : Medium value customer
3>rfm score>1.6 : Low-value customer
rfm score<1.6 :Lost customer

```
rfm["Customer_segment"] = np.where(rfm['RFM_Score'] >
                                     4.5, "Top Customers",
                                     (np.where(
                                       rfm['RFM_Score'] > 4,
                                       "High value Customer",
                                       (np.where(rfm['RFM_Score'] > 3,
                             "Medium Value Customer",
                             np.where(rfm['RFM_Score'] > 1.6,
                            'Low Value Customers', 'Lost Customers'))))))
rfm[['RFM_Score', 'Customer_segment']].head()
```

USER_ID	RFM_Score	Customer_segment
14	1.18	Lost Customers
17	3.19	Medium Value Customer
27	2.09	Low Value Customers
40	0.37	Lost Customers
59	0.72	Lost Customers

VISUALIZATION:

```
plt.pie(rfm.Customer_segment.value_counts(),
        labels=rfm.Customer_segment.value_counts().index,
        autopct='%.0f%%')
plt.title('Customer segmentation')
plt.show()
```



Based on the visualization about, most of the customers are **low value customers** (31%) which means that their rfm score is between 3 and 1.6. and **lost customers(**30%**),** their rfm score is less than 1.6. 23% are **medium value customers,** 11% **high value customers** and 6% **top customers.**

Top customers completed recent purchase, buy frequently and spend the most. Together with high value customers they are important for the company. To keep them motivated, company could reward them, build somehow loyalty. With lost customers, company can try to reactivate them with for example personalized campaigns and if it doesn't work, ignore them, and focus the important customers.

We should not forget that customers are dynamic and can jump from one category to another.