Project Description

You work at a startup that sells food products. You need to investigate user behavior for the company's app. First study the sales funnel. Find out how users reach the purchase stage. How many users actually make it to this stage? How many get stuck at previous stages? Which stages in particular?

Then look at the results of an A/A/B test. The designers would like to change the fonts for the entire app, but the managers are afraid the users might find the new design intimidating. They decide to make a decision based on the results of an A/A/B test.

The users are split into three groups: two control groups get the old fonts and one test group gets the new ones. Find out which set of fonts produces better results.

Creating two A groups has certain advantages. We can make it a principle that we will only be confident in the accuracy of our testing when the two control groups are similar. If there are significant differences between the A groups, this can help us uncover factors that may be distorting the results. Comparing control groups also tells us how much time and data we'll need when running further tests.

Description of the data

Each log entry is a user action or an event.

- **EventName** — event name
- **DeviceIDHash** — unique user identifier
- **EventTimestamp** — event time
- **ExpId** — experiment number: 246 and 247 are the control groups, 248 is the test group

Table of contents

Part 1. Opening the data file and reading the general information

In [1]:
```
import pandas as pd
from scipy import stats as st
import datetime as dt
import numpy as np

import seaborn as sns
from matplotlib import pyplot as plt
import plotly.express as px
```

In [2]:
```
import sys
import warnings
if not sys.warnoptions:
       warnings.simplefilter("ignore")
```

In [3]:
```
!pip install plotly
```

```
Requirement already satisfied: plotly in c:\programdata\anaconda3\lib\site-packages (4.8.1)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages (from plotly) (1.15.0)
Requirement already satisfied: retrying>=1.3.3 in c:\programdata\anaconda3\lib\site-packages (from plotly) (1.3.3)
```

In [4]:
```
from plotly.subplots import make_subplots
import plotly.graph_objects as go
```

In [5]:
```
try:
    data = pd.read_csv('/datasets/logs_exp_us.csv', sep = '\t' )
except:
    data = pd.read_csv('logs_exp_us.csv', sep = '\t')
```

In [6]:
```
data.head()
```

Out[6]:

	EventName	DeviceIDHash	EventTimestamp	ExpId
0	MainScreenAppear	4575588528974610257	1564029816	246
1	MainScreenAppear	7416695313311560658	1564053102	246
2	PaymentScreenSuccessful	3518123091307005509	1564054127	248
3	CartScreenAppear	3518123091307005509	1564054127	248
4	PaymentScreenSuccessful	6217807653094995999	1564055322	248

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

```
<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
```

Conclusion: As we see there is no missing data, but columns' names and data types (EventName, EventTimestamp, ExpId) and some of data types should be changed, we will do it on the next step

[Back to table of contents]

Part 2. Preparing the data for analysis

Let's rename the columns in more convenient way:

In [8]:
```
data.columns = ['event_name', 'device_id', 'event_datetime', 'group']
data.head()
```

Out[8]:

	event_name	device_id	event_datetime	group
0	MainScreenAppear	4575588528974610257	1564029816	246
1	MainScreenAppear	7416695313311560658	1564053102	246
2	PaymentScreenSuccessful	3518123091307005509	1564054127	248
3	CartScreenAppear	3518123091307005509	1564054127	248
4	PaymentScreenSuccessful	6217807653094995999	1564055322	248

Now we will check for missing values and data types and correct the data if needed:

In [9]:
```
data.isna().sum()
```

Out[9]:
```
event_name        0
device_id         0
event_datetime    0
group             0
dtype: int64
```

In [10]:
```
# check this columns for typos

print(data.event_name.value_counts())
print()
print(data.group.value_counts())
```

```
MainScreenAppear           119205
OffersScreenAppear          46825
CartScreenAppear            42731
PaymentScreenSuccessful     34313
Tutorial                     1052
Name: event_name, dtype: int64

248    85747
246    80304
247    78075
Name: group, dtype: int64
```

In [11]:
```
data.event_name = data.event_name.astype('category')
data['event_datetime'] = pd.to_datetime(data['event_datetime'], unit = 's')
data.info()
```

```
<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  category
 1   device_id       244126 non-null  int64
 2   event_datetime  244126 non-null  datetime64[ns]
 3   group           244126 non-null  int64
dtypes: category(1), datetime64[ns](1), int64(2)
memory usage: 5.8 MB
```

In [12]:
```
data.head(10)
```

Out[12]:

	event_name	device_id	event_datetime	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
5	CartScreenAppear	6217807653094995999	2019-07-25 11:48:43	248
6	OffersScreenAppear	8351860793733343758	2019-07-25 14:50:42	246
7	MainScreenAppear	5682100281902512875	2019-07-25 20:14:37	246
8	MainScreenAppear	1850981295691852772	2019-07-25 20:31:42	247
9	MainScreenAppear	5407636962369102641	2019-07-26 03:35:12	246

Looks better. Now we will also add a date and time column and a separate column for dates

In [13]:
```
data['event_date'] = pd.to_datetime(data['event_datetime']).dt.date
data.head(10)
```

Out[13]:

	event_name	device_id	event_datetime	group	event_date
0	MainScreenAppear	4575588528974610257	2019-07-25 04:43:36	246	2019-07-25
1	MainScreenAppear	7416695313311560658	2019-07-25 11:11:42	246	2019-07-25
2	PaymentScreenSuccessful	3518123091307005509	2019-07-25 11:28:47	248	2019-07-25
3	CartScreenAppear	3518123091307005509	2019-07-25 11:28:47	248	2019-07-25
4	PaymentScreenSuccessful	6217807653094995999	2019-07-25 11:48:42	248	2019-07-25
5	CartScreenAppear	6217807653094995999	2019-07-25 11:48:43	248	2019-07-25
6	OffersScreenAppear	8351860793733343758	2019-07-25 14:50:42	246	2019-07-25
7	MainScreenAppear	5682100281902512875	2019-07-25 20:14:37	246	2019-07-25
8	MainScreenAppear	1850981295691852772	2019-07-25 20:31:42	247	2019-07-25
9	MainScreenAppear	5407636962369102641	2019-07-26 03:35:12	246	2019-07-26

In [14]:
```
#Let's check it for duplicates
print(data.info())
print()
data.drop_duplicates()
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244126 entries, 0 to 244125
Data columns (total 5 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   event_name      244126 non-null  category
 1   device_id       244126 non-null  int64
 2   event_datetime  244126 non-null  datetime64[ns]
 3   group           244126 non-null  int64
 4   event_date      244126 non-null  object
dtypes: category(1), datetime64[ns](1), int64(2), object(1)
memory usage: 7.7+ MB
None

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244126 entries, 0 to 244125
Data columns (total 5 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   event_name      244126 non-null  category
 1   device_id       244126 non-null  int64
 2   event_datetime  244126 non-null  datetime64[ns]
 3   group           244126 non-null  int64
 4   event_date      244126 non-null  object
dtypes: category(1), datetime64[ns](1), int64(2), object(1)
memory usage: 7.7+ MB
None
```

Conclusion: Columns' names are renamed, data types are changed and a date and time column and a separate column for dates are created. Missing data and duplicates are not found.

Back to table of contents

Part 3. Explorary data analysis

Let's take a look at numbers of events and unique users in the data:

In [15]:
```
raw_events = len(data)
print("Number of all events:", raw_events, '\n')
print('Number of unique events:', '\n')
print(data.event_name.value_counts())
```

```
Number of all events: 244126

Number of unique events:

MainScreenAppear           119205
OffersScreenAppear          46825
CartScreenAppear            42731
PaymentScreenSuccessful     34313
Tutorial                     1052
Name: event_name, dtype: int64
```

In [16]:
```
raw_users = data.device_id.nunique()
print("Number of users:", raw_users, '\n')
print('Number of of events per user:', '\n')
print(data.groupby('device_id').agg({'event_datetime': 'count'}).mean())
```

```
Number of users: 7551

Number of of events per user:

event_datetime    32.330287
dtype: float64
```

What period of time does the data cover? Let's find the maximum and the minimum date.

In [17]:
```
print("Frist event datetime:", data.sort_values(by = 'event_datetime').head(1).loc[0,'event_datetime'])
print()
print("Last event datetime:", data.sort_values(by = 'event_datetime').tail(1).loc[len(data)-1,'event_datetime'])
print()
print('The raw data covers', data.sort_values(by = 'event_datetime').tail(1).loc[len(data)-1,'event_datetime']
      - data.sort_values(by = 'event_datetime').head(1).loc[0,'event_datetime'])
```

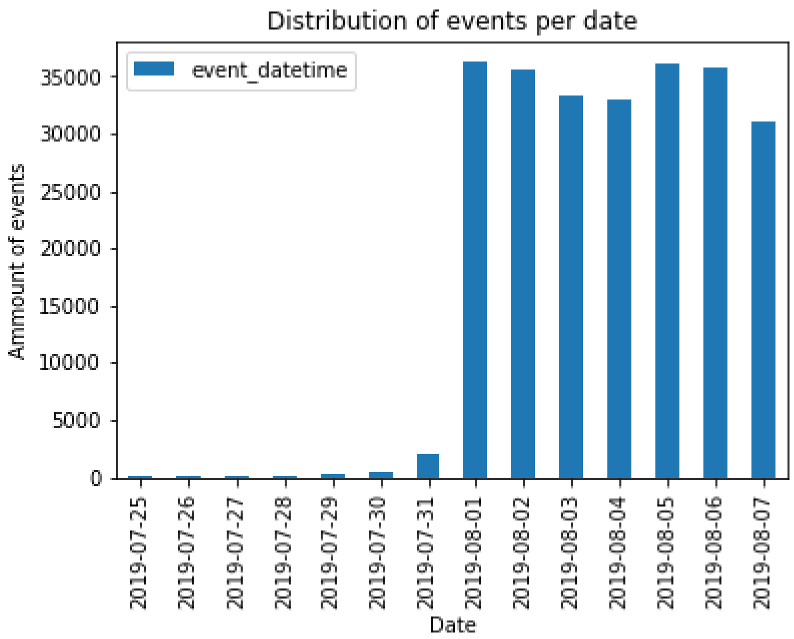
```
Frist event datetime: 2019-07-25 04:43:36

Last event datetime: 2019-08-07 21:15:17

The raw data covers 13 days 16:31:41
```

Now we will plot a histogram by date and time. Older events could end up in some users' logs for technical reasons, and this could skew the overall picture. We will find the moment at which the data starts to be complete and ignore the earlier section.

In [18]:
```
data.groupby('event_date').agg({'event_datetime': 'count'}).plot(kind='bar')
plt.xlabel('Date')
plt.ylabel('Ammount of events')
plt.title('Distribution of events per date')
plt.show()
```



As we see there are no full data before 2019-08-01 so the data contain information only for 7 days

In [19]:
```
#rasing data before 2019-08-01
query_time = dt.datetime(2019, 8, 1)
data = data.query('event_datetime > @query_time').reset_index()
data.head()
```

Out[19]:

	index	event_name	device_id	event_datetime	group	event_date
0	2828	Tutorial	3737462046622621720	2019-08-01 00:07:28	246	2019-08-01
1	2829	MainScreenAppear	3737462046622621720	2019-08-01 00:08:00	246	2019-08-01
2	2830	MainScreenAppear	3737462046622621720	2019-08-01 00:08:55	246	2019-08-01
3	2831	OffersScreenAppear	3737462046622621720	2019-08-01 00:08:58	246	2019-08-01
4	2832	MainScreenAppear	1433840883824088890	2019-08-01 00:08:59	247	2019-08-01

Let's check how many events and users we lost:

In [20]:
```
filtered_events = len(data)
print('New ammount of events:', filtered_events)
print('After filtering was lost', round(100 - (filtered_events * 100/ raw_users) , 2), '% of events')
print()
filtered_users = len(data.device_id.unique())
print('New ammount of users:', filtered_users)
print('After filtering was lost', round(100 - (filtered_users * 100 / raw_users), 2), '% of users')
```

```
New ammount of events: 241298
After filtering was lost -3095.58 % of events

New ammount of users: 7534
After filtering was lost 0.23 % of users
```

Acceptable results. Now we will make sure we have users from all three experimental groups:

In [21]:
```
print(data.group.value_counts())
```

```
248    84726
246    79425
247    77147
Name: group, dtype: int64
```

All 3 groups are still with us. Now we can finish EDA part and continue with funnels

[Back to table of contents]

Part 4. Building the event funnel

Now we will see what events are in the logs and their frequency of occurrence and sort them by frequency

In [22]:
```
print(data.event_name.value_counts())
```

```
MainScreenAppear           117431
OffersScreenAppear          46350
CartScreenAppear            42365
PaymentScreenSuccessful     34113
Tutorial                     1039
Name: event_name, dtype: int64
```

Let's find the number of users who performed each of these actions and sort the events by the number of users:

In [23]:
```
data_5_prep = data.groupby('device_id').agg({'event_name': 'nunique'}).reset_index()
data_5 = data_5_prep.query('event_name == 5') # here we use 5 in query because we have 5 different types of events

print('The number of users who performed each of the actions:', len(data_5))
```

```
The number of users who performed each of the actions: 466
```

In [24]:
```
sort_events = data.groupby('event_name').agg({'device_id': 'nunique'}).reset_index().sort_values(by =
            'device_id', ascending = False).reset_index().drop('index', axis = 1)
sort_events
```

Out[24]:

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

Now it is tim to calculate the proportion of users who performed the action at least once

In [25]:
```
data_at_least_1 = data_5_prep.query('event_name > 0')
print("Proportion of users who performed the action at least once is", round(len(data_at_least_1) / len(data_5_prep), 2))
```

```
Proportion of users who performed the action at least once is 1.0
```

Every user performed at least one action (which was obvious because we had no missing data in 'event_name' column)

Let's think about the order the actions took place. Are all of them part of a single sequence?

Indeed every user has his own "experience". Users can enter the app by tapping on the icon or by direct link to a product page (not every 7534 users appear in MainScreenAppear category but only 7419 of them), far not every user watches tutorial. There are users that buy several goods so their action sequences contain several 'OffersScreenAppear' -> 'CartScreenAppear' -> 'PaymentScreenSuccessful' chains. Even in this chains there are posibilities for payment errors so not everyone comes to 'PaymentScreenSuccessful'. Everyone is literally UNIQUE.

Now we will build the event funnel to find the share of users that proceed from each stage to the next:

In [26]:
```
data_funnel = data.pivot_table(index=['event_name', 'group'], values= 'event_datetime', aggfunc = 'count').unstack()
data_funnel.columns = data_funnel.columns.droplevel([1])
data_funnel.columns = ['246', '247', '248']
data_funnel = data_funnel.sort_values(by = '246', ascending = False)
data_funnel = data_funnel.reset_index()
data_funnel
```

Out[26]:

	event_name	246	247	248
0	MainScreenAppear	37708	39123	40600
1	OffersScreenAppear	14773	15182	16395
2	CartScreenAppear	14711	12456	15198
3	PaymentScreenSuccessful	11910	10043	12160
4	Tutorial	323	343	373

In [27]:
```
data_funnel = data_funnel.head(4) #Erase 'Tutorial' since it isn't a stage

stages = ["Main Screen Appear ", "Offers Screen Appear ", "Cart Screen Appear ", "Payment Screen Successful "]

g246 = pd.DataFrame(dict(number=data_funnel['246'], stage=stages))
g246['group'] = '246'

g247 = pd.DataFrame(dict(number=data_funnel['247'], stage=stages))
g247['group'] = '247'

g248 = pd.DataFrame(dict(number=data_funnel['248'], stage=stages))
g248['group'] = '248'

funnel_df = pd.concat([g246, g247, g248], axis=0)
```

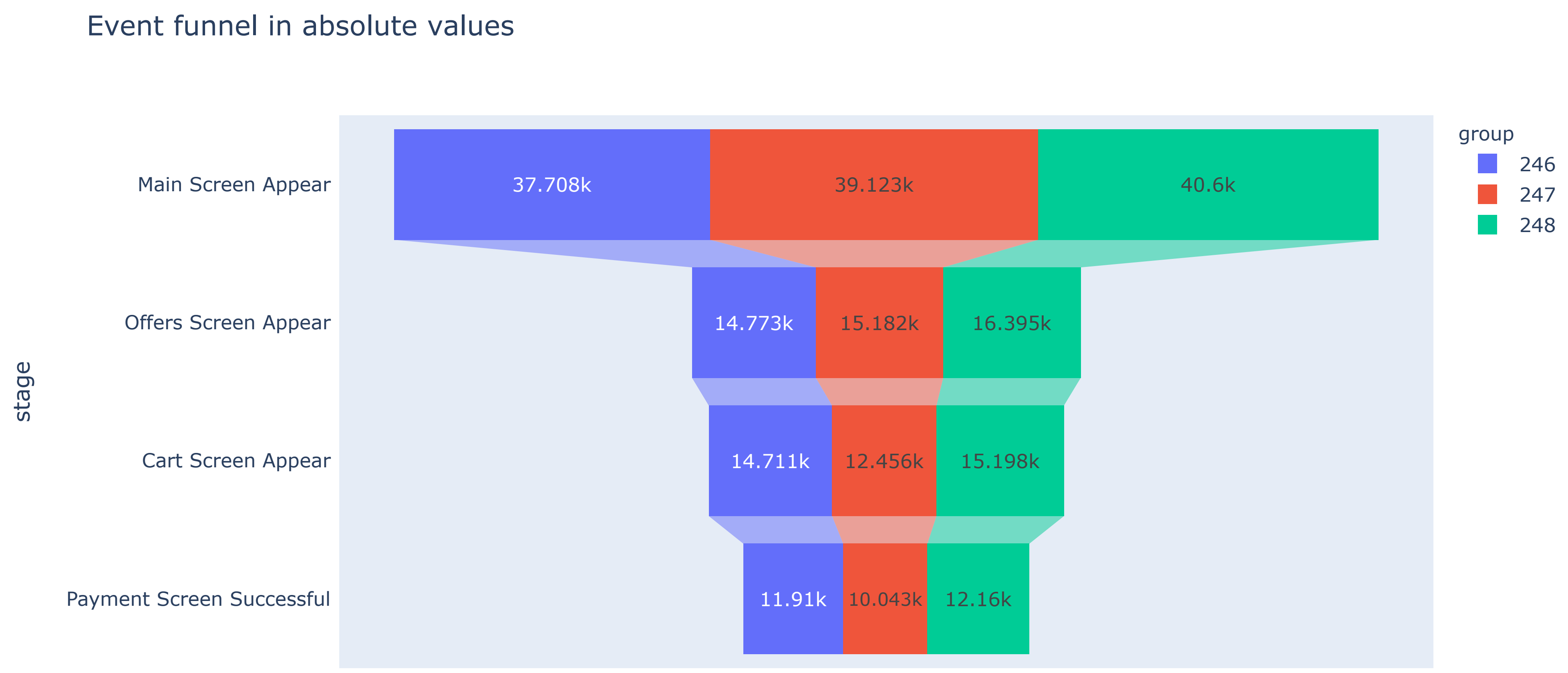
In [28]:
```python
try:
    fig = px.funnel(funnel_df, x='number', y='stage', color='group', title='Event funnel in absolute values')
    fig.show()

except:
    fig = make_subplots(rows=1, cols=5,subplot_titles=("group 246",'',"group 247",'',"group 248"))
    #I add here 2 empty cols with empty titles because otherwise satges' names cover the graphs


    fig.add_trace(
        go.Funnel(
        y = g246['stage'],
        x = g246['number'],
        textposition = "inside",
        textinfo = "value+percent previous",
        marker = {"color": "#ce5a57"}
    ),
        row=1, col=1
    )

    fig.add_trace(
        go.Funnel(
        y = g247['stage'],
        x = g247['number'],
        textposition = "inside",
        textinfo = "value+percent previous",
            marker = {"color": "#78a5a3"}
    ),
        row=1, col=3
    )

    fig.add_trace(
        go.Funnel(
        y = g248['stage'],
        x = g248['number'],
        textposition = "inside",
        textinfo = "value+percent previous",
            marker = {"color": "#e1b16a"}
    ),
        row=1, col=5
    )
    fig.update_layout(showlegend=False,height=400, width=1000, title_text="Groups' funnels")
    fig.show()
```



In [29]:
```python
print('Share of users that make the entire journey is:',
      round(sort_events.loc[3,'device_id'] / sort_events.loc[0,'device_id'] , 3))
```

```
Share of users that make the entire journey is: 0.477
```

As we see on the funnel chart it is the very first stage - between "Main Screen Appear" and "Offers Screen Appear". Only 39% of MainScreenAppear-users come to OffersScreenAppear while other stages have more then 80% retention. Maybe it is worth to check the main screen of the app for bugs with icons or simply make the interface more convenient?

Conclusion:

- The number of users who performed each of the actions: 466

- Proportion of users who performed the action at least once is 1.0 - Every user performed at least one action (which was obvious because we had no missing data in 'event_name' column)
- "MainScreenAppear" is the most popular event while "Tutorial" is the least one
- We loose the most users on the very first stage - between "Main Screen Appear" and "Offers Screen Appear"
- Share of users that make the entire journey from "MainScreenAppear" to "PaymentScreenSuccessful" is: 0.477

[Back to table of contents]

Part 5. Studying the results of the experiment

- How many users are there in each group?

In [30]:
```
sample_246 = data.query('group == 246')['device_id'].nunique()
sample_247 = data.query('group == 247')['device_id'].nunique()
sample_248 = data.query('group == 247')['device_id'].nunique()

print('In group №246', sample_246 , 'users and it takes',
      round(sample_246 / (sample_246 + sample_247 + sample_248), 3), "of all users", '\n')
print('In group №247', sample_247 , 'users and it takes',
      round(sample_247 / (sample_246 + sample_247 + sample_248), 3), "of all users", '\n')
print('In group №248', sample_248 , 'users and it takes',
      round(sample_248 / (sample_246 + sample_247 + sample_248), 3), "of all users", '\n')
```

```
In group №246 2484 users and it takes 0.331 of all users

In group №247 2513 users and it takes 0.335 of all users

In group №248 2513 users and it takes 0.335 of all users
```

We have two control groups in the A/A test, where we check our mechanisms and calculations. Let's see if there is a statistically significant difference between samples 246 and 247. In statistics we work with 2 hyposeses - H0 and H1. **H0** is always stated with an equal sign, our one will look like **"There is no statistically significant difference between samples 246 and 247"** and **H1** is **"There is a statistically significant difference between samples 246 and 247"**

In [31]:
```
import math

def check_hypothesis(group1,group2, alpha=0.05):

    successes1=data.query('group in @group2')['device_id'].nunique()
    successes2=data.query('group in @group2')['device_id'].nunique()

    #for trials we can go back to original df or used a pre-aggregated data
    trials = data['device_id'].nunique()

    #proportion for success in the first group
    p1 = successes1/trials

    #proportion for success in the second group
    p2 = successes2/trials

    # proportion in a combined dataset
    p_combined = (successes1 + successes2) / (trials)

    difference = p1 - p2

    z_value = difference / math.sqrt(p_combined * (1 - p_combined) * (2/trials))


    distr = st.norm(0, 1)


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

    if (p_value < alpha):
        print("Reject H0 for groups",group1,group2)
    else:
        print("Fail to Reject H0 for groups" ,group1,group2)
```

In [32]:
```
check_hypothesis('246','247', alpha=0.05)
```

```
Fail to Reject H0 for groups 246 247
```

Conclusion: We can't reject the hypothesis that there is no statistically significant difference between samples 246 and 247

Let's now select the most popular event. In each of the control groups, find the number of users who performed this action and find their share.

In [33]:
```python
data_funnel['all'] = data_funnel['246'] + data_funnel['247'] + data_funnel['248']
data_funnel['share_246'] = round(data_funnel['246'] / data_funnel['all'], 2)
data_funnel['share_247'] = round(data_funnel['247'] / data_funnel['all'], 2)
data_funnel['share_248'] = round(data_funnel['248'] / data_funnel['all'], 2)
data_funnel
```

Out[33]:

	event_name	246	247	248	all	share_246	share_247	share_248
0	MainScreenAppear	37708	39123	40600	117431	0.32	0.33	0.35
1	OffersScreenAppear	14773	15182	16395	46350	0.32	0.33	0.35
2	CartScreenAppear	14711	12456	15198	42365	0.35	0.29	0.36
3	PaymentScreenSuccessful	11910	10043	12160	34113	0.35	0.29	0.36

Let's check whether the difference between the groups is statistically significant and repeat the procedure for all other events:

In [34]:
```python
def check_hypothesis_for_events(group1,group2, event, alpha=0.05):

    successes1=data[data['event_name']==event].query('group in @group2')['device_id'].nunique()
    successes2=data[data['event_name']==event].query('group in @group2')['device_id'].nunique()

    #for trials we can go back to original df or used a pre-aggregated data

    trials1 = data.query('group in @group1')['device_id'].nunique()
    trials2 = data.query('group in @group2')['device_id'].nunique()

    #proportion for success in the first group
    p1 = successes1/trials1

   #proportion for success in the second group
    p2 = successes2/trials2


    # proportion in a combined dataset
    p_combined = (successes1 + successes2) / (trials1 + trials2)

    difference = p1 - p2

    z_value = difference / math.sqrt(p_combined * (1 - p_combined) * (1/trials1 + 1/trials2))


    distr = st.norm(0, 1)


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

    #print('p-value: ', p_value)

    if (p_value < alpha):
        print("Reject H0 for",event, 'and groups',group1,group2)
    else:
        print("Fail to Reject H0 for", event,'and groups',group1,group2)
```

In [35]:
```python
for event in sort_events['event_name']:
    check_hypothesis_for_events('246', '247', event, alpha=0.05)
    print()
```

```
Reject H0 for MainScreenAppear and groups 246 247

Fail to Reject H0 for OffersScreenAppear and groups 246 247

Fail to Reject H0 for CartScreenAppear and groups 246 247

Fail to Reject H0 for PaymentScreenSuccessful and groups 246 247

Fail to Reject H0 for Tutorial and groups 246 247
```

Conclusion: As we see in 4 of 5 events we can't reject the hypothesis that there is no statistically significant difference between samples 246 and 247

Now we will do the same thing for the group with altered fonts:

In [36]:
```
for group in ['246', '247']:
    print("RESULTS FOR GROUP", group,'\n')
    for event in sort_events['event_name']:
        check_hypothesis_for_events('248', group, event, alpha=0.025)

        # here we take aplha / 2 according to Bonferoni correction

        print()
    print('***********')
```

```
RESULTS FOR GROUP 246

Reject H0 for MainScreenAppear and groups 248 246

Fail to Reject H0 for OffersScreenAppear and groups 248 246

Fail to Reject H0 for CartScreenAppear and groups 248 246

Fail to Reject H0 for PaymentScreenSuccessful and groups 248 246

Fail to Reject H0 for Tutorial and groups 248 246

***********
RESULTS FOR GROUP 247

Reject H0 for MainScreenAppear and groups 248 247

Fail to Reject H0 for OffersScreenAppear and groups 248 247

Fail to Reject H0 for CartScreenAppear and groups 248 247

Fail to Reject H0 for PaymentScreenSuccessful and groups 248 247

Fail to Reject H0 for Tutorial and groups 248 247

***********
```

Conclusion: As we see in 4 of 5 events we can't reject the hypothesis that there is no statistically significant difference between samples for both pares 246-248 and 247-248. The result we've got are the same as for groups 246-247 since the event that give us rejecting of H0 is 'MainScreenAppear'

Let's also check our results for alpha = 0.01

In [37]:
```
check_hypothesis('246','247', alpha=0.01)
```

```
Fail to Reject H0 for groups 246 247
```

We still can't reject the hypothesis that there is no statistically significant difference between samples 246 and 247

Our 0.01 results here are the same as 0.05 results. There is still no statistically significant difference between samples 246 and 247 in 4 of 5 cases

In [38]:
```
for group in ['246', '247']:
    print("RESULTS FOR GROUP", group,'\n')
    for event in sort_events['event_name']:
        check_hypothesis_for_events('248', group, event, alpha=0.005)

        # here we take aplha / 2 according to Bonferoni correction

        print()
    print('***********')
```

```
RESULTS FOR GROUP 246

Reject H0 for MainScreenAppear and groups 248 246

Fail to Reject H0 for OffersScreenAppear and groups 248 246

Fail to Reject H0 for CartScreenAppear and groups 248 246

Fail to Reject H0 for PaymentScreenSuccessful and groups 248 246

Fail to Reject H0 for Tutorial and groups 248 246

***********
RESULTS FOR GROUP 247

Fail to Reject H0 for MainScreenAppear and groups 248 247

Fail to Reject H0 for OffersScreenAppear and groups 248 247

Fail to Reject H0 for CartScreenAppear and groups 248 247

Fail to Reject H0 for PaymentScreenSuccessful and groups 248 247

Fail to Reject H0 for Tutorial and groups 248 247

***********
```

We have got such interesting results - for pares 246-247 and 246-248 everything is the same but not for 247-248! In this pare we also get "Fail to Reject H0 for

MainScreenAppear". It means that MainScreenAppear samples for groups 247 and 248 are close enough for significance level 0.01 but far enough for significance level 0.05.

[Back to table of contents](#)

General conclusion

Event funnels results:

- The number of users who performed each of the actions: 466
- Proportion of users who performed the action at least once is 1.0 - Every user performed at least one action (which was obvious because we had no missing data in 'event_name' column)
- "MainScreenAppear" is the most popular event while "Tutorial" is the least one
- We loose the most users on the very first stage - between "Main Screen Appear" and "Offers Screen Appear"
- Share of users that make the entire journey from "MainScreenAppear" to "PaymentScreenSuccessful" is: 0.477

The experiments results:

- In group №246 2484 users, in group №247 2513 users and in group №248 2537 users
- We can't reject the hypothesis that there is no statistically significant difference between samples 246 and 247 in general and in 4 of 5 event sample tests (for both values of alpha - 0.05 and 0.01)
- For pares 246-247 and 246-248 the results of event sample tests (for both values of alpha - 0.05 and 0.01) are the same but not for 247-248! In this pare we also get "Fail to Reject H0 for MainScreenAppear". It means that MainScreenAppear samples for groups 247 and 248 are close enough for significance level 0.01 but far enough for significance level 0.05.

There is no need to change the fonts in the app but maybe it is worth to check the main screen of the app for bugs with icons or simply make the interface more convenient because as we see on the funnel chart, we loose the greatest ammount of users on the very first stage - between "Main Screen Appear" and "Offers Screen Appear". Only 39% of MainScreenAppear-users come to OffersScreenAppear while other stages have more then 80% retention.

[Back to table of contents](#)

Requirements

In [39]:
```
pip freeze > requirements.txt
```

```
Note: you may need to restart the kernel to use updated packages.
```