Introduction

The analytical task in our hand is for an international online store. We want to test a new *recommender system* by performing an A/B test - A is a controlled group without the new recommender system. While B is the treatment group, with the new recommender system. According to the technical description of the test, we expect an increase in all types of conversions in the online store (login to product page views, login to purchase etc) among 6000 new users which 15% of them are from EU region.

Plan of Work

- We are going to start our project by preprocessing
    1. Loading the necessary data and libraries
    2. Overviewing the datasets
    3. Look for problematic data and dealing with it
    4. Converting data-types
    5. Summary
- Then we proceed with EDA:
    1. We study the given data
    2. Support the studies with visual graphs
    3. Perform alterations to our dataframes for future studies if needed
    4. Make sure that our data is ready for an A/B test
    5. Conclusions
- A/B Test:
    1. Perform the A/B testing
    2. Reach to conclusions
- General summary and recommendations

1 PART 1: PREPROCESSING

Lets start with loading the necessary libraries and datasets

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from plotly import graph_objects as go
import seaborn as sns
import math
from scipy import stats as st
```

Next: Loading the first dataset, *marketing_events* which includes data about dates and regions of marketing events during the year.

In [2]:

```
marketing_events = pd.read_csv('/datasets/ab_project_marketing_events_us.csv')
marketing_events.head()
```

Out[2]:

	name	regions	start_dt	finish_dt
0	Christmas&New Year Promo	EU, N.America	2020-12-25	2021-01-03
1	St. Valentine's Day Giveaway	EU, CIS, APAC, N.America	2020-02-14	2020-02-16
2	St. Patric's Day Promo	EU, N.America	2020-03-17	2020-03-19
3	Easter Promo	EU, CIS, APAC, N.America	2020-04-12	2020-04-19
4	4th of July Promo	N.America	2020-07-04	2020-07-11

In [3]:

```
marketing_events.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   name       14 non-null     object
 1   regions    14 non-null     object
 2   start_dt   14 non-null     object
 3   finish_dt  14 non-null     object
dtypes: object(4)
memory usage: 576.0+ bytes
```

We have no missing values or problematic data here, pretty simple dataframe. Lets convert the dates into datetime[Day] and move on.

In [4]:

```
# converting the start and finish dates of each event into datetime[day]
marketing_events['start_dt'] = marketing_events['start_dt'].astype('datetime64[D]')
marketing_events['finish_dt'] = marketing_events['finish_dt'].astype('datetime64[D]')

# checking for results
display(marketing_events.iloc[0])
marketing_events.info()
```

```
name         Christmas&New Year Promo
regions                 EU, N.America
start_dt          2020-12-25 00:00:00
finish_dt         2021-01-03 00:00:00
Name: 0, dtype: object

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   name       14 non-null     object
 1   regions    14 non-null     object
 2   start_dt   14 non-null     datetime64[ns]
 3   finish_dt  14 non-null     datetime64[ns]
dtypes: datetime64[ns](2), object(2)
memory usage: 576.0+ bytes
```

Do we have any *duplicates*?

In [5]:

```
marketing_events.duplicated().sum() # checking for duplicates
```

Out[5]:

```
0
```

No duplicates!

Next: Loading the *new_users* dataframe:

In [6]:

```
new_users = pd.read_csv('/datasets/final_ab_new_users_upd_us.csv')
display(new_users.head())
new_users.info()
```

	user_id	first_date	region	device
0	D72A72121175D8BE	2020-12-07	EU	PC
1	F1C668619DFE6E65	2020-12-07	N.America	Android
2	2E1BF1D4C37EA01F	2020-12-07	EU	PC
3	50734A22C0C63768	2020-12-07	EU	iPhone
4	E1BDDCE0DAFA2679	2020-12-07	N.America	iPhone

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 58703 entries, 0 to 58702
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   user_id     58703 non-null  object
 1   first_date  58703 non-null  object
 2   region      58703 non-null  object
 3   device      58703 non-null  object
dtypes: object(4)
memory usage: 1.8+ MB
```

Again, we don't seem to have any missing values here, lets have a deeper look at this dataframe:

In [7]:

```
display(new_users['device'].unique()) # looking into the sign-up devices we got
display(new_users['region'].unique()) # looking into the regions we got
new_user_count = new_users['user_id'].nunique() # the number of unique users

if new_user_count == 58703: # if we have as much users as rows then there's no dups
    print("There is 58703 users, no duplicates!")
else:
    print(new_user_count, '- careful, we got duplicates!')
```

```
array(['PC', 'Android', 'iPhone', 'Mac'], dtype=object)
```

```
array(['EU', 'N.America', 'APAC', 'CIS'], dtype=object)
```

```
There is 58703 users, no duplicates!
```

Everything looks fine except for dates, which need to be converted into datetimes:

In [8]:

```
new_users['first_date'] = new_users['first_date'].astype('datetime64[D]')
display(new_users.iloc[0])
new_users.info()
```

```
user_id          D72A72121175D8BE
first_date    2020-12-07 00:00:00
region                         EU
device                         PC
Name: 0, dtype: object

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 58703 entries, 0 to 58702
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype         
---  ------      --------------  -----         
 0   user_id     58703 non-null  object        
 1   first_date  58703 non-null  datetime64[ns]
 2   region      58703 non-null  object        
 3   device      58703 non-null  object        
dtypes: datetime64[ns](1), object(3)
memory usage: 1.8+ MB
```

Next: Loading the events dataset, this dataset has all the events that were performed during the test period.

In [9]:

```
events = pd.read_csv('/datasets/final_ab_events_upd_us.csv')
display(events.head())
events.info()
```

	user_id	event_dt	event_name	details
0	E1BDDCE0DAFA2679	2020-12-07 20:22:03	purchase	99.99
1	7B6452F081F49504	2020-12-07 09:22:53	purchase	9.99
2	9CD9F34546DF254C	2020-12-07 12:59:29	purchase	4.99
3	96F27A054B191457	2020-12-07 04:02:40	purchase	4.99
4	1FD7660FDF94CA1F	2020-12-07 10:15:09	purchase	4.99

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 423761 entries, 0 to 423760
Data columns (total 4 columns):
 #   Column      Non-Null Count   Dtype  
---  ------      --------------   -----  
 0   user_id     423761 non-null  object 
 1   event_dt    423761 non-null  object 
 2   event_name  423761 non-null  object 
 3   details     60314 non-null   float64
dtypes: float64(1), object(3)
memory usage: 12.9+ MB
```

Surprise, suprise! Off the bat we can see alot of missing values, lets investigate it.

In [10]:

```
display(events[events['event_name'] == 'purchase']['user_id'].count())
display(events[events['event_name'] != 'purchase']['user_id'].isna().count())
```

```
60314

363447
```

After investigating, we notice that only rows with *event_name* as **purchase** have **details** filled in, this means missing values here were intentional and we can move on.

Lets convert dates into datetime here as well:

In [11]:

```
events['event_dt'] = events['event_dt'].astype('datetime64[D]')
print(events['event_dt'].min()) # to check if we only have events from 7th of Dec and on
display(events.iloc[0])
events.info()
```

```
2020-12-07 00:00:00

user_id          E1BDDCE0DAFA2679
event_dt      2020-12-07 00:00:00
event_name               purchase
details                     99.99
Name: 0, dtype: object

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 423761 entries, 0 to 423760
Data columns (total 4 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   user_id     423761 non-null  object
 1   event_dt    423761 non-null  datetime64[ns]
 2   event_name  423761 non-null  object
 3   details     60314 non-null   float64
dtypes: datetime64[ns](1), float64(1), object(2)
memory usage: 12.9+ MB
```

Lets have a final look at the **event_name** column and check for any duplicates:

In [12]:

```
display(events['event_name'].unique())
events.duplicated().sum()
```

```
array(['purchase', 'product_cart', 'product_page', 'login'], dtype=object)
```

Out[12]:

```
2
```

2 duplicates - means that we have 2 events that happened for the same user, at the same time with the same event name - could be a double click or an error, either way we can live with it.

Finally, lets load the last dataset, it includes data about all the participants that participated in the A/B test.

In [13]:

```
participants = pd.read_csv('/datasets/final_ab_participants_upd_us.csv')
display(participants.head())
participants.info()
```

	user_id	group	ab_test
0	D1ABA3E2887B6A73	A	recommender_system_test
1	A7A3664BD6242119	A	recommender_system_test
2	DABC14FDDFADD29E	A	recommender_system_test
3	04988C5DF189632E	A	recommender_system_test
4	4FF2998A348C484F	A	recommender_system_test

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14525 entries, 0 to 14524
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   user_id  14525 non-null  object
 1   group    14525 non-null  object
 2   ab_test  14525 non-null  object
dtypes: object(3)
memory usage: 340.6+ KB
```

Great! no missing values. Lets investigate further more.

In [14]:

```
display(participants['group'].value_counts()) # checking if we have only groups A and B
display(participants['ab_test'].value_counts()) # checking that we have only one test
display(participants['user_id'].nunique()) # counting unique user ids
```

```
A    8214
B    6311
Name: group, dtype: int64
```

```
interface_eu_test          10850
recommender_system_test     3675
Name: ab_test, dtype: int64
```

```
13638
```

Bad news - *ab_test* column seems to contain more than one test: *recommender_system_test* and *interface_eu_test*. Lets keep this in mind and move on. Looking for duplicates:

In [15]:

```
participants.duplicated().sum() # looking for duplicates
```

Out[15]:

```
0
```

No dups!

1.1 SUMMARY

- We have 14 marketing events annually.
- 58703 new users signed-up between 7th of Dec till 21st of Dec
- Devices used for sign-up: PC, Android, I-phone, Mac
- Regions new users come from: EU, NA, APAC, CIS
- During the test period, we had almost 424,000 user events, among them around 60,000 purchases.
- 13638 participants participated in the test

2 PART 2: EDA

2.1 General data study

Lets start the study by looking at what events are residents of the test time period, do we have any? What does it mean to have such marketing events within the test period and how does it affect our results?

In [16]:

```
display(marketing_events[marketing_events['start_dt'] >= '2020-12-01']) # checking what eve
```

	name	regions	start_dt	finish_dt
0	Christmas&New Year Promo	EU, N.America	2020-12-25	2021-01-03
10	CIS New Year Gift Lottery	CIS	2020-12-30	2021-01-07

Problems! We can see that not one, but 2 marketing events are within the period of our test. This could lead to alot of errors to our conclusions because any changes during the test could be due to a marketing event and not due to the *recommender system*. We should perform our test in normal circumstances without any disturbance.

In [18]:

```
display(new_users[new_users['region'] == 'EU'].shape[0]) # counting EU new users
```

```
43396
```

This could be concerning because this percent is way above 15% of the users, could it change when we start talking about *participants* rather than *users*? Lets investigate more:

Next: we seperate the tests into different dataframes, to be able to continue with our main test.

In [19]:

```
# this dataframe will have all interface_eu participants
eu_ui_test = participants[participants['ab_test'] == 'interface_eu_test']
# this dataframe will have all recommender_sys participants
recommender_test = participants[participants['ab_test'] == 'recommender_system_test']
```

In [20]:

```
print('Interface EU test has', eu_ui_test['user_id'].count(), 'participants')
print('Recommender System test has', recommender_test['user_id'].count(), 'participants')
```

```
Interface EU test has 10850 participants
Recommender System test has 3675 participants
```

Lets get rid of the second dataframe, but first; lets check if there's an intersection between the tests, in case we have participants that belong to both tests, we'll have to also get rid of them as well, since their existance may cause disturbance to our conclusions - changes to this group can possibly be due to the second test (interface_eu), thus disturbing the main test results.

In [21]:

```
# we find the intersection using the merge method with inner as how to merge
intersection = recommender_test.merge(eu_ui_test, how='inner', on='user_id')
intersection.info()
intersection.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 887 entries, 0 to 886
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   user_id    887 non-null    object
 1   group_x    887 non-null    object
 2   ab_test_x  887 non-null    object
 3   group_y    887 non-null    object
 4   ab_test_y  887 non-null    object
dtypes: object(5)
memory usage: 41.6+ KB
```

Out[21]:

	user_id	group_x	ab_test_x	group_y	ab_test_y
0	DABC14FDDFADD29E	A	recommender_system_test	A	interface_eu_test
1	04988C5DF189632E	A	recommender_system_test	A	interface_eu_test
2	B3A2485649E4A012	A	recommender_system_test	A	interface_eu_test
3	EAFB9027A27D510C	B	recommender_system_test	A	interface_eu_test
4	5D5E6EE92AF6E9E0	B	recommender_system_test	B	interface_eu_test

We have 887 users that belong to both tests, what do we do with them? Well, in my opinion, It is not very healthy for our conclusion to have users that belong to 2 tests;

1. because we can't tell then which test actually contributed to the changes, if there was any.
2. There could've been an error or a mistake that led to all this and we can't know for sure.

Therefore, we are going to drop the intersection off our final data.

In [22]:

```
intersect_list = [user_id for user_id in intersection['user_id']] # making a list of user_i
recommender_test = recommender_test[~(recommender_test['user_id'].isin(intersect_list))] #
recommender_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2788 entries, 0 to 3674
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   user_id  2788 non-null   object
 1   group    2788 non-null   object
 2   ab_test  2788 non-null   object
dtypes: object(3)
memory usage: 87.1+ KB
```

In [23]:

```
# the reviewer's code:
participants.groupby('user_id').agg({'ab_test':'nunique'}).reset_index().query('ab_test > 1
```

Out[23]:

887

In [24]:

```
recommender_test['user_id'].nunique()
```

Out[24]:

2788

We are left with 2788 participants - **way** below the 6000 participants mentioned in the technical description.

Now that we have all our necessary participants, lets merge our dataframe with other dataframes to have all the information in one place

In [25]:

```
# merging with new users
final_df = recommender_test.merge(new_users, how = 'left', on = 'user_id')
final_df.info()
final_df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2788 entries, 0 to 2787
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype         
---  ------      --------------  -----         
 0   user_id     2788 non-null   object        
 1   group       2788 non-null   object        
 2   ab_test     2788 non-null   object        
 3   first_date  2788 non-null   datetime64[ns]
 4   region      2788 non-null   object        
 5   device      2788 non-null   object        
dtypes: datetime64[ns](1), object(5)
memory usage: 152.5+ KB
```

Out[25]:

	user_id	group	ab_test	first_date	region	device
0	D1ABA3E2887B6A73	A	recommender_system_test	2020-12-07	EU	PC
1	A7A3664BD6242119	A	recommender_system_test	2020-12-20	EU	iPhone
2	4FF2998A348C484F	A	recommender_system_test	2020-12-20	EU	Mac
3	7473E0943673C09E	A	recommender_system_test	2020-12-16	EU	iPhone
4	C46FE336D240A054	A	recommender_system_test	2020-12-17	EU	iPhone

In [26]:

```
# merging with events to get all the events that belong to our participants
final_df = final_df.merge(events, how = 'left', on = 'user_id')
final_df.info()
final_df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18194 entries, 0 to 18193
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   user_id     18194 non-null  object
 1   group       18194 non-null  object
 2   ab_test     18194 non-null  object
 3   first_date  18194 non-null  datetime64[ns]
 4   region      18194 non-null  object
 5   device      18194 non-null  object
 6   event_dt    18194 non-null  datetime64[ns]
 7   event_name  18194 non-null  object
 8   details     2390 non-null   float64
dtypes: datetime64[ns](2), float64(1), object(6)
memory usage: 1.4+ MB
```

Out[26]:

	user_id	group	ab_test	first_date	region	device	event_dt	e
0	D1ABA3E2887B6A73	A	recommender_system_test	2020-12-07	EU	PC	2020-12-07	
1	D1ABA3E2887B6A73	A	recommender_system_test	2020-12-07	EU	PC	2020-12-07	p
2	D1ABA3E2887B6A73	A	recommender_system_test	2020-12-07	EU	PC	2020-12-07	pr
3	D1ABA3E2887B6A73	A	recommender_system_test	2020-12-07	EU	PC	2020-12-07	
4	A7A3664BD6242119	A	recommender_system_test	2020-12-20	EU	iPhone	2020-12-20	pr

We have 2788 unique user ids for our A/B test with 18,194 events, lets seperate the groups of the test into 2 dataframes, one for group **A** and one for group **B**:

In [27]:

```
rec_sys_a = final_df[final_df['group'] == 'A']
rec_sys_b = final_df[final_df['group'] == 'B']
print('Group A has:', rec_sys_a['user_id'].nunique(), 'unique users\nGroup B has:', rec_sys
print('Group A has:', rec_sys_a['user_id'].count(), 'events\nGroup B has:', rec_sys_b['user
```

```
Group A has: 2082 unique users
Group B has: 706 unique users
Group A has: 14215 events
Group B has: 3979 events
```

We have removed the users belonging to the eu_interface_test, including the intersection of the tests. Thus, we

are left with 2082 users of group A and 706 users of group B.

We already know that we don't have users that belong to both groups, A and B. Why? Because we already counted the total number of unique users in the main group and it was 2788, then we counted the number of unique users in each group; 2082 in A, 706 in B meaning their sum is 2788 - which leads us to think that there can't be a user with two groups. But to make sure of that, lets look at the intersection of the samples

In [28]:

```
intersection = rec_sys_a.merge(rec_sys_b, how = 'inner', on = 'user_id')
intersection.shape[0]
```

Out[28]:

```
0
```

2.2 Conversion rates

Below we'll calculate the conversion rates of each group - including the conversion rate for both groups together:

(*conversion rate is the rate of first login to first purchase, rate of users to customers*)

In [29]:

```
# For both groups with only users that signed up aft
user_purchases = final_df[(final_df['event_name'] == 'purchase') & (final_df['first_date']
user_purchases_min = user_purchases.groupby("user_id")['event_dt'].min()
total_participants = recommender_test['user_id'].nunique()
total_purchaser = user_purchases_min.count()

print('General conversion rate is: {:.2f}'.format(total_purchaser/total_participants))
```

```
General conversion rate is: 0.30
```

In [30]:

```
# For group A
# finding how many people did a purchase in the store from group A
group_A_purchases = rec_sys_a[(rec_sys_a['event_name'] == 'purchase') & (rec_sys_a['first_d
group_A_purchases_min = group_A_purchases.groupby('user_id')['event_dt'].min()
total_participants = rec_sys_a['user_id'].nunique() # getting unique ids of purchasers
total_purchaser = group_A_purchases_min.count() # all users of group A

print('Group A conversion rate is: {:.2f}'.format(total_purchaser/total_participants))
```

```
Group A conversion rate is: 0.31
```

In [31]:

```
# For group B
# finding how many people did a purchase in the store from group B
group_B_purchases = rec_sys_b[(rec_sys_b['event_name'] == 'purchase') & (rec_sys_b['first_d
group_B_purchases_min = group_B_purchases.groupby('user_id')['event_dt'].min()
total_participants = rec_sys_b['user_id'].nunique() # getting unique ids of purchasers
total_purchaser = group_B_purchases_min.count() # all users of group B

print('Group B conversion rate is: {:.2f}'.format(total_purchaser/total_participants))
```

```
Group B conversion rate is: 0.28
```

Interesting, group **A** has higher conversion rates (31% vs 28% for group **B**)- This means that group A users were more likely to make their first purchase and convert from user to a **customer**.

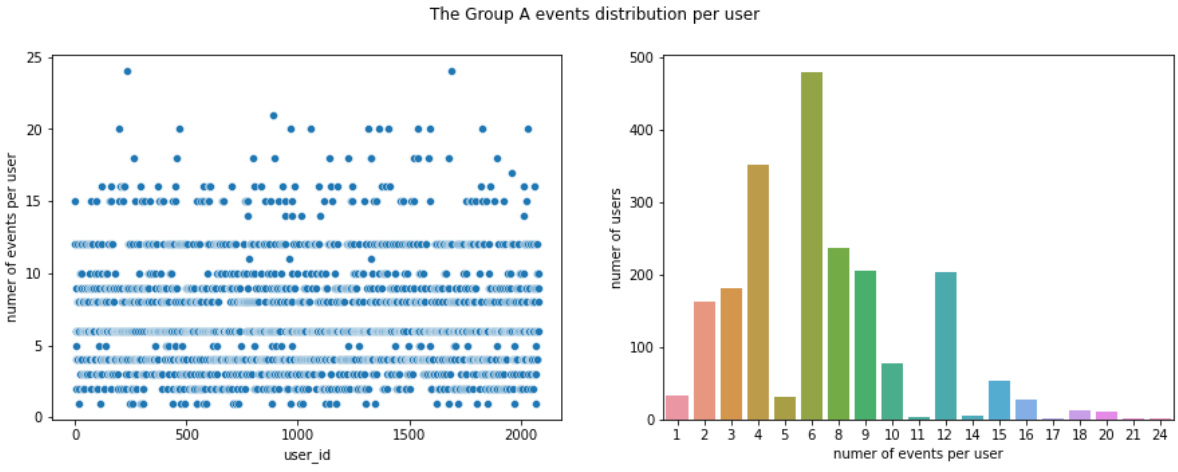
2.3 Data Distribution

Lets take a look at events distribution per user:

In [32]:

```
# grouping events dataframe by user_id, counting events to see how many events each user ha
events_per_user_a = rec_sys_a.groupby(['user_id'])['event_name'].count().reset_index()
display(events_per_user_a.describe())
x_values = pd.Series(range(0,len(events_per_user_a)))
test_a = events_per_user_a.groupby('event_name')['user_id'].count().reset_index()
test_a.columns = ['event_number', 'user_id']

fig, axes = plt.subplots(1, 2, figsize=(15,5))
sns.scatterplot(x = x_values , y = events_per_user_a['event_name'], ax=axes[0])
sns.barplot(data = test_a, y = 'user_id', x = 'event_number', ax=axes[1])

axes[0].set_xlabel('user_id')
axes[1].set_xlabel('numer of events per user')

axes[0].set_ylabel('numer of events per user')
axes[1].set_ylabel('numer of users')

fig.suptitle('The Group A events distribution per user')
plt.show()
```

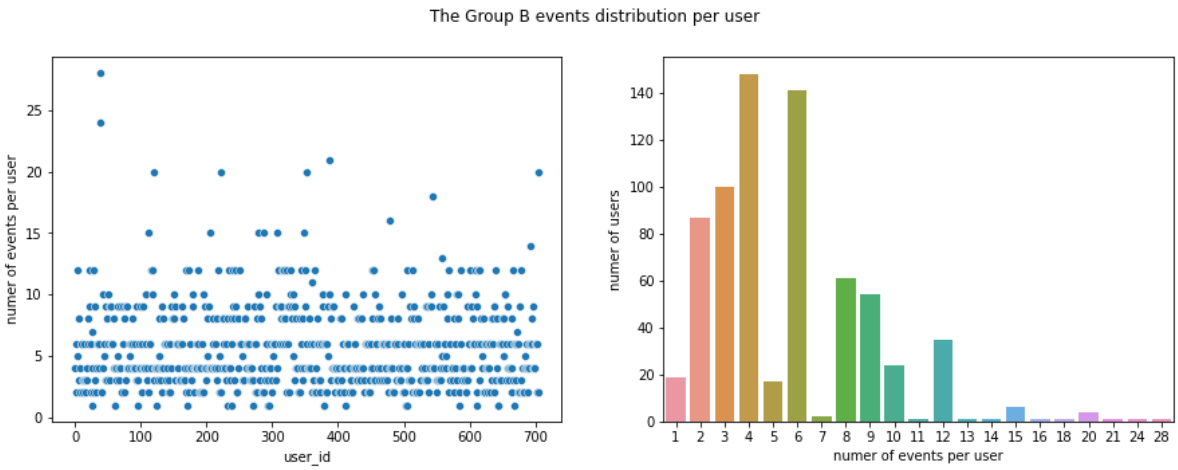
	event_name
count	2082.000000
mean	6.827570
std	3.737359
min	1.000000
25%	4.000000
50%	6.000000
75%	9.000000
max	24.000000



In [33]:

```
events_per_user_b = rec_sys_b.groupby(['user_id'])['event_name'].count().reset_index()
display(events_per_user_b.describe())
x_values = pd.Series(range(0,len(events_per_user_b)))
test_b = events_per_user_b.groupby('event_name')['user_id'].count().reset_index()
test_b.columns = ['event_number', 'user_id']

fig, axes = plt.subplots(1, 2, figsize=(15,5))
sns.scatterplot(x = x_values , y = events_per_user_b['event_name'], ax=axes[0])
sns.barplot(data = test_b, y = 'user_id', x = 'event_number', ax=axes[1])

axes[0].set_xlabel('user_id')
axes[1].set_xlabel('numer of events per user')

axes[0].set_ylabel('numer of events per user')
axes[1].set_ylabel('numer of users')

fig.suptitle('The Group B events distribution per user')
plt.show()
```

	event_name
count	706.000000
mean	5.635977
std	3.440156
min	1.000000
25%	3.000000
50%	4.000000
75%	8.000000
max	28.000000



The graphs looks somewhat similar, most users had less than 15 events in total for both groups.

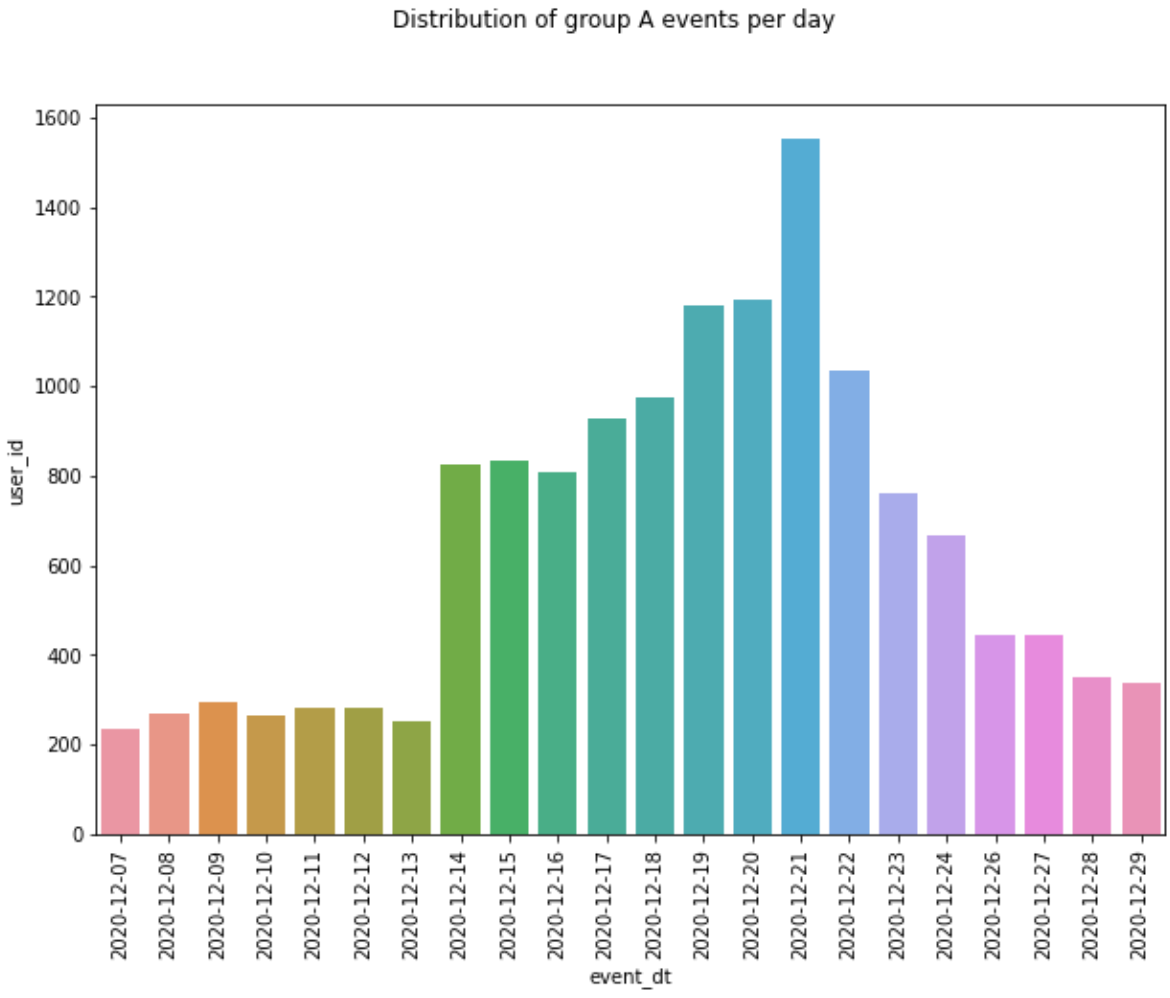
Looking at the stats above each graph, we can see that:

1. group A had a median of 6, with std as 3.7 then most of the values in group A is in range between 2 events to 10 events.
2. group B had a median of 4, with std as 3.4 then most the values in group B is in the range between 1 events to 8 events
3. For both groups, 75% of the data is between 1 and 9 events.

Now, lets have a look at the distribution of events per day, how many events did we have per day during the testing period - for both groups A, and B.

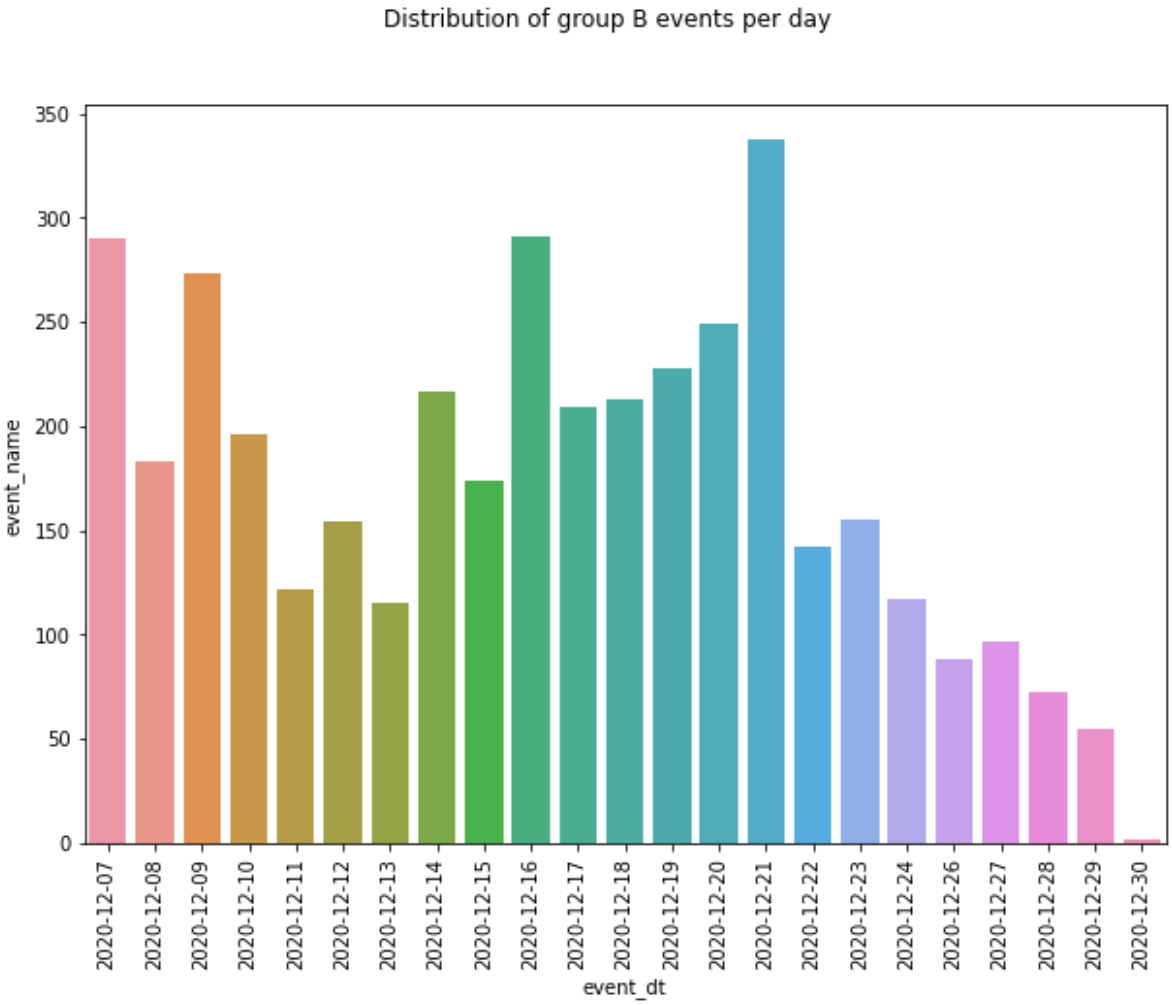
In [35]:

```
# here we group by event_dt (date) since we want results to be events per day
events_per_day_a = rec_sys_a.groupby(['event_dt'], as_index = False)['user_id'].count().res
x_dates = events_per_day_a['event_dt'].dt.strftime('%Y-%m-%d').sort_values().unique()

plt.figure(figsize = (10,7))
plt.xticks(rotation=90)
sns.barplot(data = events_per_day_a, x = 'event_dt', y = 'user_id').set_xticklabels(labels=
plt.suptitle('Distribution of group A events per day')
plt.show()
```



In [37]:

```
events_per_day_b = rec_sys_b.groupby(['event_dt'])['event_name'].count().reset_index()
x_dates = events_per_day_b['event_dt'].dt.strftime('%Y-%m-%d').sort_values().unique()

plt.figure(figsize = (10,7))
plt.xticks(rotation=90)
sns.barplot(data = events_per_day_b, x = 'event_dt', y = 'event_name').set_xticklabels(labe
plt.suptitle('Distribution of group B events per day')
plt.show()
```



The distribution of events per day for the groups is different, we can see that group A started low, then peaked and then at the end of the month it started going low again. For group **B**, we can see that started high and had a little bit of highs and lows until it peaked and then started to get lower and lower.

2.4 Conversion Funnel

Lets visualize conversion of each step, do we notice any differences? did group **A** score better here? Lets find out.

In [38]:

```
# grouping by event_name and counting user_ids to count events at each stages
final_df_14 = final_df[final_df['event_dt'] <= '2020-12-21']
events_funnel = final_df_14.groupby('event_name')['user_id'].count().reset_index().sort_val
events_funnel
```

Out[38]:

	event_name	user_id
0	login	6068
2	product_page	3764
1	product_cart	1839
3	purchase	1752

In [39]:

```
events_funnel['perc_ch'] = events_funnel['user_id'].pct_change()
events_funnel
```

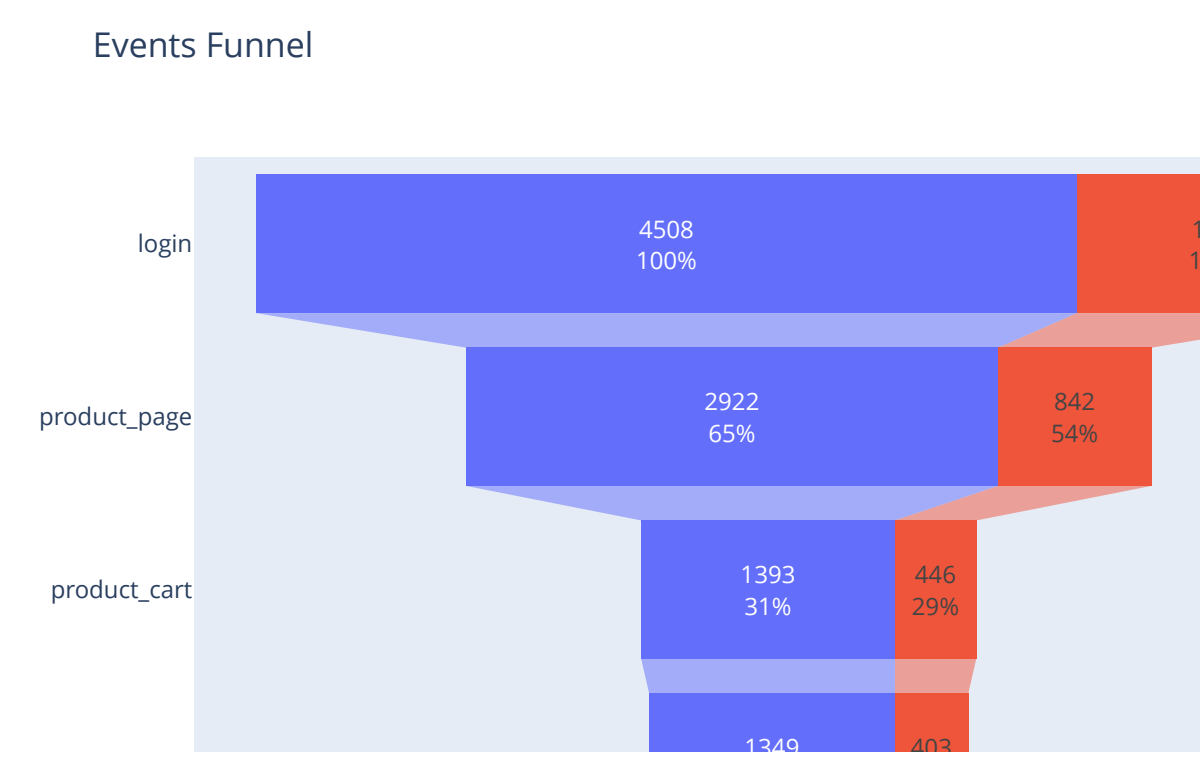
Out[39]:

	event_name	user_id	perc_ch
0	login	6068	NaN
2	product_page	3764	-0.379697
1	product_cart	1839	-0.511424
3	purchase	1752	-0.047308

We can see that the order of the funnel is as expected; Login > Product Page > Cart Page > Purchase. Lets have a look how each of the groups scored from each step to the next one:

In [40]:

```
# adding the funnels for each group into a list to display later
funnel_by_groups=[]
for i in final_df_14['group'].unique():
    group = final_df_14[final_df_14['group'] == i].groupby(['event_name'])['user_id'].count
    funnel_by_groups.append(group)

fig = go.Figure(layout_title_text = 'Events Funnel')

fig.add_trace(go.Funnel(
    name = 'A',
    y = funnel_by_groups[0]['event_name'],
    x = funnel_by_groups[0]['user_id'],
    textinfo = "value+percent initial"))

fig.add_trace(go.Funnel(
    name = 'B',
    orientation = "h",
    y = funnel_by_groups[1]['event_name'],
    x = funnel_by_groups[1]['user_id'],
    textinfo = "value+percent initial"))

fig.show()
```



Not very promising, group A score better in terms of event numbers in each stage during the first 14 days.

In [41]:

```
users_funnel = final_df.groupby('event_name')['user_id'].nunique().reset_index().sort_value
users_funnel = users_funnel.reindex([0, 2, 1, 3]) # manually editing to put product_cate ab
users_funnel
```

Out[41]:

	event_name	user_id
0	login	2787
2	product_page	1757
1	product_cart	826
3	purchase	850

In [42]:

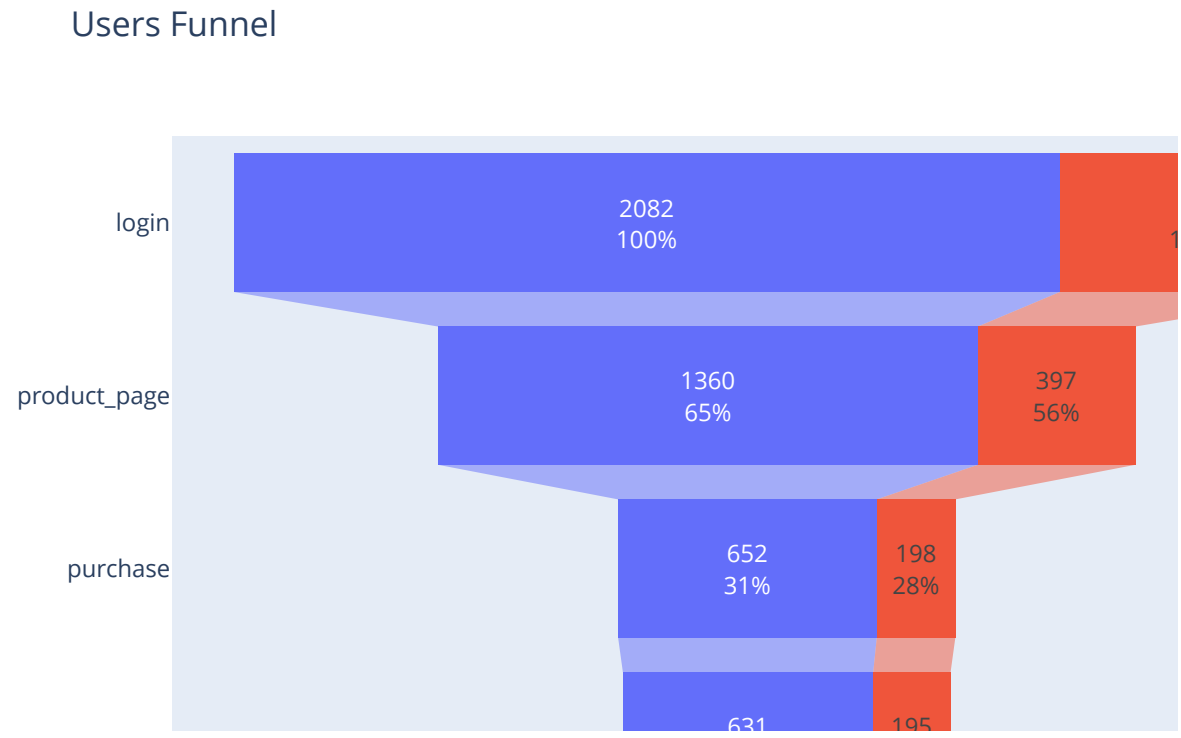
```
users_funnel['perc_ch'] = users_funnel['user_id'].pct_change()
users_funnel
```

Out[42]:

	event_name	user_id	perc_ch
0	login	2787	NaN
2	product_page	1757	-0.369573
1	product_cart	826	-0.529880
3	purchase	850	0.029056

In [43]:

```
funnel_by_groups=[]
for i in final_df['group'].unique():
    group = final_df[final_df['group'] == i].groupby(['event_name'])['user_id'].nunique().r
    funnel_by_groups.append(group)

fig = go.Figure(layout_title_text = 'Users Funnel')

fig.add_trace(go.Funnel(
    name = 'A',
    y = funnel_by_groups[0]['event_name'],
    x = funnel_by_groups[0]['user_id'],
    textinfo = "value+percent initial"))

fig.add_trace(go.Funnel(
    name = 'B',
    orientation = "h",
    y = funnel_by_groups[1]['event_name'],
    x = funnel_by_groups[1]['user_id'],
    textinfo = "value+percent initial"))

fig.show()
```



Same here, user conversion rates in each stage was lower in group B than in group A. This means that at group A we had higher percentages of users move to the next stage and finally make a purchase than users in group B.

So unlike the expected results of the technical descriptions of the test; we have a drop in the conversions to the second stage (product page views), third stage (product cart) and purchase stage (last).

Other than that:

In [46]:

```
region_grouped = final_df.groupby('region')['user_id'].nunique().reset_index()
display(region_grouped)

print('Total EU new users:', new_users[new_users['region'] == 'EU'].shape[0])
```

	region	user_id
0	APAC	45
1	CIS	30
2	EU	2594
3	N.America	119

```
Total EU new users: 43396
```

We can see that the total number of participants is 2788 (not 6000, even if we kept the 887 users that were in the intersection of the 2 tests). Besides, we noticed that 2594 participants are from EU while we had around 43k EU new users - making their participance percentage around 5.9%, where the technical description mentioned that it is going to be 15% - **The technical description was not met by our test results**.

3 Part 3: A/B Test

To check whether there's a statistical difference between groups A and B, lets perform a Z-test, checking whether there was difference at each stage:

In [48]:

```
groups_pivot = final_df.pivot_table(index='event_name', columns='group',values='user_id',ag
groups_pivot
```

Out[48]:

group	event_name	A	B
0	login	2082	705
1	product_cart	631	195
2	product_page	1360	397
3	purchase	652	198

- Null hypothesis(H0): Groups A and B difference is not statistically significant
- Alt. hypothesis(H1): Groups A and B difference is statistically significant

In [49]:

```
def check_hypothesis(data, pivot, group1, group2, event, alpha=0.05):
    success1 = pivot[pivot['event_name'] == event][group1].iloc[0]
    success2 = pivot[pivot['event_name'] == event][group2].iloc[0]

    trials1 = data[data['group'] == group1]['user_id'].nunique()
    trials2 = data[data['group'] == group2]['user_id'].nunique()

    # success proportion in the first group:
    p1 = success1/trials1

    # success proportion in the second group:
    p2 = success2/trials2

    # success proportion in the combined dataset:
    p_combined = (success1 + success2) / (trials1 + trials2)

    # the difference between the datasets' proportions
    difference = p1 - p2
    # calculating the statistic in standard deviations of the standard normal distribution
    z_value = difference / math.sqrt(p_combined * (1 - p_combined) * (1/trials1 + 1/trials2

    # setting up the standard normal distribution (mean 0, standard deviation 1)
    distr = st.norm(0, 1)
    # calculating the statistic in standard deviations of the standard normal distribution


    p_value = (1 - distr.cdf(abs(z_value))) * 2

    print('p-value: ', p_value)

    if (p_value < alpha):
        print("Rejecting the null hypothesis for event:", event + ",", "in groups", group1,
    else:
        print("Failed to reject the null hypothesis for event:", event + ',', "in groups",
```

In [50]:

```
for v in final_df['event_name'].unique():
    check_hypothesis(final_df, groups_pivot, 'A', 'B', v, alpha = 0.05)
```

```
p-value:  0.10281767567786759
Failed to reject the null hypothesis for event: purchase, in groups A and B
p-value:  0.1766337419130104
Failed to reject the null hypothesis for event: product_cart, in groups A an
d B
p-value:  1.5371909704686715e-05
Rejecting the null hypothesis for event: product_page, in groups A and B
p-value:  0.08587401754779211
Failed to reject the null hypothesis for event: login, in groups A and B
```

Our test failed to reject 3 out of 4 statistical tests, but since we performed 4 dependant statistical tests simultaneously, we should consider the risk of false-positives increaseing. Therefore, we'll use the Bonfferi Correction method to avoid this increasing risk of false-positives

Bonfferi Correction: alpha / number_of_tests -> 5%/4 -> 1.25%

In [51]:

```
for v in final_df['event_name'].unique():
    check_hypothesis(final_df, groups_pivot, 'A', 'B', v, alpha = 0.0125)
```

```
p-value:  0.10281767567786759
Failed to reject the null hypothesis for event: purchase, in groups A and B
p-value:  0.1766337419130104
Failed to reject the null hypothesis for event: product_cart, in groups A an
d B
p-value:  1.5371909704686715e-05
Rejecting the null hypothesis for event: product_page, in groups A and B
p-value:  0.08587401754779211
Failed to reject the null hypothesis for event: login, in groups A and B
```

Again, 3 out of 4 tests were failed to reject the null hypothesis - meaning that the difference is not statistically significant for most of the conversions - except for the product_page conversion, where the null hypothesis was rejected, but still according to our studies, group A actually scored better at this conversion, which worsen the situation for group B even more.

4 General Summary

At the first step PART-1:-PREPROCESSING, we found out that:

- We have 14 marketing events annually.
- 58703 new users signed-up between 7th of Dec till 21st of Dec
- Devices used for sign-up: PC, Android, I-phone, Mac
- Regions new users come from: EU, NA, APAC, CIS
- During the test period, we had almost 424,000 user events, among them around 60,000 purchases.
- 13638 participants participated in the test

At the second step PART-2:-EDA, we found out that:

- We have two marketing events within the test period - only one that interests us (EU region)
- Group A conversion rates are higher than group B's conversion rates
- Group A's funnel for events and users looked better than group B's for events and users - meaning, conversion from each stage to another (login to product page, ..., product_cart to purchase) were higher for group A.
- Almost none of the technical descriptions were met.

At the third step PART-3:-A\B Test, we found out that:

- There's no statistical difference between our groups, A and B - The test failed to reject the null hypothesis.

4.1 Recommendations

The data for the test was really messy, alot of disturbance due to having more than one test in the data and also having marketing events during the test period which makes predicting how good our conclusions are almost impossible. Therefore, I think we should re-do the test, in a more stable enviroment, with more precise and correct data that has lower levels of disturbance.

Reviewer's comment v.1

👍 It is good that you provided the conclusions for each stage of the project. Also, the general recommendations are correct - several mistakes were made during the test, which can affect the result. Also, we remember that the number of users in the test has decreased, and this may affect the power of the test (i.e. increase the number of errors of the second type). To calculate the power of the test, we can use online calculators like this: https://abtestguide.com/calc/ (https://abtestguide.com/calc/)

📝 However, please check the conclusions after working with the comments.

Student's comment v.1

Thanks for your insights, and may we meet again in the future!

Reviewer's comment v.2

👍 Thank you for your additional jobs and your feedback on my comments! It was a pleasure to work with you!