Hello Ameer!

My name is Alexey Kargin, and I will review your **Integrated Project 2** project. Nice to meet you! I have added all my comments on your new project to cells with different coloring. Please, do not move or delete them - it helps us move faster:

Reviewer's comment v.1

👍 I am using green color for my comment if everything is done successfully.

Reviewer's comment v.1

📝 I will use yellow color for my comment if I can give you a recommendation or think that something can be improved. These are optional recommendations, but it will be perfect if you work with them in this or future projects.

Reviewer's comment v.1

✍ I am using red color for my comment if the block requires some extra work or corrections. The project couldn't be accepted with red comments.

To make a dialog, let's use this style for your remarks:

Student's comment v.1

Just like this. To make a similar block - double click on the block, copy and edit it in places you are changing. Also, if you have any questions, you could highlight them using this block.

General feedback v.1

👏 Ameer, thank you for sending your project!

👍 You've made a lot of correct steps. The code has not made mistakes and looks professional. I see you demonstrated solid skills in data analysis. You are able to apply any tool you need: python coding, data processing, hypotheses testing, detailed markdown text, and visualization. However, a few things need to be corrected in your project. You will find them as red comments. I guess that it will be easy to fix.

Every issue with our code is a chance for us to learn something new.

Best regards,
Alexey

General feedback v.2

👏 Ameer, thank you for sending in the revised version of the project and feedback on my comments 👍

Your corrections look great, and I'm glad to say that your project has been accepted 🥕 Please check my additional comments about how we could modification the stat function. It was a pleasure to review your project. Keep up the good work, and good luck on the next projects!

Best regards,
Alexey

Project Description

General Analytics: I work at a startup that sells food products. We need to investigate user behavior for the company's app. We'll start by the sales funnel, see how many users end up purchasing, their joureny to the purchase stage and how many of them get stuck at previous stages.

A/A/B Analysis: Once we're done with the **general analytics** part, we'll continue to test whether changing the font for the entire app would lead to a change in the customer's behaviors. Based on the results, the management team will decide whether to do such step or not.

- We are using A/A/B analysis to be more confident in the results we get. We'll only be considering the results in case there's no significal difference between the two A groups.

Purpose of Project

- General Analytics, purpose:

1. Study users' behaviors
2. Find out how many users end up purchasing
3. Calculate the path of the users from the start till the purchase
4. Find out where do users usually get stuck on and quit.

- A/A/B Analysis:

1. Find out whether changing the font for the entire app have an effect on our customers/users.

Plan Of Work

- We'll start by loading all the data that we have and the libraries needed respectively.
- Then we'll have a look at our data, find potential mistakes such as; duplicates, missing values etc and fix them.
- We'll proceed and start studying the data, learning the parameters presented in the dataset to have a better understanding of what we're up against.
- We'll study the sales funnel, find out which path do users often take to reach the purchase page and where do they often get stuck at etc.
- Once we're done with the sales funnel and have determined our conclusions about the general analytics part and the sales funnel, we move on to A/A/B analysis.
- We'll study how changing the font for the entire app affect our users and based on the results we'll decide whether to change the font or not.
- Finally, list all of our conclusions!

1 Initilization

We'll start by loading needed libraries and the dataset

1.1 Loading datasets and libraries

Loading necessary libraries:

Reviewer's comment v.1

👍 Well done. The main information about the project has been provided. We could also add the description of the source data.

In [1]:

```
import pandas as pd
from IPython.display import display
import plotly.express as px
from plotly import graph_objects as go
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
import math
from scipy import stats as st
import numpy as np
warnings.filterwarnings("ignore")
```

Reviewer's comment v.1

👍 Well done! It's a good practice to make all imports in 1st cell (https://www.python.org/dev/peps/pep-0008/#imports).

Loading given datasets:

In [2]:

```
log_data = pd.read_csv('/datasets/logs_exp_us.csv', sep = '\t', engine = 'python')
display(log_data.sample(10))
display(log_data.info())
```

	EventName	DeviceIDHash	EventTimestamp	ExpId
7285	MainScreenAppear	9147387344835590723	1564641959	247
1917	MainScreenAppear	1289632273471556632	1564601884	246
231439	OffersScreenAppear	7726349659490295219	1565187996	246
187324	OffersScreenAppear	689942998779770777	1565083143	248
231922	MainScreenAppear	4711030239663802604	1565188587	246
161381	PaymentScreenSuccessful	197027893265565660	1565014661	246
56093	MainScreenAppear	907516373885804773	1564748439	246
59172	MainScreenAppear	284324717439504162	1564753626	246
174785	CartScreenAppear	2305766456715991733	1565034810	248
7583	OffersScreenAppear	7702139951469979	1564642669	247

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244126 entries, 0 to 244125
Data columns (total 4 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   EventName       244126 non-null  object
 1   DeviceIDHash    244126 non-null  int64
 2   EventTimestamp  244126 non-null  int64
 3   ExpId           244126 non-null  int64
dtypes: int64(3), object(1)
memory usage: 7.5+ MB

None
```

Reviewer's comment v.1

👍 Well done! The data have been loaded and observed. Good.

We can see that the column names are not very ideal, we should probably make some changes to them. Other than that, make some type converting like converting dates from strings to datetimes for instance.

1.2 Fixing minor issues

Fixing column names:

In [3]:

```
# for clearer column names we'll use the following names:
log_data.columns = ['event_name', 'device_id', 'date_time', 'group']
```

Reviewer's comment v.1

👍 Sometimes, it is better to use the `.rename()` method to replace the column names. It allows us to avoid some problems if the data are changed in real projects. Please see additional information (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.rename.html).

Converting dates into datetime:

In [4]:

```
# converting out timestamp to a datetime
log_data['date_time'] = pd.to_datetime(log_data['date_time'], unit='s')
log_data.head()
```

Out[4]:

	event_name	device_id	date_time	group
0	MainScreenAppear	4575588528974610257	2019-07-25 04:43:36	246
1	MainScreenAppear	7416695313311560658	2019-07-25 11:11:42	246
2	PaymentScreenSuccessful	3518123091307005509	2019-07-25 11:28:47	248
3	CartScreenAppear	3518123091307005509	2019-07-25 11:28:47	248
4	PaymentScreenSuccessful	6217807653094995999	2019-07-25 11:48:42	248

Reviewer's comment v.1

👍 Correct!

Changing the group IDs 246,247 and 248 into group names A1, A2 and B respectively for easier and clearler use later on:

In [5]:

```
# changing the group ids to group names
log_data['group'] = log_data['group'].replace({246:'A1', 247:'A2', 248:'B'})
log_data.info()
log_data.head(5)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244126 entries, 0 to 244125
Data columns (total 4 columns):
 #   Column      Non-Null Count   Dtype         
---  ------      --------------   -----         
 0   event_name  244126 non-null  object        
 1   device_id   244126 non-null  int64         
 2   date_time   244126 non-null  datetime64[ns]
 3   group       244126 non-null  object        
dtypes: datetime64[ns](1), int64(1), object(2)
memory usage: 7.5+ MB
```

Out[5]:

	event_name	device_id	date_time	group
0	MainScreenAppear	4575588528974610257	2019-07-25 04:43:36	A1
1	MainScreenAppear	7416695313311560658	2019-07-25 11:11:42	A1
2	PaymentScreenSuccessful	3518123091307005509	2019-07-25 11:28:47	B
3	CartScreenAppear	3518123091307005509	2019-07-25 11:28:47	B
4	PaymentScreenSuccessful	6217807653094995999	2019-07-25 11:48:42	B

Reviewer's comment v.1

👍 Well done! It is a good idea to rename the test groups according to their purpose.

We're done with minor fixes, its time to tackle the bigger problems like duplicates or missing values.

2 Preparing the data for analysis

This is the preprocessing step, we'll check our data for any problematic values that may disturb or distort our results and make sure to have a clean dataframe when we're done with this step so we can moved on to studying it.

2.1 Looking for missing values

In [6]:

```
# count nulls
log_data.isna().sum()
```

Out[6]:

```
event_name    0
device_id     0
date_time     0
group         0
dtype: int64
```

Short, no missing values. Great, lets move on!

Reviewer's comment v.1

👍 Good news!

2.2 Looking for duplicates

In [7]:

```
# count duplicated rows
log_data.duplicated().sum()
```

Out[7]:

```
413
```

Duplicated rows represent events that happened at the same date and at the same second/time in the same device. Rows like this must be a mistake, so lets remove them:

In [8]:

```
# removing all duplicated rows
log_data.drop_duplicates(inplace=True)
log_data.reset_index(inplace=True, drop=True)
log_data.duplicated().sum()
```

Out[8]:

```
0
```

Reviewer's comment v.1

👍 Good job here! In this case, we do not know the nature of these duplicates because the data does not have an event identification number. And we do not know if these were several events or a single one. But you are right here - we could drop the duplicates with this case. Moreover, it is only `413` events from `244126`.

Also, it is better to show not only number of duplicate values but their share:
`log_data.duplicated().agg({'mean','sum'})`

We have no more problematic valuesin our dataframe. How else can we improve it? Well, it is a good practice to add a date column to the dataframe when we have a datetime column. This allows us to group by date later on (since it wouldn't work on datetimes as every datetime is different from the other).

In [9]:

```
# adding a date column to log_data
log_data['date'] = log_data['date_time'].dt.date
log_data.head()
```

Out[9]:

	event_name	device_id	date_time	group	date
0	MainScreenAppear	4575588528974610257	2019-07-25 04:43:36	A1	2019-07-25
1	MainScreenAppear	7416695313311560658	2019-07-25 11:11:42	A1	2019-07-25
2	PaymentScreenSuccessful	3518123091307005509	2019-07-25 11:28:47	B	2019-07-25
3	CartScreenAppear	3518123091307005509	2019-07-25 11:28:47	B	2019-07-25
4	PaymentScreenSuccessful	6217807653094995999	2019-07-25 11:48:42	B	2019-07-25

Reviewer's comment v.1

👍 Well done!

Finally, lets check that no users belong to more than one group. Users that belong to more than one group are problematic users that will only distort the results thus we should remove them.

In [10]:

```
# dividing the logs into 3 groups
a1_group = log_data.query('group == "A1"')
a2_group = log_data.query('group == "A2"')
b_group = log_data.query('group == "B"')

# merging each 2 groups with 'inner' to get any shared rows
a1_a2_group = pd.merge(a1_group, a2_group, how = 'inner', on = ['device_id'])
a1_a2_b_group = pd.merge(a1_a2_group, b_group, how = 'inner', on = ['device_id'])

# checking if the final df has any rows
a1_a2_b_group.shape[0]
```

Out[10]:

```
0
```

Reviewer's comment v.1

👍 Correct! This checking is very important in terms of the A/B test.

0! We're ready to start studying our data!

Reviewer's comment v.1

⭐ Perfect job at this stage! The data have been prepared for the following investigation.

3 Studying the data

Lets get familiar with the data that we have, starting with the events:

3.1 How many events are in the logs

In [11]:

```
print("types of events:" ,log_data['event_name'].nunique())
print("number of total events:" ,log_data.shape[0]) # counting how many events we have in t
```

```
types of events: 5
number of total events: 243713
```

Now that we know how many types of events we have and how much total events there is, lets have a similar look at our user data:

3.2 How many users are in the logs

In [12]:

```
print("number of unique devices:" ,log_data['device_id'].nunique())
```

```
number of unique devices: 7551
```

Reviewer's comment v.1

👍 Well done!

We know how much events we have in total, we know how much users we have in total. What can we do with this information?

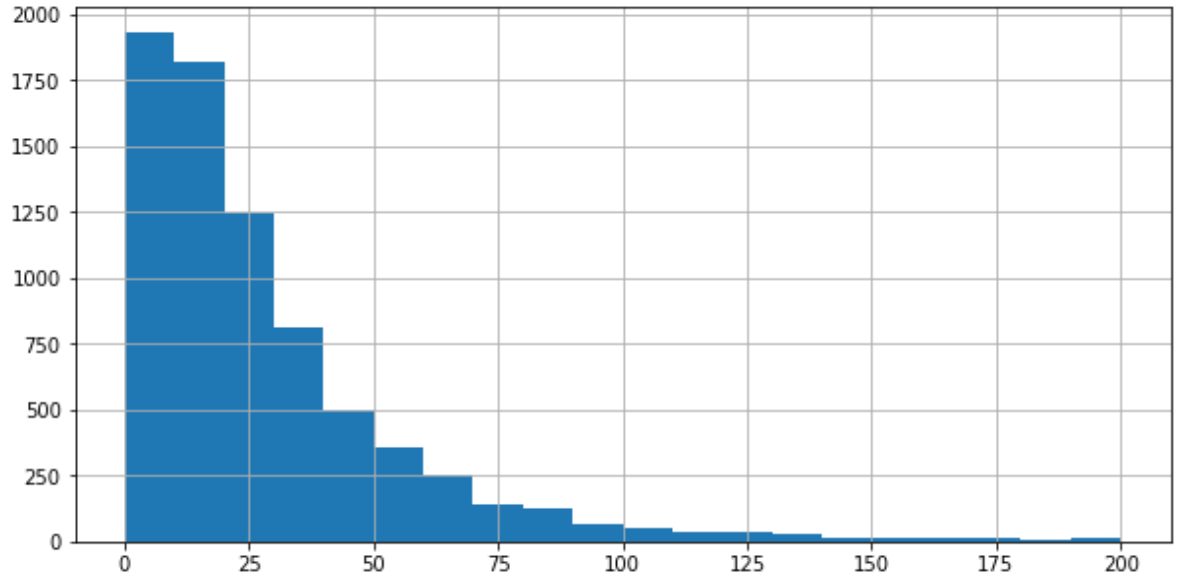
3.3 Average number events per user

In [13]:

```
events_per_user = log_data.groupby('device_id')['event_name'].count()
events_per_user.hist(bins = 20, range=(0,200), figsize=(10,5))
```

Out[13]:

```
<AxesSubplot:>
```



In [14]:

```
print("Median number of events per user: {:.2f}".format(events_per_user.median()))
```

```
Median number of events per user: 20.00
```

Our data, presented in the graph is skewed to the right. Therefore we used the median method (instead of the average AKA mean) since it will be a more suitable option in this case.

Reviewer's comment v.1

📝 Correct. However, please note that it is better to study this data in detail in this case. We could calculate the number of events per user and plot a histogram to check the data type distribution. And based on the result, we could estimate what central tendency value characterizes the sample average: mean or median.

Student's comment v.1

Yep

CORRECTED

Reviewer's comment v.2

👍 Thank you for your additional job here! The results are correct - probably it is more informative to use the median value in this case. However, please do not forget about plot's title and axis labels.

3.4 What period of time does the data cover

In [15]:

```
# printing the minimum and maximum date to see what does the data cover
print("min date:" ,log_data['date_time'].min())
print("max date:" ,log_data['date_time'].max())
print("duration:" ,(log_data['date_time'].max() - log_data['date_time'].min()))
```

```
min date: 2019-07-25 04:43:36
max date: 2019-08-07 21:15:17
duration: 13 days 16:31:41
```

That's interesting. 13 days, 16 hours is almost 2 weeks! Do we have complete data for the whole 2 weeks?

In [16]:

```
# checking data completeness for the whole week
events_date = log_data.groupby('date', as_index = False).agg({'device_id': ['count']})
events_date.columns = ['date', 'number_of_events']
display(events_date)
fig = px.bar(events_date, x= 'date', y='number_of_events', title = 'number of events per da
fig.show()
```

	date	number_of_events
0	2019-07-25	9
1	2019-07-26	31
2	2019-07-27	55
3	2019-07-28	105
4	2019-07-29	184
5	2019-07-30	412
6	2019-07-31	2030
7	2019-08-01	36141
8	2019-08-02	35554
9	2019-08-03	33282
10	2019-08-04	32968
11	2019-08-05	36058
12	2019-08-06	35788
13	2019-08-07	31096

Reviewer's comment v.1

👍 Nice informative plot!

It seems like the data begins to be complete on the 1st of August. That's the start of week 2. This means that we practically have complete data for one week and not for the whole period, therefore we'll be excluding the period where data wasn't complete.

Lets exclude any data before that date (1st of August):

In [17]:

```
# filtering the data to only have dates higher than 31st of July
flog_data = log_data[log_data['date'] > pd.to_datetime('2019-7-31')] # f for filtered

# checking for min and max again for validation
print(flog_data['date'].min())
print(flog_data['date'].max())
```

```
2019-08-01
2019-08-07
```

We're left with the dates from 1 to 7 of August, 2019, one week exactly. But does the removal of the 7 other days have a big impact on the dataframe?

Reviewer's comment v.1

👍 Correct!

3.5 Lost events and users from the exclusion

To calculate how much events and users were lost, we are going to create a new filtered dataframe that includes only data from the excluded data. This means there's no intersection between the dataframes.

Lost events is simply the number of rows deleted. While lost users isn't necessairaly this. Lost users are users that were lost forever, meaning; users that appeared in the dataframe from data before 31st of August but didn't appear in the other dataframe.

Lets see how much of these we have:

In [18]:

```
# filtering our core data to only include dates equal/lower to 31st of July
flog_data_2 = log_data[log_data['date'] <= pd.to_datetime('2019-7-31')] # f for filtered
```

In [19]:

```
# lost events
flog_data_2.shape[0]
```

Out[19]:

```
2826
```

In [20]:

```
# lost users
f2_users = flog_data_2['device_id'].unique() # lost users
f1_users = flog_data['device_id'].unique() # remained users
f2_events = flog_data_2['device_id'].count() # lost events
f1_events = flog_data['device_id'].count() # remained events

shared_users = [value for value in f2_users if value in f1_users] # users that are both in

print("Total unique users before the 31st of July:", len(f2_users))
print("Shared unique users between users before 31st of July, 2019 and users after 31st of
print('Total data lost:', f2_events)
print('Total data remained:', f1_events)
print('Percentage of excluded data: {:.2%}'.format(f2_events/(f2_events+f1_events)))
print('Percentage of excluded users: {:.2%}'.format((len(f2_users) - len(shared_users))/(le
print('Users lost: {}'.format(len(f2_users) - len(shared_users)))
```

```
Total unique users before the 31st of July: 1451
Shared unique users between users before 31st of July, 2019 and users after
31st of July, 2019 is 1434
Total data lost: 2826
Total data remained: 240887
Percentage of excluded data: 1.16%
Percentage of excluded users: 0.19%
Users lost: 17
```

Lost users are users that were lost forever, meaning that they exist in the data before the 31st of July but don't exist in the data after the 31st of July. We have 17 such users (Since we had 1451 total unique users in the second filtered dataframe 'flog_data_2' (before 31-7-2019) and 1434 unique users in the shared_users dataframe, meaning 1434 users moved to the first filtered dataframe 'flog_data' (after 31-7-2019) and 17 didn't make it)

Finally, lets make sure that we still have users from all the groups after we have removed the unnecessary rows.

Reviewer's comment v.1

📝 Overall, you have provided good job at this stage, however, it is also better to calculate the share of excluded data and users here.

Student's comment v.1

Looks better and more informative now!

CORRECTED

Reviewer's comment v.2

👍 Thank you for considering this point!

3.6 Making sure we have users from all groups.

In [21]:

```
# checking the value counts for each group
print(flog_data[flog_data['group'] == 'A1']['device_id'].nunique())
print(flog_data[flog_data['group'] == 'A2']['device_id'].nunique())
print(flog_data[flog_data['group'] == 'B']['device_id'].nunique())
```

```
2484
2513
2537
```

We have pretty close numbers of users in each group, this is a good indicator. We're good and ready to start sales funnel and the general analytics part

Lets move on to sales funnel studying.

Reviewer's comment v.1

✍ Please note that here we have calculated the number of events not users. Please check and fix that. To calculate the number of users, it is better to use the `nunique()` method.

Student's comment v.1

You're totally right! Thank you for your advice!

CORRECTED

Reviewer's comment v.2

👍 Good! Thank you for correction here!

4 Studying the event funnel

Lets start funnel studying with calculating the frequencies of each of the events. Which event occurred more often? and which occurred the least.

How does this help us?

- It helps us build the funnel, usually events with the most occurrencies are the main and very first events to happen, making the top of the funnel and so on.

4.1 Events frequency

In [22]:

```
# counting occurrences of each event by grouping by event_name
grouped_events = flog_data.groupby('event_name')['device_id'].count().reset_index()
grouped_events.columns = ['event_name', 'occurrences']
grouped_events.sort_values('occurrences', ascending = False, inplace=True)
grouped_events
```

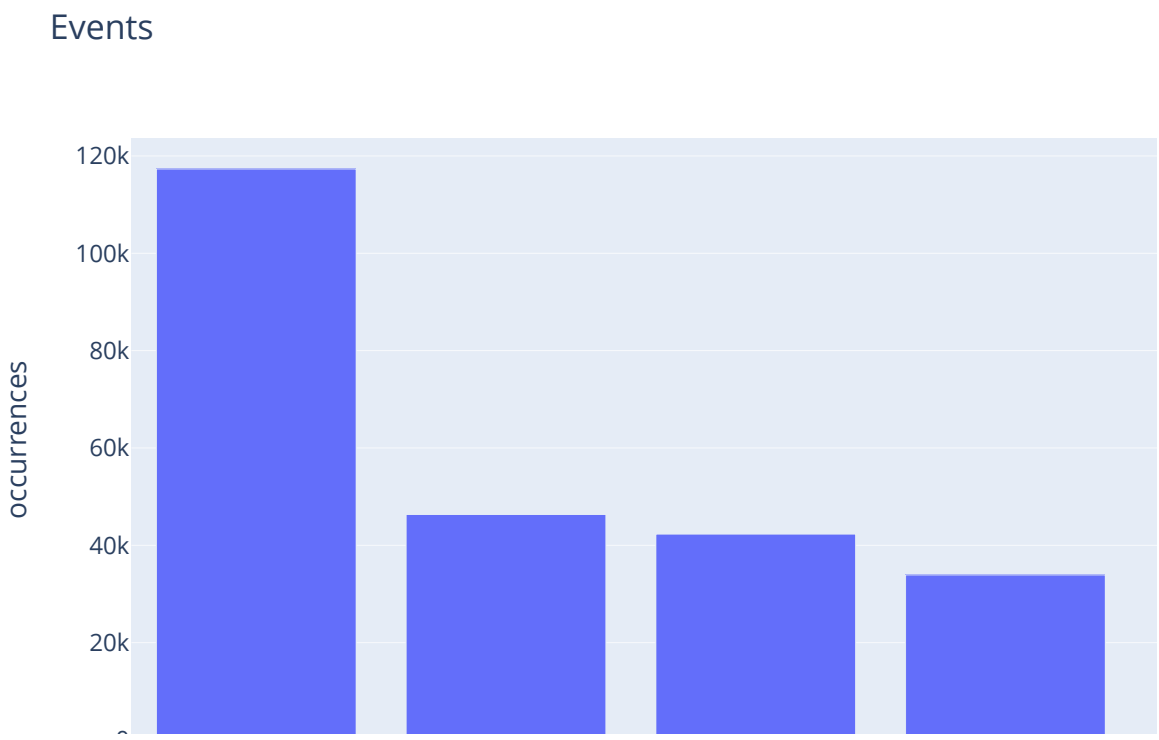
Out[22]:

	event_name	occurrences
1	MainScreenAppear	117328
2	OffersScreenAppear	46333
0	CartScreenAppear	42303
3	PaymentScreenSuccessful	33918
4	Tutorial	1005

We can see a funnel forming, with MainScreenAppear at the top, OffersScreenAppear next to it and PaymentScreenSuccessful at the bottom (or actually Tutorial, depending on the path). In the mean while lets have a look at a graph describing this funnel:

In [23]:

```
fig = px.bar(grouped_events, x= 'event_name', y='occurrences', title = 'Events')
fig.show()
```



Looks like a funnel to me :D

4.2 Unique users per event

We've found which event was performed the most often and which came next and which came last place. Lets calculate how much distinct users performed these events to confirm our funnel.

In [24]:

```
# calculating unique users for each event
unique_grouped_events = flog_data.groupby('event_name')['device_id'].nunique().reset_index(
unique_grouped_events.columns = ['event_name', 'unique_users']
unique_grouped_events.sort_values('unique_users', ascending = False, inplace=True)
unique_grouped_events.reset_index(drop=True)
```

Out[24]:

	event_name	unique_users
0	MainScreenAppear	7419
1	OffersScreenAppear	4593
2	CartScreenAppear	3734
3	PaymentScreenSuccessful	3539
4	Tutorial	840

Same order was the previous chart!

In [25]:

```
# calculating the proportion of users who performed an action at least once
unique_grouped_events['proportion'] = unique_grouped_events['unique_users']/flog_data['devi
unique_grouped_events = unique_grouped_events.reset_index(drop=True)
unique_grouped_events
```

Out[25]:

	event_name	unique_users	proportion
0	MainScreenAppear	7419	0.984736
1	OffersScreenAppear	4593	0.609636
2	CartScreenAppear	3734	0.495620
3	PaymentScreenSuccessful	3539	0.469737
4	Tutorial	840	0.111495

What do we learn from this? Most amount of users lost is between MainScreenAppear and OffersScreenAppear (around 38%), what could've gone wrong in-between? Our management team will have to look onto that!

In [26]:

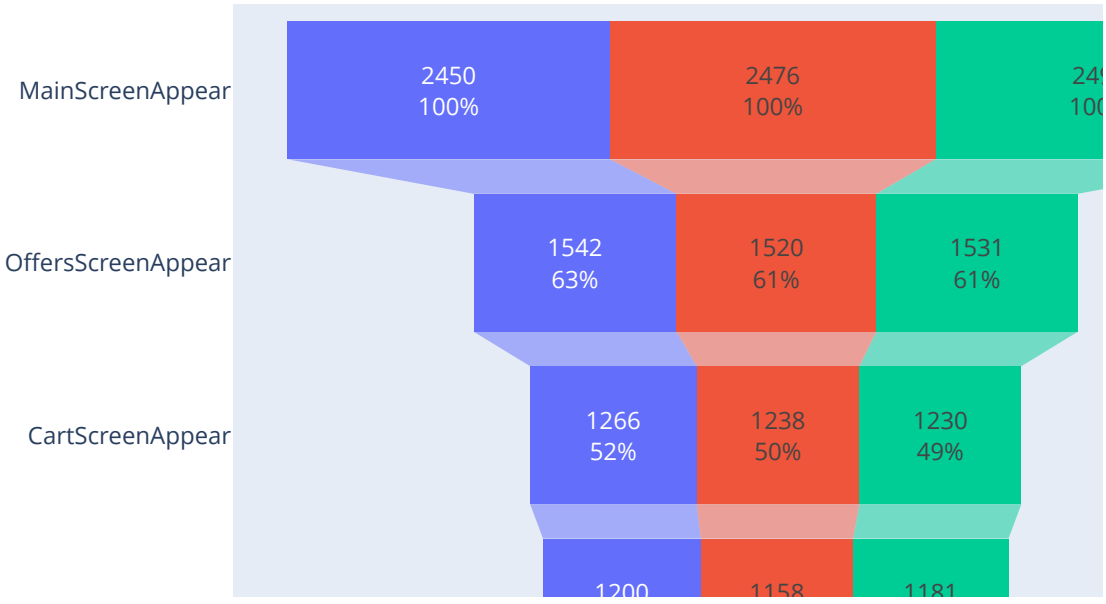
```
# below is a little funnel describing our data
funnel_shift = flog_data.groupby(['event_name'])['device_id'].nunique().sort_values(ascendi
funnel_shift['perc_ch'] = funnel_shift['device_id'].pct_change()

funnel_by_groups=[] # a list to store our 3 groups in
for i in flog_data.group.unique():
    # grouping each group by group and event name
    group = flog_data[flog_data.group==i].groupby(['event_name','group'])['device_id'].nuni
    group = group[group['event_name'] != 'Tutorial']
    funnel_by_groups.append(group)

funnel_by_groups
```

Out[26]:

```
[               event_name group  device_id
 1         MainScreenAppear    A1       2450
 2       OffersScreenAppear    A1       1542
 0         CartScreenAppear    A1       1266
 3  PaymentScreenSuccessful    A1       1200,
               event_name group  device_id
 1         MainScreenAppear    A2       2476
 2       OffersScreenAppear    A2       1520
 0         CartScreenAppear    A2       1238
 3  PaymentScreenSuccessful    A2       1158,
               event_name group  device_id
 1         MainScreenAppear     B       2493
 2       OffersScreenAppear     B       1531
 0         CartScreenAppear     B       1230
 3  PaymentScreenSuccessful     B       1181]
```

In [27]:

```
#fig = go.Figure(go.Funnel(funnel_by_groups,x='device_id',y='event_name',color='group', tit

fig = go.Figure()

fig.add_trace(go.Funnel(
    name = 'A1',
    y = funnel_by_groups[0]['event_name'],
    x = funnel_by_groups[0]['device_id'],
    textinfo = "value+percent initial"))

fig.add_trace(go.Funnel(
    name = 'A2',
    orientation = "h",
    y = funnel_by_groups[1]['event_name'],
    x = funnel_by_groups[1]['device_id'],
    textposition = "inside",
    textinfo = "value+percent initial"))

fig.add_trace(go.Funnel(
    name = 'B',
    orientation = "h",
    y = funnel_by_groups[2]['event_name'],
    x = funnel_by_groups[2]['device_id'],
    textposition = "inside",
    textinfo = "value+percent initial"))

fig.show()
```



4.3 The sequence of events

Obviously, the main sequence is MainScreen > OffersScreen > CartScreen > PaymentScreen because this is the logical sequence. It doesn't make sense to go to the PaymentScreen without going to the CartScreen first or go to the CartScreen without going to the Main/OffersScreen. By looking at the graph above, the results confirm the sequence we suggested because MainScreenAppear has most unique users, then comes OffersScreenAppear then CartScreenAppear and finally PaymentScreenSuccessful.

*This still doesn't mean that there was a single sequence, because appearently there's another screen that appeared to the users and it is the **Tutorial** screen and this implies that there was another sequence because **Tutorial** wasn't included in the main sequence. An alternative second sequence could be the same as the main sequence but with **Tutorial** in-between MainSreen and OffersScreen or OffersScreen and CartScreen for instance as an optional action.*

*Is that all the sequences there was? Not quite. Looking at the results of the proportions, we can see that 98% of the users performed the MainScreenAppear action, meaning 2% didn't. The two sequences above both included the MainScreenAppear action which leads us to think that there was a third sequence that didn't include the event **MainScreenAppear** at all and it probably started with **OffersScreenAppear** instead (could be due to users clicking an ad banner of one of the offers which leads directly to the offer screen?)*

Reviewer's comment v.1

👍 Ameer, at this stage you have provided good jobs - the shares of events and users et each stage have been calculated correctly, and you have described the funnel in detail. Also, you have used correct plot type to discuss the funnel for each test groups.

Reviewer's comment v.1

✍ However, I would suggest and additional jobs here. Let's study the funnel again - you have correctly suggested the Tutorial wasn't included in the main sequence. It means that this event is an optional one and a lot of users miss it. In this case it is better to omit this event from the product funnel because at the plot above the Tutorial event seems as main goal of users. However, this is not the case, since the main goal is the purchase. In this case, it is better to plot the funnel using events from the main sequence. Please check and fix that.

Also, please add the title for the funnel plot.

Student's comment v.1

Obviously! I removed the tutorial and the funnel looks better already! I also added percentages for more information.

CORRECTED

Reviewer's comment v.2

👍 Good job here! Thank you! Yep, additional information as conversion from the initial stage will be useful for us.

4.4 User ratios for every step

Lets calculate how much users of each step proceeded to make the next step. Assuming that the main sequence is as suggested above then the results would look like this:

In [28]:

```
proportions = unique_grouped_events['proportion'] # the porportion we calculated earlier
percents = [] # we'll store the ratios here
for i in range(1,len(proportions)-1): # excluding 'tutorial' step
    percents.append(proportions[i]/proportions[i-1])

percents.insert(0,1) # inserting 1 at 0 because 0 is the MainScreenAppear and it didn't hav
percents.append(0) # adding 0 at the end for tutorial
# I did it that way because I don't think tutorial came after Payment so we shouldnt compar

percents = pd.Series(percents, name='funnel_churn_rates')

unique_grouped_events['funnel_churn_rates'] = percents
unique_grouped_events
```

Out[28]:

	event_name	unique_users	proportion	funnel_churn_rates
0	MainScreenAppear	7419	0.984736	1.000000
1	OffersScreenAppear	4593	0.609636	0.619086
2	CartScreenAppear	3734	0.495620	0.812976
3	PaymentScreenSuccessful	3539	0.469737	0.947777
4	Tutorial	840	0.111495	0.000000

Obviously, as we saw earlier. Only 62% for users that performed the first event, proceeded to the second event. While the other events had pretty high ratios.

4.5 At what stage do you lose the most users?

judging by the results we got on the previous question, the transition between MainScreenAppear and OffersScreenAppear had the lowest churn rate (61.9%) meaning 38% of users didn't make it to the OffersScreenAppear. This means that most users lost was at this step, probably due to users not being interested in any of the offers offered to them? or there could be another reason, we should inform our management team!

Reviewer's comment v.1

👍 Correct! Also, here - we could omit the Tutorial events.

4.6 What share of users make the entire journey from their first event to payment?

Again, assuming that the main sequence is as suggested above then the result would look like this:

In [29]:

```
main_users = unique_grouped_events.iloc[(0,1)] # users that performed the MainScreenAppear
payment_users = unique_grouped_events.iloc[(3,1)] # users that performed the PaymentScreenA
main_payment_users = payment_users/main_users

print("The share of users that made the entire journey is: {:.2%}".format(main_payment_user
```

```
The share of users that made the entire journey is: 47.70%
```

47.7% of the users made the whole journey from the Main screen to the payment screen, this is impressive!

A good addition to this result, would be to add the amount of time it took these customers to do this journey:

Reviewer's comment v.1

👍 Well done!

In [30]:

```
pay_time = pd.pivot_table(flog_data[flog_data['event_name'] != 'Tutorial'], index = ['devic
pay_time = pay_time.reset_index()
pay_time
```

Out[30]:

event_name	device_id	CartScreenAppear	MainScreenAppear	OffersScreenAppear	I
0	6888746892508752	NaT	2019-08-06 14:06:34	NaT	
1	6909561520679493	2019-08-06 18:52:58	2019-08-06 18:52:54	2019-08-06 18:53:04	
2	6922444491712477	2019-08-04 14:19:40	2019-08-04 14:19:33	2019-08-04 14:19:46	
3	7435777799948366	NaT	2019-08-05 08:06:34	NaT	
4	7702139951469979	2019-08-02 14:28:45	2019-08-01 04:29:54	2019-08-01 04:29:56	
...	...	...	...	...	
7525	9217594193087726423	2019-08-02 09:00:58	NaT	2019-08-02 09:01:08	
7526	9219463515465815368	2019-08-06 16:49:40	2019-08-05 05:26:26	2019-08-06 16:48:57	
7527	9220879493065341500	2019-08-02 17:59:16	2019-08-02 17:58:48	2019-08-02 17:59:00	
7528	9221926045299980007	NaT	2019-08-01 17:30:27	NaT	
7529	9222603179720523844	NaT	2019-08-01 06:52:13	NaT	

7530 rows × 5 columns

In [31]:

```
pay_time['time_difference'] = pay_time['PaymentScreenSuccessful'] - pay_time['MainScreenApp
pay_time.describe()
```

Out[31]:

event_name	device_id	time_difference
count	7.530000e+03	3441
mean	4.675790e+18	0 days 12:30:56.080209241
std	2.655474e+18	1 days 06:48:59.845620951
min	6.888747e+15	-7 days +10:05:52
25%	2.394672e+18	0 days 00:00:07
50%	4.687179e+18	0 days 00:03:27
75%	7.006700e+18	0 days 15:13:54
max	9.222603e+18	6 days 13:24:44

Reviewer's comment v.1

👍 Perfect additional studies here! The results are correct!

Looking at the results, the average time difference between a user being in the main screen and making a purchase was 12:30 hours. The median stood 3:27 minutes though (meaning the average was skewed towards up due to outliers like our maximum difference of 6 days!)

4.7 Summary

To sum everything up;

- We started by finding out which events occured more often and then we continued and calculated which users performed which events. These data helped us reach to a conclusion about the sequence of events.
- The sequence of events wasn't single. According to the results we receieved, we could conclude that there wasn't one sequence of events. But there was a main sequence, which was used the most often, and some secondary other sequences.
- Main sequence looks like this: MainScreen > OffersScreen > CartScreen > PaymentSuccessful with Tutorial somewhere in between. This sequence look very reasonable.
- Secondary sequence could be as follows: OffersScreen > CartScreen > PaymentScreen. How did we reach to such conclusion? Well, the MainScreenAppear had 98% performance ratio, meaning not all users saw this page. Which suggests that there was another sequence other than the main sequence that didn't include the MainScreenAppear.
- We also found out that the most users lost were between the steps MainScreenAppear and OffersScreenAppear, losing 38% of users at this step. Also, 47.7% of users make it from MainScreenAppear to PaymentSuccessful which is our conversion rate, and it is high!
- Finally, we calculated the average and median time difference between the event MainScreenAppear and PaymentSuccessful for customers. average stood at 12:30 hours and median at ~3 mins, suggesting that we have alot of outliers (or masssive outliers).

Reviewer's comment v.1

👍 The conclusions are correct! Good job here!

5 Studying the results of the experiment

Now that we're done with general analytics, lets proceed with the second part of the project. A/A/B analysis. At this part we will be checking whether changing the font of the entire app would have an effect on the customers. We have 2 control groups A1 and A2 and a test group B which had the font change. We'll only consider the results of this test if there was no statistical significance difference between the groups A1 and A2.

Lets start our research by calculating how much users do we have in each of the groups:

5.1 How many users are there in each group?

In [32]:

```
flog_data.groupby('group')['device_id'].nunique()
```

Out[32]:

```
group
A1    2484
A2    2513
B     2537
Name: device_id, dtype: int64
```

Reviewer's comment v.1

👍 Correct!

5.2 Statistical significance difference between control groups

In [33]:

```
events_pivot = flog_data.pivot_table(index='event_name', columns='group',values='device_id'
events_pivot.reset_index(inplace=True)
events_pivot
```

Out[33]:

group	event_name	A1	A2	B
0	CartScreenAppear	1266	1238	1230
1	MainScreenAppear	2450	2476	2493
2	OffersScreenAppear	1542	1520	1531
3	PaymentScreenSuccessful	1200	1158	1181
4	Tutorial	278	283	279

Now we have a dataframe that includes all the data needed for us to calculate the difference in signigicance level between our control groups.

We'll be using the function below to check our hypothesis by comparing the successes (amount of users that made it to a certain event) and trails (over-all users)

To test the conversion rate we'll have to compare between the users who reach the PaymentSuccessful stage between both groups and we'll be doing this test among other tests below.

Null Hypothesis (H0): The proportions of the two groups are equal

Alt. Hypothesis (H1): The proportions of the two groups are different.

Reviewer's comment v.1

📝 It is better to formulate the hypotheses differently: H0 - (what we will compare) the proportions of two groups are equal. And H1 - the proportions of two groups are not equal.

Student's comment v.1

THANKS!

```
CORRECTED
```

Reviewer's comment v.2

👍 Thank you for considering this point!

In [34]:

```
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]['device_id'].nunique()
    trials2 = data[data.group == group2]['device_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,"and groups", group1,group2)
    else:
        print("Failed to reject the null hypothesis for",event,"and groups", group1,group2
```

Reviewer's comment v.1

👍 The function is correct!

In [35]:

```
for v in flog_data['event_name'].unique():
    check_hypothesis(flog_data, events_pivot, 'A1', 'A2', v, alpha = 0.05)
```

```
p-value:  0.9376996189257114
Failed to reject the null hypothesis for Tutorial and groups A1 A2
p-value:  0.7570597232046099
Failed to reject the null hypothesis for MainScreenAppear and groups A1 A2
p-value:  0.2480954578522181
Failed to reject the null hypothesis for OffersScreenAppear and groups A1 A2
p-value:  0.22883372237997213
Failed to reject the null hypothesis for CartScreenAppear and groups A1 A2
p-value:  0.11456679313141849
Failed to reject the null hypothesis for PaymentScreenSuccessful and groups
A1 A2
```

The test failed to reject the null hypotheses for every step including conversion, this means there's no significant difference between our control groups and we can count on the results that we'll get with group B.

Reviewer's comment v.1

👍 Well done!

5.3 Statistical significance difference between control groups and test group

Similarily to what we did previously, we'll have to calculate the difference in statistical significance between the groups A1, A2 and B to check if we had different results for group B.

In [36]:

```
# just like we did before, but now we compare A1 and B
for v in flog_data['event_name'].unique():
    check_hypothesis(flog_data, events_pivot, 'A1', 'B', v, alpha = 0.05)
```

```
p-value:  0.8264294010087645
Failed to reject the null hypothesis for Tutorial and groups A1 B
p-value:  0.2949721933554552
Failed to reject the null hypothesis for MainScreenAppear and groups A1 B
p-value:  0.20836205402738917
Failed to reject the null hypothesis for OffersScreenAppear and groups A1 B
p-value:  0.07842923237520116
Failed to reject the null hypothesis for CartScreenAppear and groups A1 B
p-value:  0.2122553275697796
Failed to reject the null hypothesis for PaymentScreenSuccessful and groups
A1 B
```

In [37]:

```
for v in flog_data['event_name'].unique():
    check_hypothesis(flog_data, events_pivot, 'B', 'A2', v, alpha = 0.05)
```

```
p-value:  0.765323922474501
Failed to reject the null hypothesis for Tutorial and groups B A2
p-value:  0.4587053616621515
Failed to reject the null hypothesis for MainScreenAppear and groups B A2
p-value:  0.9197817830592261
Failed to reject the null hypothesis for OffersScreenAppear and groups B A2
p-value:  0.5786197879539783
Failed to reject the null hypothesis for CartScreenAppear and groups B A2
p-value:  0.7373415053803964
Failed to reject the null hypothesis for PaymentScreenSuccessful and groups
B A2
```

The test failed to reject the null hypothesis for both group A1 and B, and group A2 and B. This suggests that there's no difference in statistical significance between these 3 groups with a significance level of (alpha = 0.05) 5%. Judging by the results we got at the first step, it is very reasonable for the test to fail to reject the null hypothesis because there wasn't much difference between the conversion rate of groups A1, A2 and B, and it looked almost identical.

Lets confirm our data by also testing it on the combined data of A1 and A2 with B, logically all tests should fail:

In [38]:

```
# creating a new dataframe that only has A and B for the matter of comparing A1 and A2 comb
new_flog_data = flog_data.copy()
new_flog_data['group'] = new_flog_data['group'].replace({'A1':'A', 'A2': 'A'})
new_flog_data['group'].value_counts()
```

Out[38]:

```
A    156324
B     84563
Name: group, dtype: int64
```

In [39]:

```
new_events_pivot = new_flog_data.pivot_table(index='event_name', columns='group',values='de
new_events_pivot.reset_index(inplace=True)
new_events_pivot
```

Out[39]:

group	event_name	A	B
0	CartScreenAppear	2504	1230
1	MainScreenAppear	4926	2493
2	OffersScreenAppear	3062	1531
3	PaymentScreenSuccessful	2358	1181
4	Tutorial	561	279

In [40]:

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

```
p-value:  0.764862472531507
Failed to reject the null hypothesis for Tutorial and groups A B
p-value:  0.29424526837179577
Failed to reject the null hypothesis for MainScreenAppear and groups A B
p-value:  0.43425549655188256
Failed to reject the null hypothesis for OffersScreenAppear and groups A B
p-value:  0.18175875284404386
Failed to reject the null hypothesis for CartScreenAppear and groups A B
p-value:  0.6004294282308704
Failed to reject the null hypothesis for PaymentScreenSuccessful and groups
A B
```

As we expected, all tests failed to reject the null hypothesis and this confirms our conclusions.

Reviewer's comment v.1

👍 Well done! You have done the tests correctly! There is no difference between conversions with test and control groups.

- (Q) What significance level have you set to test the statistical hypotheses mentioned above? Calculate how many statistical hypothesis tests you carried out. With a statistical significance level of 0.1, one in 10 results could be false. What should the significance level be? If you want to change it, run through the previous steps again and check your conclusions.
- (A) Even with a significance level of 10% (alpha = 0.1) for test, according to the Bonferri Correction (alpha' = alpha/test_num) the thershold for a single test to be classified as significant is obviously (alpha' = 0.1/10 [since we have 10 tests]) -> alpha' = 0.01. Since the test failed to reject the hypothesis for alpha = 0.05 then there should be no difference here since alpha = 0.01 is even smaller this time making it harder for the test to reject the null hypothesis!

Reviewer's comment v.1

👍 Correct!

Reviewer's comment v.1

✍ Ameer, however, I would suggest and additional job here. Based on the project description, we should also "Compare the results with the combined results for the control groups (A1 + A2)". Please add this part. We probably won't have significant differences either, but we have to take this step.

Student's comment v.1

Sounds good! I added a couple new blocks of code and made some changes to the main function. And then added the test for combined groups with B

CORRECTED

Reviewer's comment v.2

👍 Thank you for fixing that. The new code and results are correct. Perfect job here! We could also use more universal function here (please cehck how I have set the success and trials values here):

In [41]:

```
# the three reviewers cells:
def check_hypothesis(data, group1, group2, event, alpha=0.05):
    success1 = data.query('group in @group1 and event_name == @event')['device_id'].nunique
    success2 = data.query('group in @group2 and event_name == @event')['device_id'].nunique

    trials1 = data.query('group in @group1')['device_id'].nunique()
    trials2 = data.query('group in @group2')['device_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,"and groups", group1,group2)
    else:
        print("Failed to reject the null hypothesis for",event,"and groups", group1,group2
```

In [42]:

```
for v in flog_data['event_name'].unique():
    check_hypothesis(flog_data, 'A1', 'B', v, alpha = 0.05)
```

```
p-value:  0.8264294010087645
Failed to reject the null hypothesis for Tutorial and groups A1 B
p-value:  0.2949721933554552
Failed to reject the null hypothesis for MainScreenAppear and groups A1 B
p-value:  0.20836205402738917
Failed to reject the null hypothesis for OffersScreenAppear and groups A1 B
p-value:  0.07842923237520116
Failed to reject the null hypothesis for CartScreenAppear and groups A1 B
p-value:  0.2122553275697796
Failed to reject the null hypothesis for PaymentScreenSuccessful and groups
A1 B
```

In [43]:

```
for v in flog_data['event_name'].unique():
    check_hypothesis(flog_data, ('A1','A2'), 'B', v, alpha = 0.05)
```

```
p-value:  0.764862472531507
Failed to reject the null hypothesis for Tutorial and groups ('A1', 'A2') B
p-value:  0.29424526837179577
Failed to reject the null hypothesis for MainScreenAppear and groups ('A1',
'A2') B
p-value:  0.43425549655188256
Failed to reject the null hypothesis for OffersScreenAppear and groups ('A
1', 'A2') B
p-value:  0.18175875284404386
Failed to reject the null hypothesis for CartScreenAppear and groups ('A1',
'A2') B
p-value:  0.6004294282308704
Failed to reject the null hypothesis for PaymentScreenSuccessful and groups
('A1', 'A2') B
```

5.4 Summary

- We found the size of each of our samples (two control groups A1 and A2 and a test group B), reaching around 2.5k users each. We then proceeded to calculate the difference in significance between our control groups to make sure that these groups are as statistically significant which allows us to continue with our research.
- Once we confirmed that there is no difference in significance between groups A1 and A2, we checked the same null hypothesis (**There is no difference in statistical significance between the groups**) on A1, B and A2, B.
- The test we conducted failed to reject the null hypothesis! Suggesting that there's no difference between our control groups and test group, and by looking at the results we got at each step for each group we can confirm the test results.

In [44]:

```
events_pivot
```

Out[44]:

group	event_name	A1	A2	B
0	CartScreenAppear	1266	1238	1230
1	MainScreenAppear	2450	2476	2493
2	OffersScreenAppear	1542	1520	1531
3	PaymentScreenSuccessful	1200	1158	1181
4	Tutorial	278	283	279

The results for the 3 groups looked almost identical! That means there's no need to change the font for the entire app since it didn't have much effect on our users. (Unless its for free and has no expenses and actually looks much better than the old font then why not!)

Reviewer's comment v.1

👍 Correct!

6 General Summary

1. General Analytics: Most users lost were between the steps MainScreenAppear and OffersScreenAppear. Whether it is a technical issue, or a lose of interest or anything else. To increase the sales and the conversion rate, we'll have to look onto these two steps and see what could we change to fix the current issue. If there's no interesting offers, lets consider new offers that might sell more. If there's a techincal issue, then lets solve it and increase our sales already!
2. A/A/B Analysis: The null hypothesis (There is no difference between groups A1, A2 and B) was failed to be rejected in the 10 tests we conducted, suggesting that there was no difference between our control groups with the old font and our test group with the new fancy font they had. Unless changing the font costs no money and it actually looks better then there's no need to consider the idea as it didn't have any impact on our users!

Reviewer's comment v.1

👍 Well done! Overall your conclusions are correct.

Students's comment v.1

👍 Thanks for your feedback!

Reviewer's comment v.2

👍 Thank you for your job and feedback on my comments! Good luck!