Title: Business Analysis and recommendations

Content of the document

1.Introduction

More and more, brands have enlisted digital technologies to scale their businesses at the center of what we'd like to call a new era of online shopping. eCommerce is bigger than ever before, especially now that new tools exist to improve platform efficiencies across the board.

Today, an eCommerce store is the primary retail growth engine for any business. In fact, eCommerce is expected to claim 17% of the industry by the end of 2022. The meteoric growth witnessed by digital commerce over the last decade has also given rise to several eCommerce challenges. With the increasing size and demand for online business, riding the digital commerce wave is not easy.

A demanding majority of shoppers (nearly 81%) begin their journey with online research. Further, 71% of the customers say that a fast and highly responsive online marketplace is essential for a pleasant shopping experience.

The leaders of the business need to strengthen their tools and capabilities for better reach and visibility amongst the consumers. Retailers should use additional customer data strategically to better understand individual customer behavior and create a personalized customer experience that should start by analyzing their eCommerce trends.

1.1. Key Study

A large multi-category online store in the US was looking for more insights about one of its business categories, Cosmetic. They wanted to understand customer behavior, so they could optimize conversions at each stage of the process. They sell only cosmetic products in the Cosmetic sub-shop.

1.2. Problem defined

This sub-shop is going to be my client. In order to help him with his request, I will analyze the file they have shared with me to give recommendations on performance improvements and strategies.

The client has collected data (5 months: Oct 2019- Feb 2019) regarding the behavior of their customers.

A. The structure of the data is as follows:

```
1.Event_time: Time when event happened at (in UTC).
2.Event_type: The type of the event
3.product_iD: ID of a product
4.category_id: Product's category ID
5.category_code: Product's category taxonomy (code name) if it was possible to make it.
                 Usually present for meaningful categories and skipped for different kinds of accessories.
6.brand: Downcased string of brand name. Can be missed.
7.price: Float price of a product. Present.
8.user_id: Permanent user ID.
9.user_session: Temporary user's session ID. Each session will change every time user come back to online store
```

```
                from a long pause.
```

B. Event types: events can be

- view : a user viewed a product
- cart : a user added a product to shopping cart
- remove_from_cart : a user removed a product from shopping cart
- purchase : a user purchased a product

C. Multiple purchases per session A session can have multiple purchase events.

This study focus is identifying key insights across every stage of the buying process, from viewing a product to adding or removing it from the shopping cart to making a purchase, in order to optimize conversion rates by providing critical recommendations and performance improvements.

1.3. Steps of the key study Analysis

1. Upload the required modules
2. Load the data
3. Preparing Data
4. Exploratory Analysis & Recommendations
5. Conclusion
6. Reference

2. Preparing environment and data

2.1.Upload the required modules

I will use this step to call the necessary modules to help me improve my algorithms during my analysis

In [1]:
```
#install the necessary library
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import datetime
from IPython.display import display, Markdown
import plotly.express as px
from plotly import graph_objects as gph
from plotly.subplots import make_subplots
from datetime import datetime
import matplotlib.dates as dates
import matplotlib.ticker as ticker
```

2.2.Load the data

This step will involve uploading the five files shared from the client separately, each with data from a different month, separately. I will then combine them into one file.My next step will be to visualize the header with the first five lines to see the different attributes.

- The Dataset is loaded in Jupiter notebook in the folder "Nassima Guenaoui Projects/Dataset/" on a "CSV" format.

In [2]:
```
# load the data
d_oct = pd.read_csv("2019-Oct.csv")
d_nov = pd.read_csv("2019-Nov.csv")
d_dec = pd.read_csv("2019-Dec.csv")
d_jan = pd.read_csv("2020-Jan.csv")
d_feb = pd.read_csv("2020-Feb.csv")
```

In [3]:
```
# add a column about the month to help me differentiate between months after I combine then in one file
d_oct['event_month'] = "10-2019"
d_nov['event_month'] = "11-2019"
d_dec['event_month'] = "12-2019"
d_jan['event_month'] = "01-2020"
d_feb['event_month'] = "02-2020"
```

In [4]:
```
# combine all the loaded data into a single data frame
d_all_months = pd.concat([d_oct, d_nov, d_dec, d_jan, d_feb])
#check the shape of the file
print(d_all_months.shape)
```

```
(20692840, 10)
```

In [5]:
```
#Count the total number of events(rows) per month
d_query = "Total events for: "
d_query += "Oct 2019: " + str(len(d_nov))+", "
d_query += "Nov 2019: " + str(len(d_oct))+", "
d_query += "Dec 2018: " + str(len(d_dec))+", "
d_query += "Jan 2021: " + str(len(d_jan))+", "
d_query += "Feb 2021: " + str(len(d_feb))+".  "

d_query += "Grand total: " +" "+ str(len(d_all_months)) + " events."

display(Markdown(d_query))
```

Total events for: Oct 2019: 4635837, Nov 2019: 4102283, Dec 2018: 3533286, Jan 2021: 4264752, Feb 2021: 4156682. Grand total: 20692840 events.

The shape of the collected data has 20692840 observations and 10 attributes

In [6]:
```
# Look into the structure of my data | Return top 5 rows of the data frame.
d_all_months.head()
```

Out[6]:

	event_time	event_type	product_id	category_id	category_code	brand	price	user_id	user_session	event_month
0	2019-10-01 00:00:00 UTC	cart	5773203	1487580005134238553	NaN	runail	2.62	463240011	26dd6e6e-4dac-4778-8d2c-92e149dab885	10-2019
1	2019-10-01 00:00:03 UTC	cart	5773353	1487580005134238553	NaN	runail	2.62	463240011	26dd6e6e-4dac-4778-8d2c-92e149dab885	10-2019
2	2019-10-01 00:00:07 UTC	cart	5881589	2151191071051219817	NaN	lovely	13.48	429681830	49e8d843-adf3-428b-a2c3-fe8bc6a307c9	10-2019
3	2019-10-01 00:00:07 UTC	cart	5723490	1487580005134238553	NaN	runail	2.62	463240011	26dd6e6e-4dac-4778-8d2c-92e149dab885	10-2019
4	2019-10-01 00:00:15 UTC	cart	5881449	1487580013522845895	NaN	lovely	0.56	429681830	49e8d843-adf3-428b-a2c3-fe8bc6a307c9	10-2019

2.3.Preparing the Data

Preprocessing the data already loaded will ensure that we don't have problems later in our interpretations.

I will start by calling the function info() to know the structure of the DataFrame, memory usage, range index, and the number of cells in each column (non-null values).

In [7]:
```
d_all_months.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20692840 entries, 0 to 4156681
Data columns (total 10 columns):
 #   Column         Dtype
---  ------         -----
 0   event_time     object
 1   event_type     object
 2   product_id     int64
 3   category_id    int64
 4   category_code  object
 5   brand          object
 6   price          float64
 7   user_id        int64
 8   user_session   object
 9   event_month    object
dtypes: float64(1), int64(3), object(6)
memory usage: 1.7+ GB
```

2.3.1 Check summary statistics

The purpose of using describe() is to show the summary statistics of the DataFrame and look into it moving ahead for further analysis

Checking datatypes and range for filtering unrelavent data

In [8]:
```
d_all_months.describe()
```

Out[8]:

	product_id	category_id	price	user_id
count	2.069284e+07	2.069284e+07	2.069284e+07	2.069284e+07
mean	5.484297e+06	1.554230e+18	8.534735e+00	5.215527e+08
std	1.305716e+06	1.691038e+17	1.938142e+01	8.744312e+07
min	3.752000e+03	1.487580e+18	-7.937000e+01	4.654960e+05
25%	5.724650e+06	1.487580e+18	2.060000e+00	4.818306e+08
50%	5.810720e+06	1.487580e+18	4.050000e+00	5.531297e+08
75%	5.857864e+06	1.487580e+18	7.060000e+00	5.788573e+08
max	5.932595e+06	2.242903e+18	3.277800e+02	6.220902e+08

The price has negative entries in the table above. We can only think about canceled orders now. In that case,the negative price will only show up when the purchase is made (event type:purchase).

- To investigate this:
  - I will count the number of rows that has a negative price
  - I will filter the rows that have an event: purchase and negative price at the same time
  - I will compare the two numbers together if they are not equal to each other. That means not only the rows with the event: purchase has negative values.
    And that would mean that these negative orders are not necessarily coming from canceled orders

In [9]:
```
#count the total number of rows that has a negative value
negative_price=d_all_months[d_all_months['price']<0]['price'].count()
print("The total number of rows with the negative price is",negative_price)
```

```
The total number of rows with the negative price is 131
```

In [10]:
```
#Count the total number of rows orders that have a negative price and have purchase as event types.

#filter the events that are considered as  order (has  purchase as event type)
```

```
orders=d_all_months.loc[d_all_months['event_type'].isin(['purchase'])].drop_duplicates()

#count the number of orders that has a negative price
cancelled_orders=orders[orders['price']<0]['price'].count()

print("Total number of cancelled orders coming from purchase event is",cancelled_orders)
```

```
Total number of cancelled orders coming from purchase event is 119
```

- The canceled order is only 119 and it is < 131, which means the negative values of price are not only coming from orders.
- The negative values are not canceled orders. We can check the % of the rows with negative prices.

In [11]:
```
# Check the percentage of rows with negative price from the total rows we have
negative_price_p=negative_price/(d_all_months['price'].count())

print("Total number of rows with a negative price is %2d from total rows which is only %.5f "
      %(negative_price,round(negative_price_p,5)))
```

```
Total number of rows with a negative price is 131 from total rows which is only 0.00001
```

0.01% is almost nonexistent. As a conclusion, I will remove these records from my dataset by creating another variable and excluding those rows.

In [12]:
```
#Exclude rows with negative price
all_months =d_all_months[d_all_months['price']>=0]
```

2.3.2. Checking Null data

In [13]:
```
#count the missing values in each column
all_months.isnull().sum()
```

Out[13]:
```
event_time              0
event_type              0
product_id              0
category_id             0
category_code    20339115
brand             8756986
price                   0
user_id                 0
user_session         4598
event_month             0
dtype: int64
```

- Both Category_code and brand columns has the majority of missing values.
- On the other hand, the user_session column has a very minimal missing value.

Lets check the number of rows where both category_code & brand have Null value the same time

In [14]:
```
#no of rows with null values
null_values=all_months['category_code'].isnull() & all_months['brand'].isnull()
print("Common rows of Category_code & brand with Null value",null_values.sum())
```

```
Common rows of Category_code & brand with Null value 8670141
```

Now,I would calculate the percentage for each missing value.

In [15]:
```
# Calculate the Percentage of missing values in All columns
perc= all_months.isnull().sum() * 100 / len(all_months)
print(round(perc,2))
```

```
event_time        0.00
event_type        0.00
product_id        0.00
category_id       0.00
category_code    98.29
brand            42.32
price             0.00
user_id           0.00
user_session      0.02
event_month       0.00
dtype: float64
```

I will remove the rows with null values because my focus would be on events with a defined brand and category code.

In [16]:
```
#remove null values
all_months=all_months.dropna()
```

In [17]:
```
print("now we have reduced the observations from (20692840,10) to ")
all_months.shape
```

Out[17]:
```
now we have reduced the observations from (20692840,10) to
(266709, 10)
```

In [18]:
```
# Checking if we still have any missing values
all_months.isnull().sum()
```

Out[18]:
```
event_time       0
event_type       0
product_id       0
category_id      0
category_code    0
brand            0
price            0
user_id          0
user_session     0
event_month      0
dtype: int64
```

The data now is ready for the next step

2.3.3. Create DateTime columns

In this step, I will split the timezone from time_event and then I will create different time columns:

- Date,
- Time
- Hour
- Weekday: for weekday we will convert it from a number to a string to make things more easier
- Week number

In [19]:
```
#seperate timezone
# Split time_event to 2 different attributes event_history and time_zone
all_months["time_zone"]= all_months["event_time"].str.rsplit(" ", n=1,expand = True)[1]
all_months["event_history"]= all_months["event_time"].str.rsplit(" ", n=1,expand = True)[0]

#convert event_history from a string representation of a date to an actual date format
all_months["event_history"]=pd.to_datetime(all_months["event_history"])
```

In [20]:
```
#convert event_history column to just a date and affect it to a new column called date
all_months["date"]=all_months['event_history'].dt.date
```

In [21]:
```
#convert event_history column to just a time and affect it to a new column called time
all_months["event_time"]=all_months['event_history'].dt.time
```

In [22]:
```
#convert event_history column to just hour and affect it to a new column called hour
all_months["hours"]=all_months['event_history'].dt.hour
```

In [23]:
```
#convert event_history column to a weekday and affect it to a new column called weekday
all_months["weekday"]=all_months['event_history'].dt.weekday
```

In [24]:
```
#convert event_history column to just a month and affect it to a new column called weeknumber
all_months['weeknumber']=all_months['event_history'].dt.isocalendar().week

#Add 'W' as a  prefix to weeknum
all_months['weeknumber'] = 'W' + all_months['weeknumber'].astype(str)
```

In [25]:
```
#convert weekday from a number to a string to make it easier for reading
all_months['weekday']= all_months['weekday'].replace({0:'Monday',1:'Tuesday',2:'Wednesday',3:'Thursday',4:'Friday',5:'Saturday',6:'Sunday'})
```

In [26]:
```
#check to see if the new attributes were added
print('The new dataset after adding time columns will looks as follow:')
all_months.head()
```

Out[26]:

The new dataset after adding time columns will looks as follow:

	event_time	event_type	product_id	category_id	category_code	brand	price	user_id	user_session	event_month	time_zone	event_history	date	hours	weekday	weeknumber
21	00:00:53	view	5856191	1487580006350586771	appliances.environment.vacuum	runail	24.44	507355498	944c7e9b-40bd-4112-a05b-81e73f37e0c0	10-2019	UTC	2019-10-01 00:00:53	2019-10-01	0	Tuesday	W40
264	00:07:58	remove_from_cart	5767493	1487580013053083824	stationery.cartrige	italwax	2.14	514753614	e2fecb2d-22d0-df2c-c661-15da44b3ccf1	10-2019	UTC	2019-10-01 00:07:58	2019-10-01	0	Tuesday	W40
269	00:08:03	remove_from_cart	5759489	1487580013053083824	stationery.cartrige	italwax	2.14	514753614	e2fecb2d-22d0-df2c-c661-15da44b3ccf1	10-2019	UTC	2019-10-01 00:08:03	2019-10-01	0	Tuesday	W40
276	00:08:10	remove_from_cart	24336	1487580013053083824	stationery.cartrige	depilflax	3.02	514753614	e2fecb2d-22d0-df2c-c661-15da44b3ccf1	10-2019	UTC	2019-10-01 00:08:10	2019-10-01	0	Tuesday	W40
278	00:08:18	remove_from_cart	5775822	1487580013053083824	stationery.cartrige	italwax	2.14	514753614	e2fecb2d-22d0-df2c-c661-15da44b3ccf1	10-2019	UTC	2019-10-01 00:08:18	2019-10-01	0	Tuesday	W40

2.3.4. Checking the Unique Values

Lets check the number of Unique values in the dataframe

In [27]:
```
#checking number of unique values in dataframe
all_months.nunique()
```

Out[27]:
```
event_time        74251
event_type            4
product_id          404
category_id          15
category_code        10
brand                44
price               328
user_id           70141
user_session     126306
event_month           5
time_zone             1
event_history    250075
date                152
hours                24
weekday               7
weeknumber           22
dtype: int64
```

Now the data is ready for Exploratory Analysis

3. Exploratory Analysis & Recommendations

Questions that will be answered from this dataset:

- The number of monthly|Weekday|Daily visits
- Visits per hour on weekdays
- Customer purchase funnel
- Conversion rates per month (The % purchases from all the visits that happens on that day)
- Cart Abandonment rate
- Number of orders per day|Average Basket Size
- Number of orders per Hour

In [28]:
```
#create a list with all the months & colors to use for different visualizations through this document
month_grp = ["10-2019", "11-2019", "12-2019", "01-2020", "02-2020"]
clr_table=["blue","orange","grey","green","red"]
```

3.1.Visits Analysis

I will check visits per month, per weekday, and per day. To identifiy the visits or sessions I will group the data by (event_month or weekday or date) & user_session

3.1.1 Visits per month

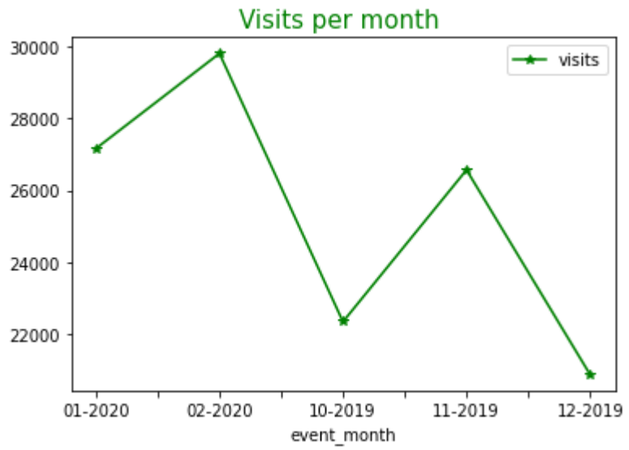
In [29]:
```
#group data by event_month & user_session & visualize
monthly_visits = all_months.groupby(['event_month'])['user_session'].agg(visits =('nunique'))
print(monthly_visits)
```

```
             visits
event_month
01-2020       27164
02-2020       29813
10-2019       22342
11-2019       26570
12-2019       20871
```

In [30]:
```
print("December 2019 21% compared to November, since then the traffic kept increasing")
#visualize visits per month
monthly_visits.sort_values(by=['event_month']).plot.line(marker='*',color='green')
```

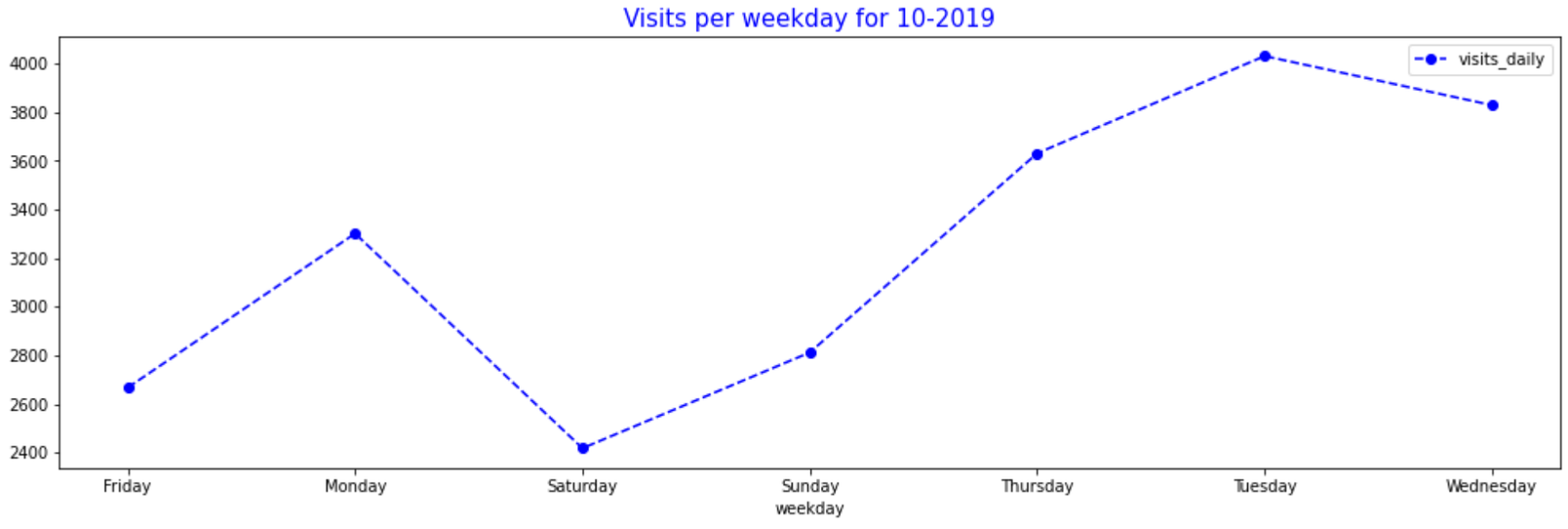
```
monthly_visits.set_ylabel = "visits"
plt.title('Visits per month ',fontsize = 15,color='green')
plt.rcParams['figure.figsize'] = (17,5)
```

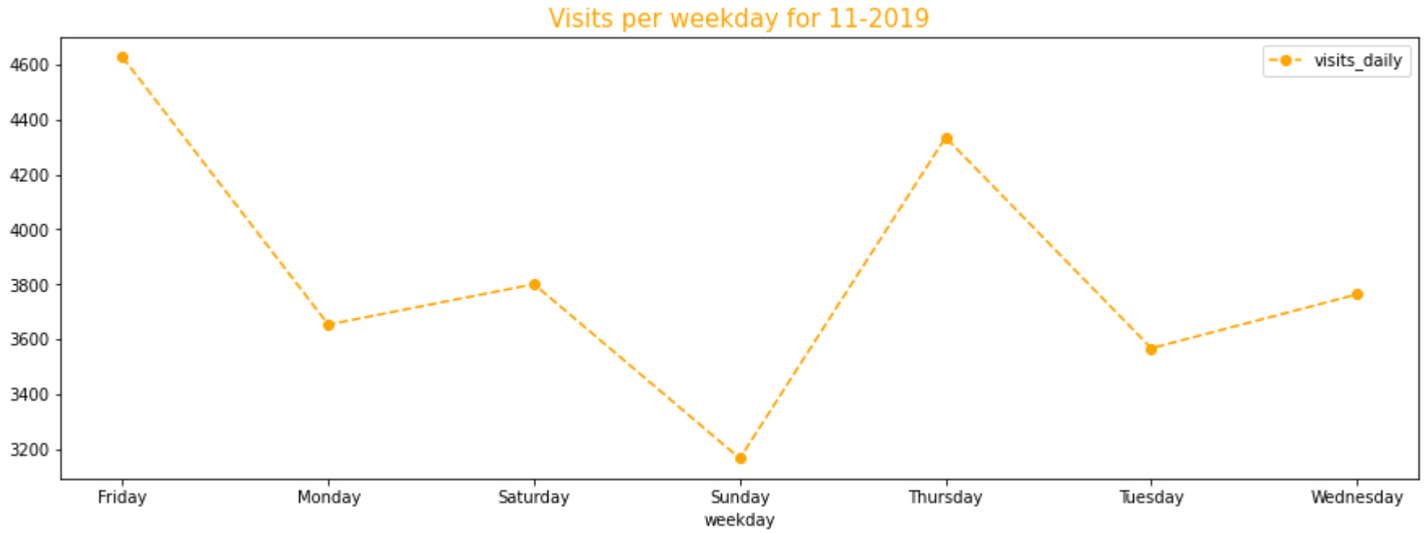
December 2019 21% compared to November, since then the traffic kept increasing

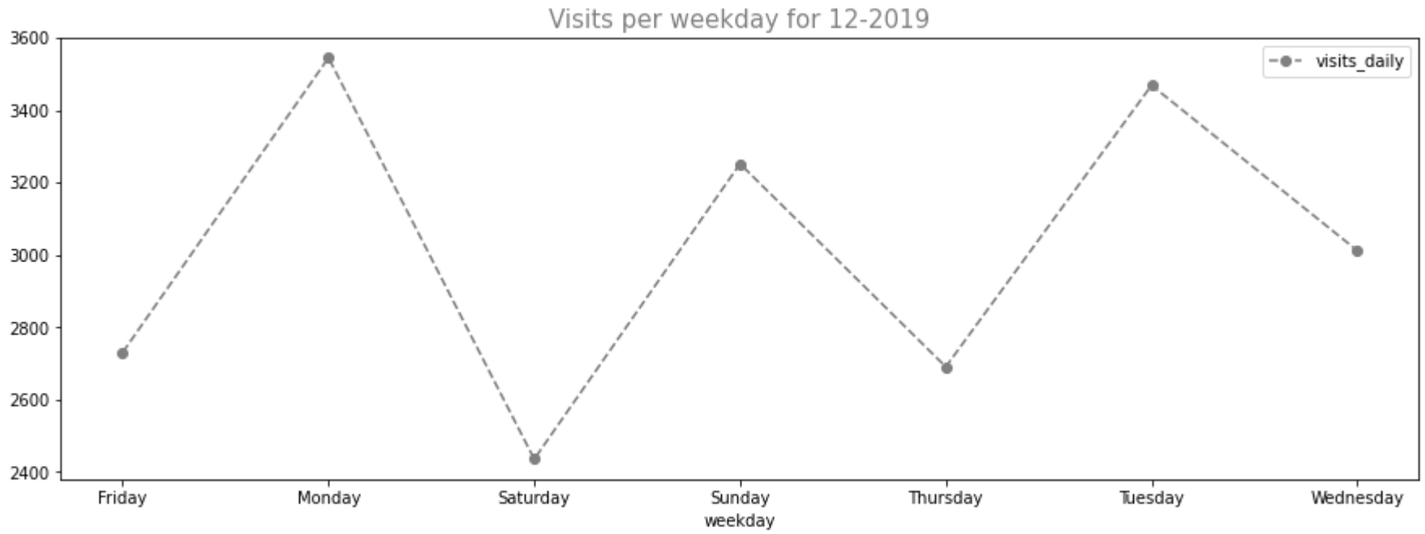


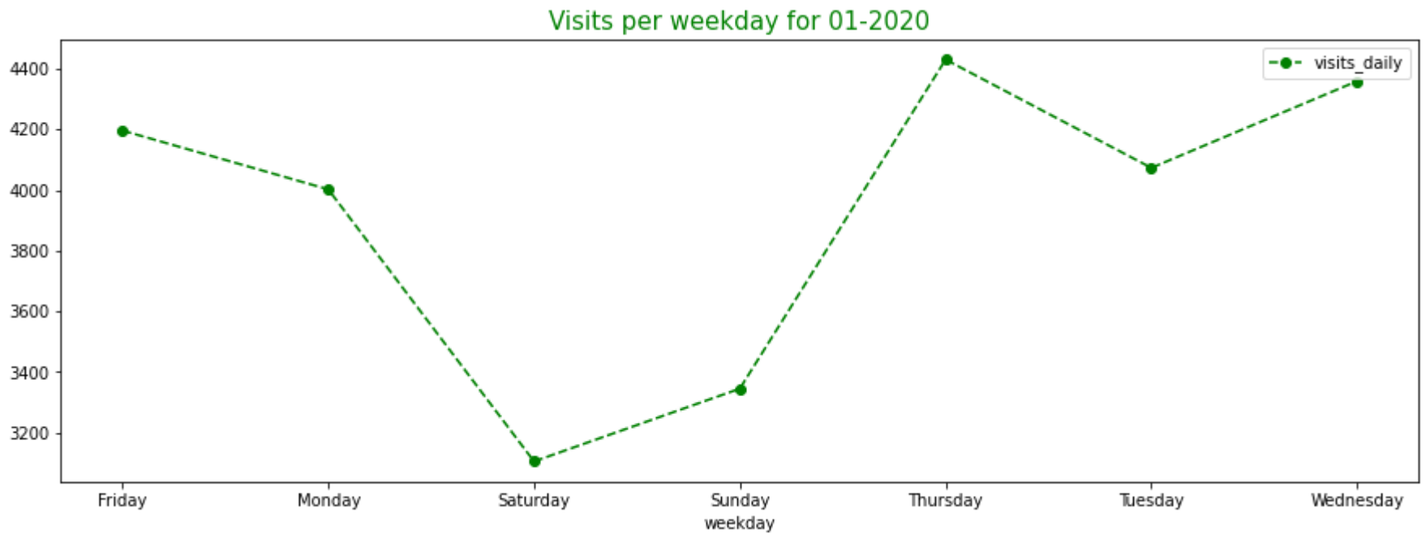
3.1.2. Visits per weekday

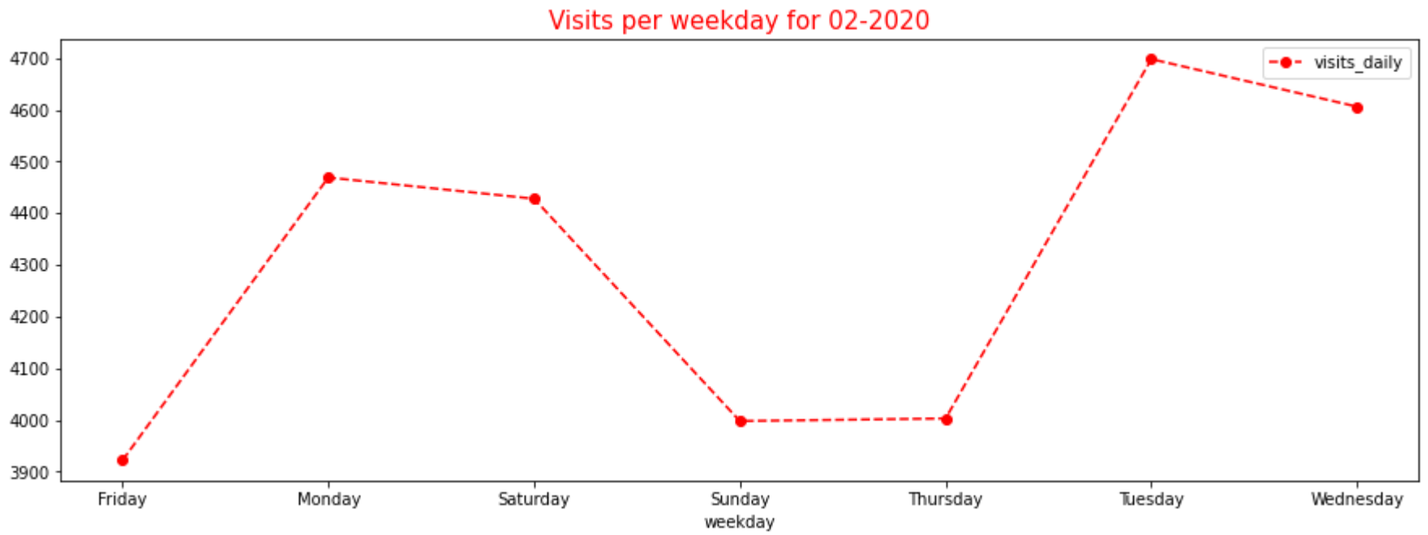
In [31]:
```
# Weekday visits for each month
# i goes from 1 to 5 because the data have only 5 months
for i in range(5):
    month=month_grp[i]
    color_t=clr_table[i]
    #filter the data per month
    month_query= all_months.query("event_month == @month")
    #define visits per weekday
    visits_weekday = month_query.groupby(['weekday'])['user_session'].agg(visits_daily =('nunique')).sort_values(by=['weekday'], ascending=True)
    #visualize the data
    visits_weekday.plot.line(linestyle = 'dashed',marker='o',color=color_t)
    visits_weekday.set_ylabel = "visits count"
    plt.title('Visits per weekday for '+ month,fontsize = 15,color=color_t)
    plt.rcParams['figure.figsize'] = (15,5)
```




Visits per weekday for 11-2019
visits_daily
4600
4400
4200
4000
3800
3600
3400
3200
Friday
Monday
Saturday
Sunday
Thursday
Tuesday
Wednesday
weekday

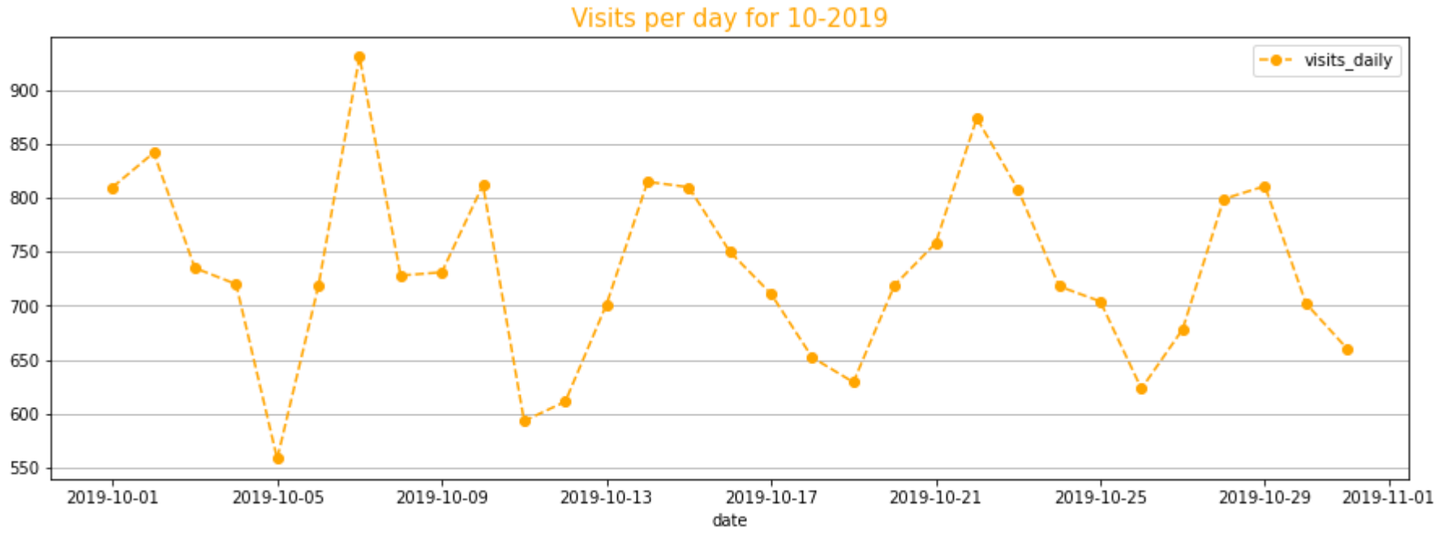

Visits per weekday for 12-2019
visits_daily
3600
3400
3200
3000
2800
2600
2400
Friday
Monday
Saturday
Sunday
Thursday
Tuesday
Wednesday
weekday

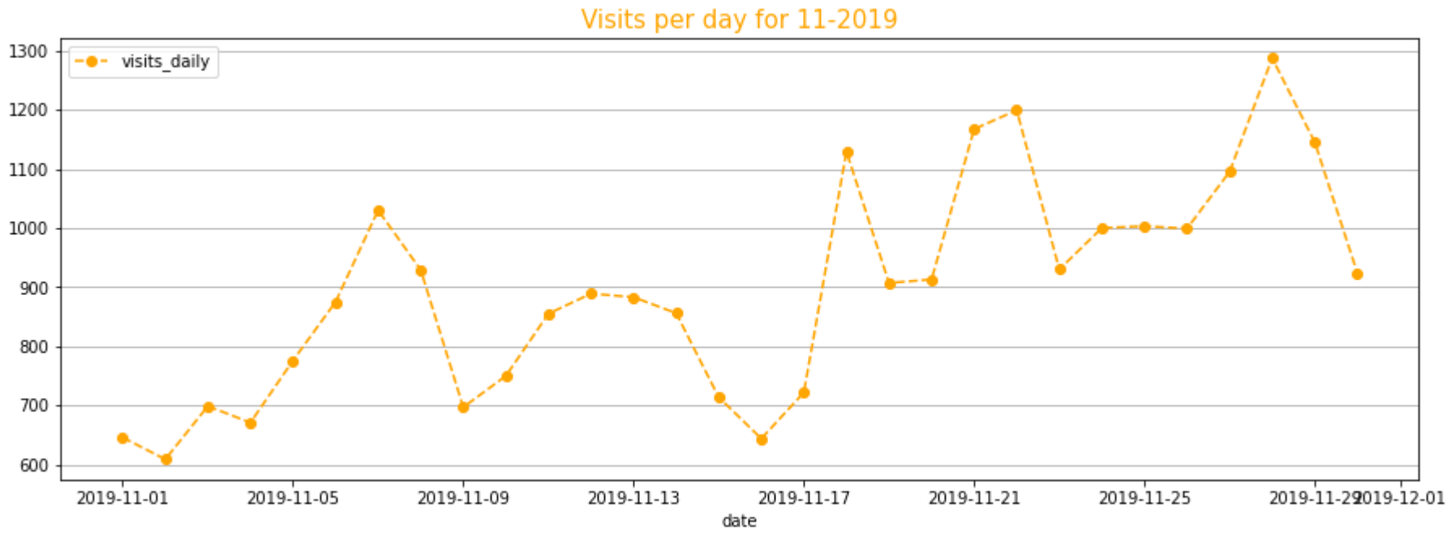

Visits per weekday for 01-2020
visits_daily
4400
4200
4000
3800
3600
3400
3200
Friday
Monday
Saturday
Sunday
Thursday
Tuesday
Wednesday
weekday

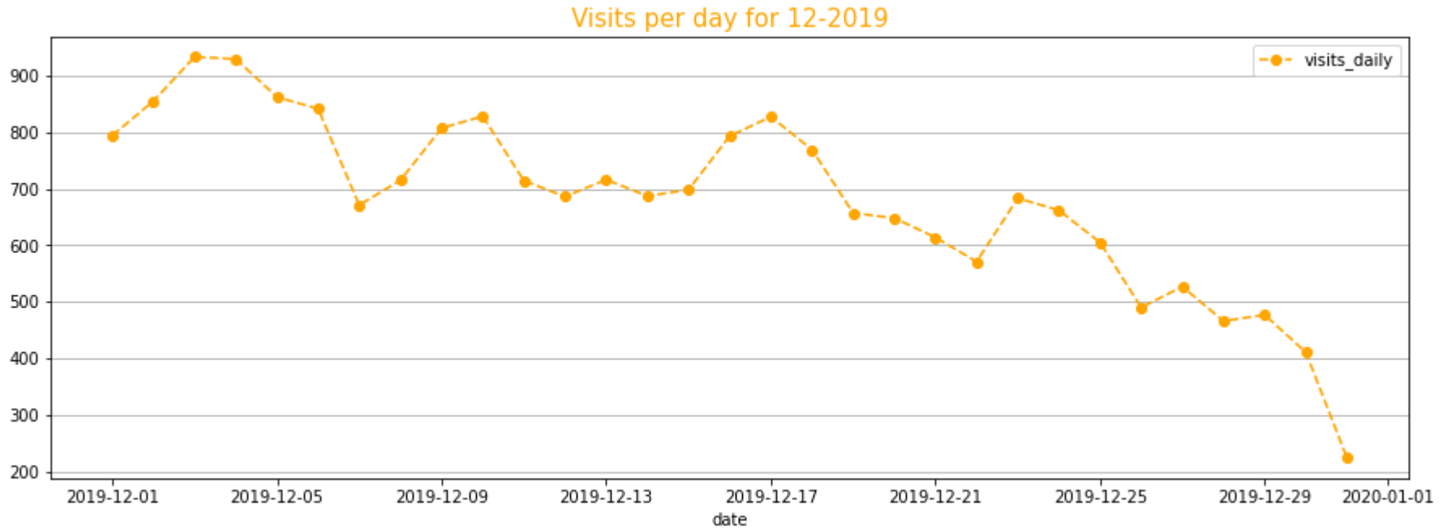


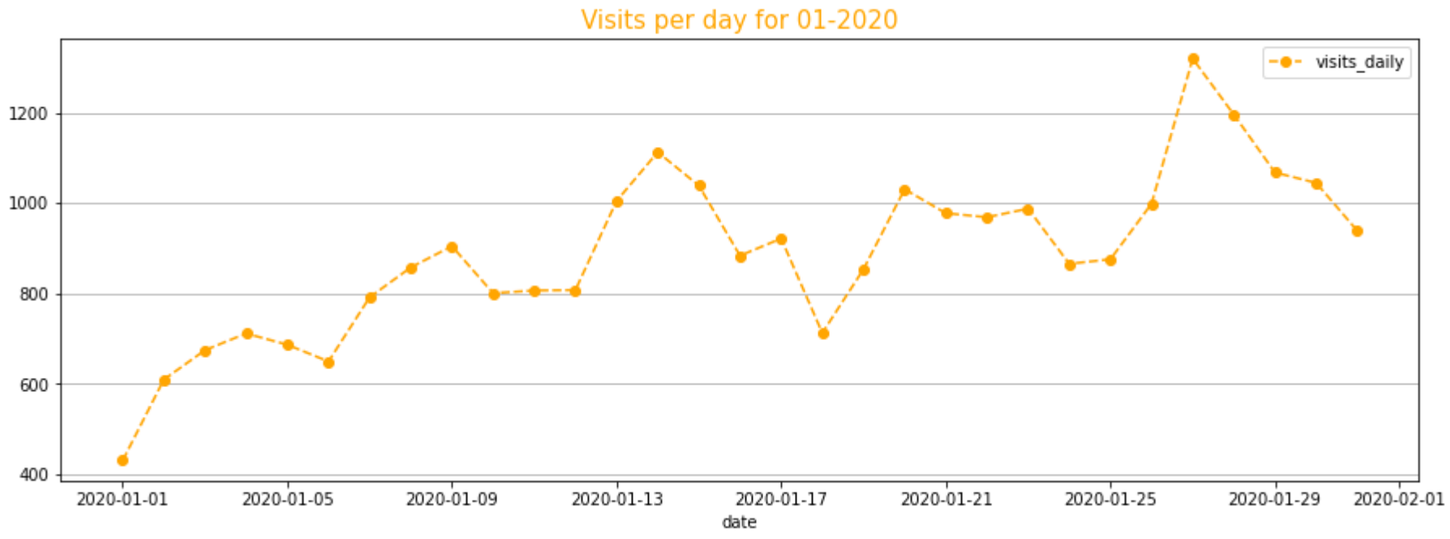
The highest traffic may be the normal flow of traffic on Tuesday, Wednesday, and Thursday. Feb has Tuesday and Wednesday. It may have been due to a special promotion, which made it different. Black Friday may have caused Friday to have a pick higher than usual. It is a bit different in December because of the high drop of traffic.

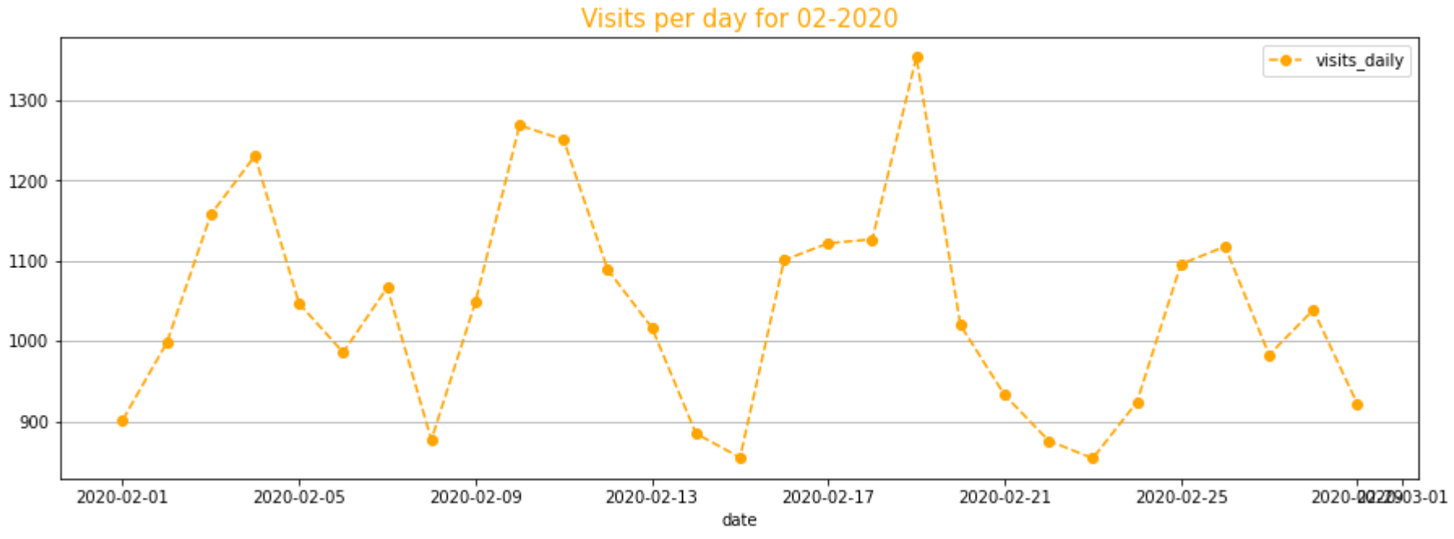
In [32]:
```
# i goes from 1 to 5 because the data have only 5 months
for i in range(5):
    month=month_grp[i]

    month_query= all_months.query("event_month == @month")
    #prepare data
    visits_day = month_query.groupby(['date'])['user_session'].agg(visits_daily =('nunique')).sort_values(by=['date'],
                                ascending=True)

    #visualize visits per day for each month
    visits_day.plot(linestyle = 'dashed',marker='o',color='orange')
    plt.grid(axis='y')
    visits_day.set_ylabel = "visits count"
    plt.title('Visits per day for '+ month,fontsize = 15,color='orange')
    plt.rcParams['figure.figsize'] = (15,5)
```




Visits per day for 11-2019
visits_daily
1300
1200
1100
1000
900
800
700
600
2019-11-01
2019-11-05
2019-11-09
2019-11-13
2019-11-17
2019-11-21
2019-11-25
2019-11-29
2019-12-01
date


Visits per day for 12-2019
visits_daily
900
800
700
600
500
400
300
200
2019-12-01
2019-12-05
2019-12-09
2019-12-13
2019-12-17
2019-12-21
2019-12-25
2019-12-29
2020-01-01
date


Visits per day for 01-2020
visits_daily
1200
1000
800
600
400
2020-01-01
2020-01-05
2020-01-09
2020-01-13
2020-01-17
2020-01-21
2020-01-25
2020-01-29
2020-02-01
date



The pick starts in the middle of the week. There was a pick on the 28th because of black Friday. A huge opportunity was missed because we saw a big drop in sales starting from the 23rd of December. Most people are coming back from holidays and traffic started to recover in January.There are different picks for Feb. That would be a special kind of promotion pushed by the marketing team.

Recommendation

- Use promotions on days where we expect more traffic (middle of the week). As for Saturday & Sunday, the Marketing team can think about engaging activities(such as events for example) that a customer would like to do on their rest day.
- Investigate the reason why there was a strong drop of traffic late December, and what did marketing miss?

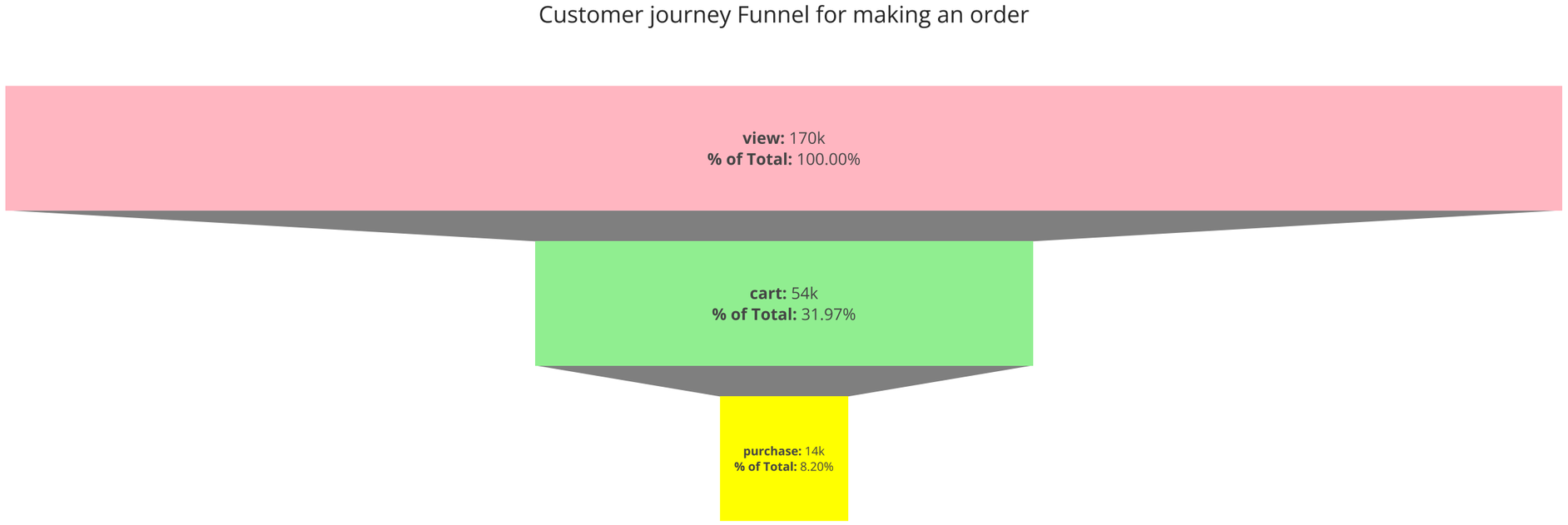
3.2. Customer Purchase Funnel

The user can do : view, remove_from_cart, cart,purshase

We want to know how many purchases were made in the end without considering those who removed a product after adding it to their cart.

In [33]:
```
#grouping and preparing data for funnel visualisation
journey_funnel=all_months[all_months['event_type']!='remove_from_cart'].groupby(['event_type'],as_index=False)['event_time'].count()
journey_funnel.columns=['event_type','# events']
journey_funnel.sort_values('# events', inplace=True,ascending=False)
journey_funnel.reset_index(drop=True,inplace=True)
journey_funnel['percent']=journey_funnel['# events'][/(journey_funnel['# events'][0].sum())*100
```

In [34]:
```
#plotly to visualise the funnel I prepared above
funnel_chart = gph.Figure(gph.Funnel(
    y = journey_funnel["event_type"],
    x = journey_funnel["# events"],
    customdata=journey_funnel["percent"],
    texttemplate= "<b>%{label}: </B>%{value:.2s}"+"<br><b>% of Total:</b> %{customdata:.2f}%",
    textposition='inside',
    marker = {"color": ["lightpink", "lightgreen", "yellow"]}
    ))
funnel_chart.update_yaxes(visible=False)
funnel_chart.update_layout(template='simple_white', title={'xanchor': 'center','yanchor': 'top', 'y':0.9, 'x':0.5,
                           'text':"Customer journey Funnel for making an order"})
funnel_chart.show()
```



Only 30% of the total visit added a product to their cart, and only 8.20% placed an order over the last 5 months.

3.3. Conversion Rate and Cart Abandonment Rate

The funnel of converion contains 3 steps:view, cart & purshase. However not every user follow these steps. Most users tend to only have a look at the product and its price. Checking the coversion rate accross each step will give us the idea of how many users actually purshased the products as apposed to how mayn times the products are viewed or added to the cart. How many products are actually purshased as opposed to number of products added to the cart.added to the cart.

Conversion rate is the overall percentage of visitors who made it the website and successfully made a purchase.

The shopping cart abandonment rate is the percentage of shoppers who add items to the cart but abandon it before purchasing, showing potential customers who were able to reach towards the end of the funneling model but gave up due to some reason. This rate shows how promising the checkout process could be.It can be calculated by dividing the number of completed purchases by the number of shopping carts created by customers. Then subtract this value from one and multiply by 100 to get the percentage.

To be able to check that, we need:

- People who viewed the item.
- People who added the item to cart.
- People who bought the item.

The function I created calculates both the conversion rate and abandonment rate for each month.

In [35]:
```
def converionrate_month(month):

#grouping and preparing data for funnel visualisation
```

```
    test=all_months.loc[all_months['event_month'].isin([month])].drop_duplicates()
    journey_funnel=test[test['event_type']!='remove_from_cart'].groupby(['event_type'],as_index=False)['event_time'].count()
    journey_funnel.columns=['event_type','# events']
    journey_funnel.sort_values('# events', inplace=True,ascending=False)
    journey_funnel.reset_index(drop=True,inplace=True)
    journey_funnel['percent']=journey_funnel['# events']/(journey_funnel['# events'][0].sum())*100
    print("")
    print("Conversion rate for customer funnel for " + month )
    print(round(journey_funnel,2))
    cart=journey_funnel['# events'][1]
    orders=journey_funnel['# events'][2]
    cart_ab_rate=str(round((1-(orders/cart))*100,2))
    print("Cart Abandonment ratio:" + cart_ab_rate +"%")
```

In [36]:
```
#calculation of the conversion rate of each month using the function conversion_rate
converionrate_month("10-2019")
converionrate_month("11-2019")
converionrate_month("12-2019")
converionrate_month("01-2020")
converionrate_month("02-2020")
```

```
Conversion rate for customer funnel for 10-2019
  event_type  # events  percent
0       view     29921   100.00
1       cart     10349    34.59
2   purchase      2555     8.54
Cart Abandonment ratio:75.31%

Conversion rate for customer funnel for 11-2019
  event_type  # events  percent
0       view     35425   100.00
1       cart     11230    31.70
2   purchase      3150     8.89
Cart Abandonment ratio:71.95%

Conversion rate for customer funnel for 12-2019
  event_type  # events  percent
0       view     27838   100.00
1       cart      8183    29.40
2   purchase      2107     7.57
Cart Abandonment ratio:74.25%

Conversion rate for customer funnel for 01-2020
  event_type  # events  percent
0       view     36303   100.00
1       cart     11459    31.56
2   purchase      3150     8.68
Cart Abandonment ratio:72.51%

Conversion rate for customer funnel for 02-2020
  event_type  # events  percent
0       view     39293   100.00
1       cart     11328    28.83
2   purchase      2829     7.20
Cart Abandonment ratio:75.03%
```

Most beauty brands have conversion rates of around 5.2%, though their cart abandonment rate is unfortunately 67%. As shown above, the conversion rate decreased from 8.54 to 7.20, while the cart abandonment rate decreased from 75% to 72%, from October to November 2019.There is a noticeable shift after November. Although cart abandonment rate increased over the following months, it didn't cross 75% as of Feb 2020, whereas conversion rate has fallen below all-time lows to a little above 7%. It could be due to a variety of reasons that conversion rates for medium-sized businesses are above average, one of which is that they are relatively less trafficked, but have a higher level of loyal customers.As a result, our primary goal should be to reduce cart abandonment rates, which in turn affect conversion rates indirectly.

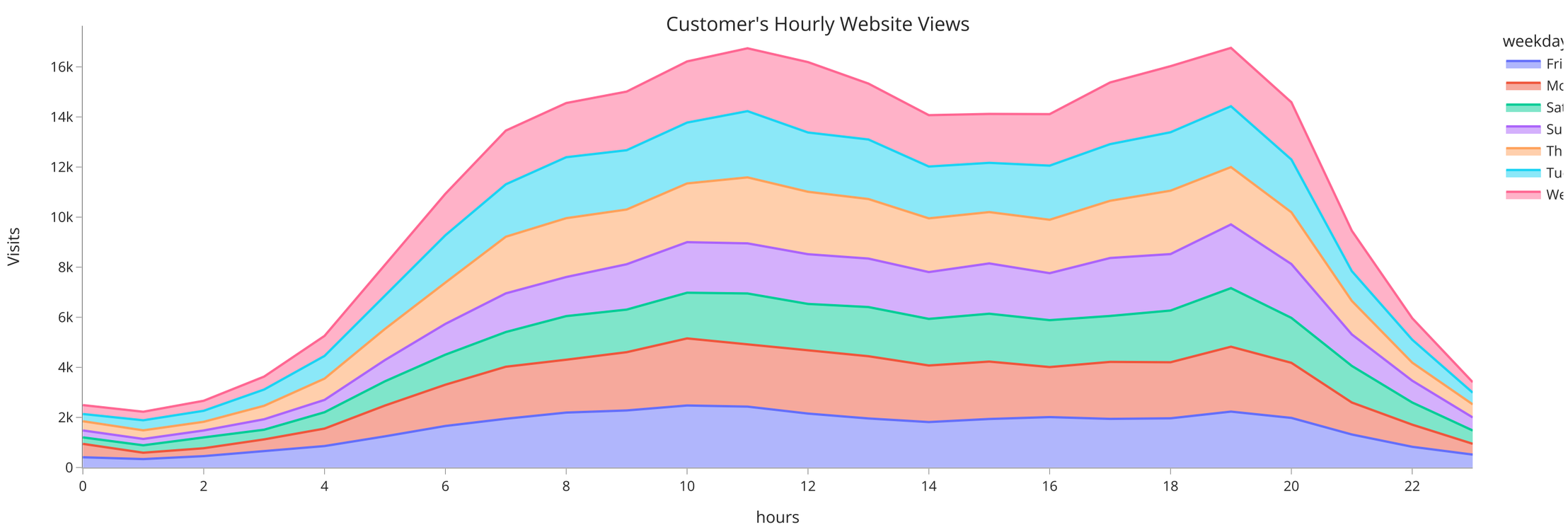
Recommendation:

- Offer customer re-targeting emails after they abandon a shopping cart along with promotional coupons of discount or other resources.
- Offering free shipping for purchases over a certain threshold and easy access to payment gateways with guest checkout functions.

3.4 Hourly Website Traffic

In [37]:
```
#Data preparation
#group data by hour & weekday
traffic_hourly=all_months.groupby(['hours','weekday'],as_index=False)['user_session'].count()
```

In [38]:
```
#Visualisation
hourly_views = px.area(traffic_hourly, x='hours', y="user_session",color='weekday')
hourly_views.update_layout(template='simple_white',
                title={'xanchor': 'center',
                        'yanchor': 'top',
                        'y':0.9,
                        'x':0.5,
                        'text':"Customer's Hourly Website Views"},
                xaxis = dict(
                    title_text='hours',
                    tickmode = 'linear',
                    tick0 = 0,
                    dtick = 2),
                 yaxis = dict(
                    title_text='Visits'))
hourly_views.show()
```



As we can see from the chart above, there are usually two picks within the day, on happens between 10AM and 2PM and the other one happens from 6pm to 8pm. These peaks can be generally translated to launch breaks, and after work hours.

Recommendation

With this information in mind, the marketing team can focus their efforts(especially conversion campaigns) on these hours to improve conversions of sales.

3.5.Daily Sales | Basket Size | Number of Orders

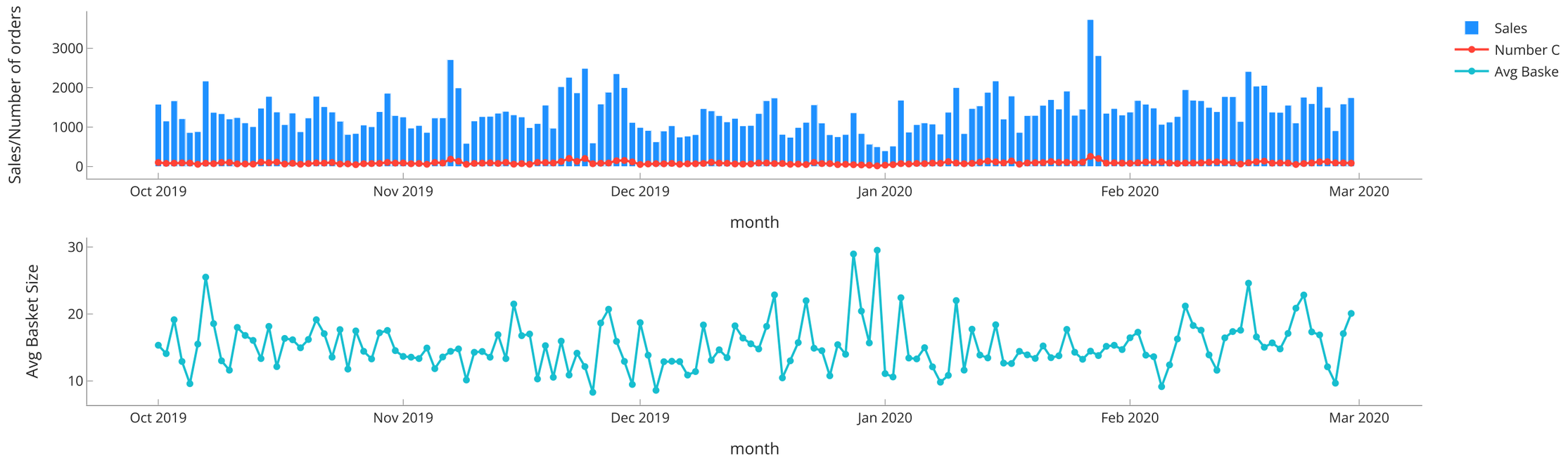
The code has been broken down into two functions to make it easier to understand. In one function, the task is to visualize only my data, while the other involves preparing it and displaying it.

In [39]:
```
def visualise_sales(value_1, value_2):
    #Visualisation
      present_chart= make_subplots(
          rows=2, cols=1,
          column_widths=[0.2],
          row_heights=[0.8, 0.8],
          specs=[[{"type": "Bar"}],[{"type": "Scatter"}]])

      present_chart.add_trace(gph.Bar(x=value_1['date'], y=value_1["price"],name='Sales',marker={'color': 'DodgerBlue'}),
                              row=1,col=1)
      present_chart.add_trace(gph.Scatter(x=value_2['date'], y=value_2['price'],mode='lines+markers',
                                      name='Number Of Orders', marker={'color': '#ff4236'}),row=1,col=1)


      present_chart.add_trace(gph.Scatter(x=value_2['date'], y=value_2['average_basket'],
                                      mode='lines+markers',name='Avg Basket Size',marker={'color':'#17becf'}),
                                      row=2,col=1)

      present_chart.update_layout(template='simple_white',
                                  title={'xanchor': 'center','yanchor': 'top',
                                         'y':1.0, 'x':1.0, 'text':"Daily Sales"})

      present_chart.update_yaxes(title_text='Sales/Number of orders',ticks="inside", row=1)
      present_chart.update_yaxes(title_text='Avg Basket Size',ticks="inside", row=2)
      present_chart.update_xaxes(title_text='month',ticks="inside")
      present_chart.show()
```

In [40]:
```
def show_sales(input_data,month_grp):

      print("Daily sales|Number of orders|AVG Basket value for all the months " )
      #prepare the data
      sales_revenue=input_data[input_data['event_type']=='purchase'].groupby(['date'],as_index=False)['price'].sum()
      orders_number=input_data[input_data['event_type']=='purchase'].groupby(['date'],as_index=False)['price'].count()
      orders_number['average_basket']=sales_revenue['price']/orders_number['price']
      sales_revenue.columns=['date','price']
      #call function to visualize the the different values
      visualise_sales(sales_revenue, orders_number)
```

In [41]:
```
#call function
show_sales(all_months,month_grp)
```

```
Daily sales|Number of orders|AVG Basket value for all the months
```



As we can see from the charts above, there are some peacks when sales were high. For example, this happend on Nov(6,7,22,24,28,29)th and Jan(27,28)th 2020. This might be due to some specific promotional campaigns. the number of sales was also high on these days but the average basket size is not that low for Nov(7,8,22,24)th 2019 and Jan(27,28)th 2020, by which we can infer that, even if there is any promotional camapigns running on these days, the dicount wasno't that low affecting the average basket size.

On the other hand we see that for 28,29th November we see higher basket size, and that could be explaind that the promotion was really attractive from the customer perspective.

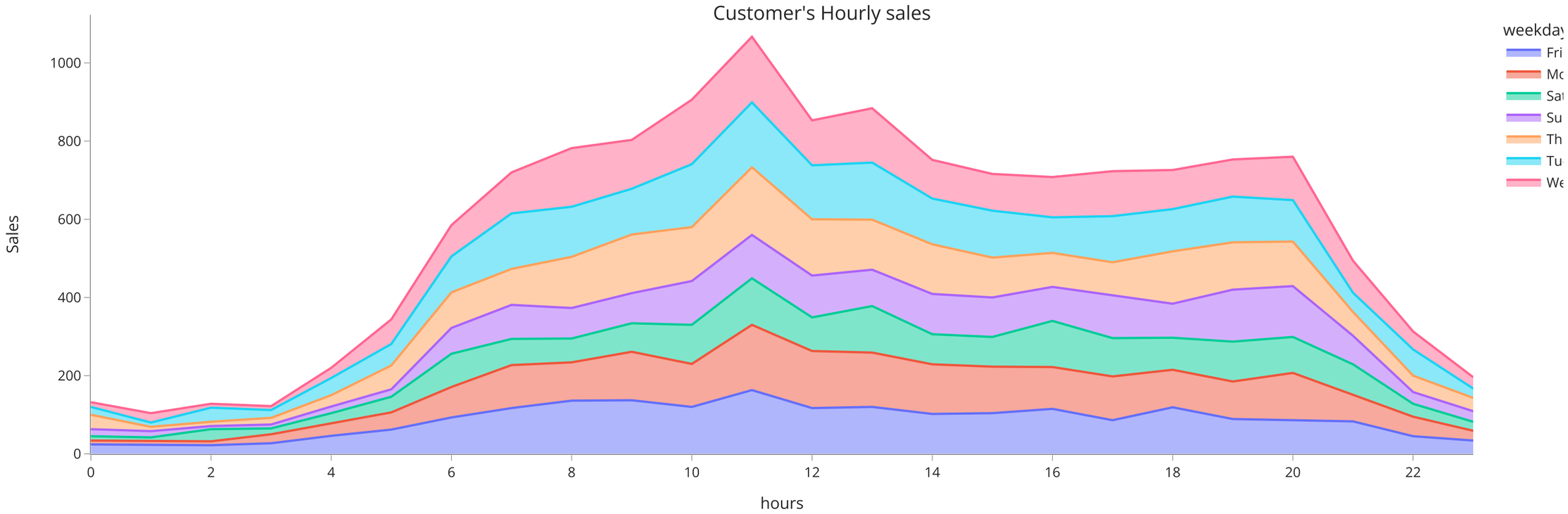
Recommendation

- During observed holidays, offer customized marketing campaigns and offers via online and offline means (emails, mobile notifications, etc).

3.6 Hourly Sales per weekday

In [42]:
```
#take only data with purshase event (sales)
orders= all_months.loc[all_months['event_type'].isin(['purchase'])].drop_duplicates()
#group purshase events by hour & weekday
hour_sales=orders.groupby(['hours','weekday'],as_index=False)['event_type'].count()
```

In [43]:
```
#Visualisation
hourly_sales = px.area(hour_sales, x='hours', y="event_type",color='weekday')
hourly_sales.update_layout(template='simple_white',
                title={'xanchor': 'center','yanchor': 'top','y':0.9,'x':0.5,'text': "Customer's Hourly sales"},
                xaxis = dict(title_text='hours', tickmode = 'linear', tick0 = 0, dtick = 2),
                yaxis = dict(title_text='Sales'))
hourly_sales.show()
```



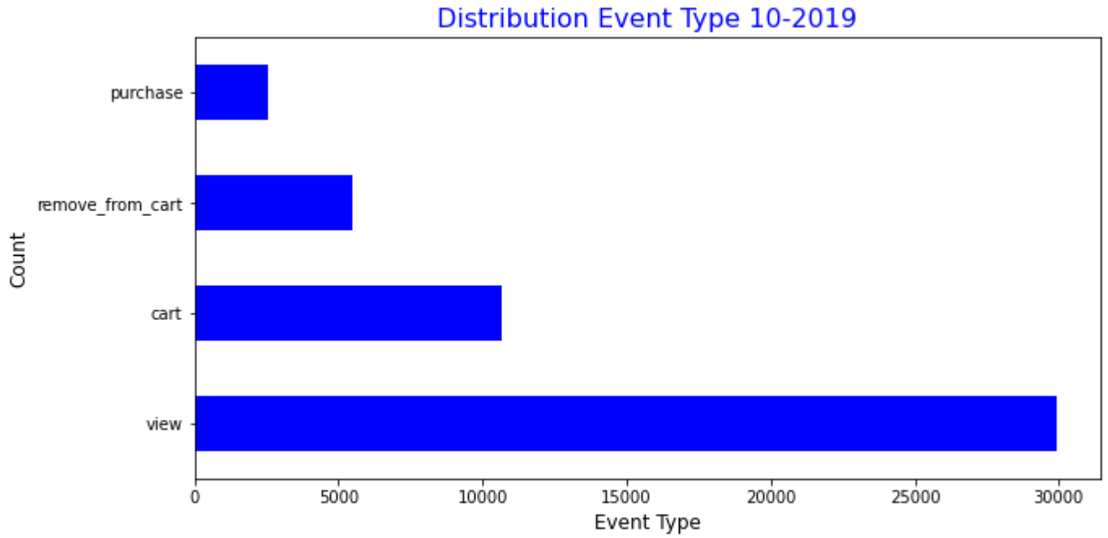
From this chart, we see that sales has a pick between 10am & 2pm, and there is another pick around 8pm that make it now more obvious to focus converting campaigns on this range of time.

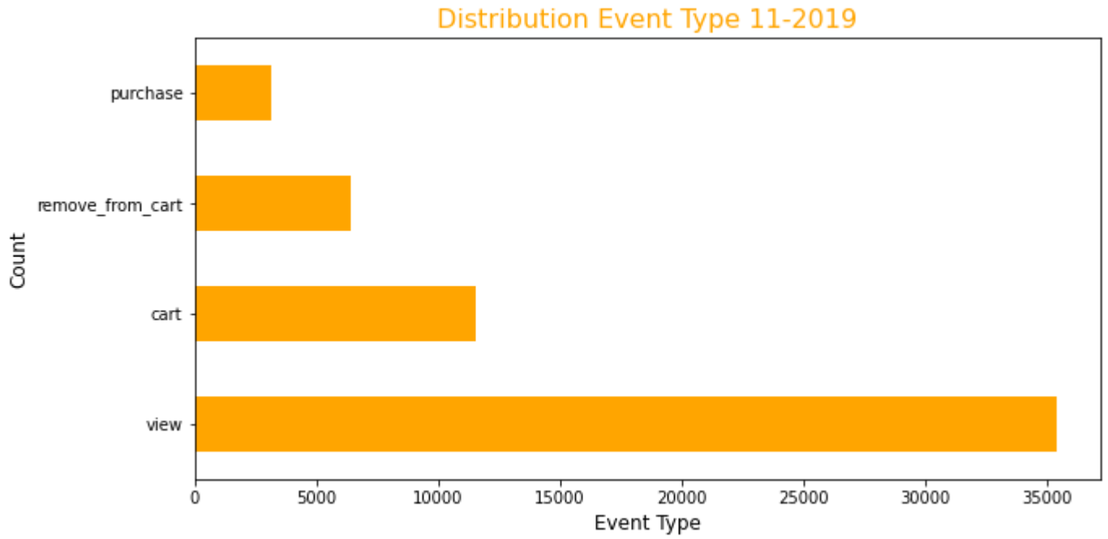
3.7. Investigating the low traffic in December

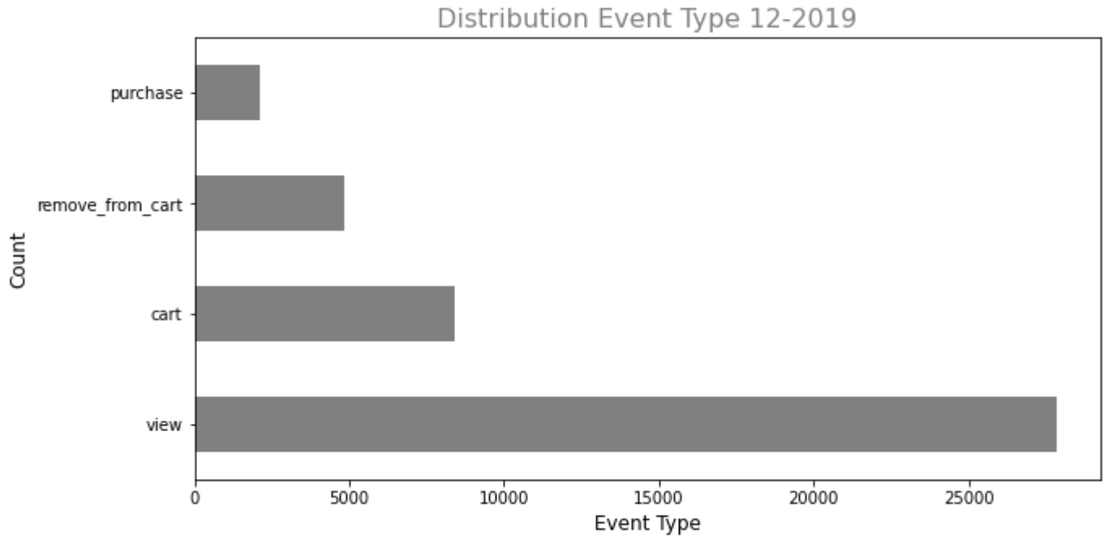
3.7.1.Distribution of event history across each month

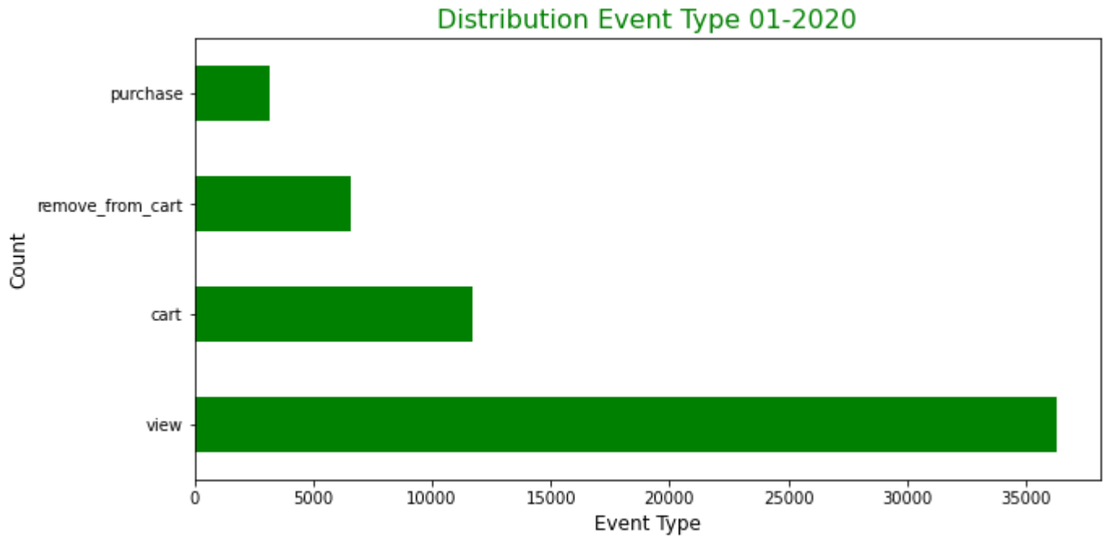
In [44]:
```
def show_distribtion(input_data,month_grp):

    i=len(month_grp)
    for i in range(5):
      month=month_grp[i]
      color_t=clr_table[i]
      #select the month
      month_event_distribution=input_data.query("event_month == @month")
      #prepare the data & Visualize
      month_event_distribution['event_type'].value_counts().head(50).plot.barh(figsize = (10,5),color =color_t,width = 0.5)
      plt.title('Distribution Event Type '+ month,fontsize = 16,color=color_t)
      plt.xlabel('Event Type',fontsize = 12)
      plt.ylabel('Count',fontsize = 12)

      plt.show()
```

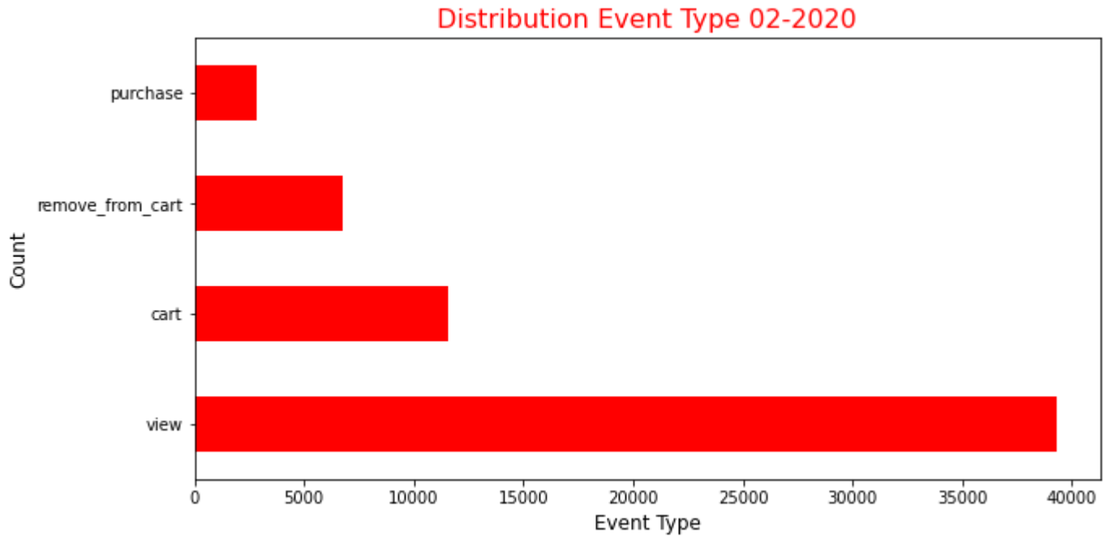
In [45]:
```
#call the function
show_distribtion(all_months,month_grp)
```


Distribution Event Type 10-2019
purchase
remove_from_cart
cart
view
Count
0
5000
10000
15000
20000
25000
30000
Event Type


Distribution Event Type 11-2019
purchase
remove_from_cart
cart
view
Count
0
5000
10000
15000
20000
25000
30000
35000
Event Type

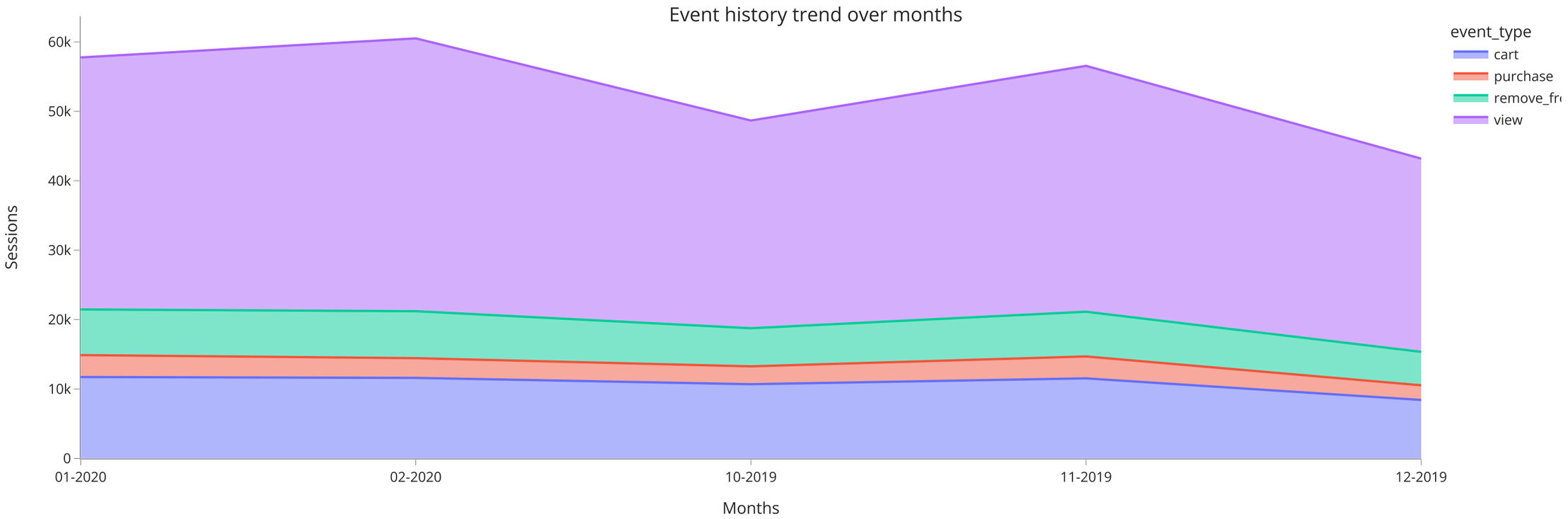

Distribution Event Type 12-2019
purchase
remove_from_cart
cart
view
Count
0
5000
10000
15000
20000
25000
Event Type





3.7.2.Event history trend over months

In [46]:
```
#Data preparation
#group data by hour & weekday
traffic_hourly=all_months.groupby(['event_month','event_type'],as_index=False)['user_session'].count()
#Visualisation
hourly_views = px.area(traffic_hourly, x='event_month', y="user_session",color='event_type')
hourly_views.update_layout(template='simple_white',
                title={'xanchor': 'center', 'yanchor': 'top','y':0.9,'x':0.5,'text':"Event history trend over months"},
                xaxis = dict(title_text='Months', tickmode = 'linear', tick0 = 0),
                yaxis = dict(title_text='Sessions'))
hourly_views.show()
```



The difference between making a purchase and removing products from the cart is obvious. It appears that there is especially a high increase in November, which may be due to "Black Friday," an informal term used to refer to the Friday following Thanksgiving in the US. During this time, several leading retailers are offering numerous promotions. So, we can conclude that the client did a decent job of targeting this crowd on this period of time and this can be used for future reference.

An additional obvious increase followed near the end of February, sometime before Valentine's Day and sometime after. Despite the fact that the days leading up to Christmas and New Year are considered crucial in terms of seasonality, there is a big drop in sales leading up to 31 Dec, 2019.

Consequently, customers may be driven towards other firms offering such convenient, "attractive" offers, as a result of poor marketing tailored just for these holiday events. There seems to be no change in reasoning from observing the abrupt dip that was added to the cart funnel while viewers did not see the same dive.

Recommendation

- Further investigation about Christmas and New Year holiday, because itt seems that the client was able to capture promotional events and holidays except the the end of December there was the big loss of opportunity, maybe that was because of something external, or the promotions offered werenot sufficient enough.

4. Conclusion & Recommendations

The client is already doing a good job, but there is still alwaya a window for improvement. I have already mentioned few recommnedations in the previous step.

Added to that I would like to highlight Order abandonment one more time which is an ongoing issue for all multichannel retail brands. However, because of the nature of cosmetics, which many shoppers prefer to try on first, this is an especially challenging issue for beauty retailers.

As a way to decrease the potential for abandonment, brands are encouraged to use personalization tactics that encourage shoppers to try new products, along with delivering a shopping experience that engages the customer and lets them know how much we value their business. And for the marketing tactics, I would suggest:

- Offering a pop-up discount may encourage a shopper to reconsider leaving their items behind when they exhibit behaviors that indicate they will leave, such as a sudden click to a different page. This approach will work best if it is tailored to the user.
- Recommend regular product replenishments: As beauty brands fully understand the average replenishment times of items such as foundation, lipstick, or eyeshadow, this data can be used to reengage customers who had added an item to their cart but then left the site.
- Notifying them via email that it's time to restock on a previous purchase will increase their likelihood of returning to purchase both items
- The fear of missing out (FOMO) can be a major motivating factor for customers who are unsure about whether they want to purchase a specific item. Notifying customers that stock is low may help them decide to purchase, since they may not get a chance to later.
- Retarget returning visitors: Often, shoppers add items to their cart, leave the website, and then come back at a later date. When the customer returns, deliver a message that reminds them of their basket details and encourages them to checkout (or continue shopping).
- Add an email to your email service provider (ESP) to remind customers to come back: If a customer has left items in their cart and has left the site entirely, trigger an email to remind them to come back. Ensure that you have rules that govern email opt-outs (or opt-ins), email frequency, and other factors that will ensure a "light touch" and avoid irritating your customers.

One more thing, in this document I didnot go through the customer analysis so I don't have a lot of insights there, but I would definitely recommand to check that as well, because it is really important part of the puzzle. And the more data we have about our customers, the more we can understand them. In this case, I would encourage my client to collect more customer data, or, if he already has it, he should share it with me since that would help me to better understand the audience.

5.Reference

Kaggle.com. 2022. eCommerce Events History in Cosmetics Shop. [online] Available at: https://www.kaggle.com/datasets/mkechinov/ecommerce-events-history-in-cosmetics-shop?select=2020-Jan.csv [Accessed 01 March 2022].