1 progect description

Let's imagine that we work in a startup that sells food products and our task is toinvestigate user behavior for the company's app:

- study the sales funnel,
- look at the results of an A/A/B test,
- formulate statistical hypotheses.

The conclusions that I will draw from it will help me improve the company conversion by interpreting user behavior and clarifying the results of statistical tests.

Table of Contents

2 Step 1. Downloading the data

We will use 9 libraries:

- pandas: for data processing
- numpy, math: for calculations
- plotly express: for data visualisation

- datetime: for working with data
- scipy for hypotheses testing
- sys, warnings: for not showing the warnings
- iterals: for nice combinations

In [113]:
```
import pandas as pd
import numpy as np
import datetime
from datetime import timedelta
from datetime import datetime
import plotly.express as px
import plotly.graph_objects as go
import math as mth
from scipy import stats as st
import re
import itertools
from plotly.offline import iplot, init_notebook_mode
import matplotlib.pyplot as plt
import matplotlib as mpl
import sys
import warnings
if not sys.warnoptions:
    warnings.simplefilter("ignore")
import seaborn as sns
pd.set_option('display.max_columns', 500)
pd.set_option('display.max_rows', 500)
plt.style.use('fivethirtyeight')
```

Let's set some parameters for ploting

In [114]:
```
mpl.rcParams['lines.linewidth'] = 2
mpl.rcParams["figure.figsize"] = [8, 6]
mpl.rcParams.update({"axes.grid": True, "grid.color": "grey"})
mpl.rcParams['image.cmap'] = 'gray'
mpl.rcParams['figure.dpi'] = 80
mpl.rcParams['savefig.dpi'] = 100
mpl.rcParams['font.size'] = 12
mpl.rcParams['legend.fontsize'] = 'large'
mpl.rcParams['figure.titlesize'] = 'medium'
```

In [115]:
```
try:
    logs_exp = pd.read_csv('/datasets/logs_exp_us.csv', sep='\t', dtype={'EventName': 'category',
                                                                          'ExpId': 'category'})  # practicum p
except:
    try:
        logs_exp = pd.read_csv('./datasets/logs_exp_us.csv', sep='\t', dtype={'EventName': 'category',
                                                                               'ExpId': 'category'})  # local
    except:
        try:
            logs_exp = pd.read_csv('https://code.s3.yandex.net//datasets/logs_exp_us.csv', sep='\t', dtype={'
                                                                                                           '

        except FileNotFoundError:
            print('Ooops, the dateset not found.')

        except pd.errors.EmptyDataError:
            print('Ooops, the dataset is empty.')
```

Let's downcast our data so it wouldn't take to much space

In [116]:
```
logs_exp['DeviceIDHash'] = pd.to_numeric(logs_exp['DeviceIDHash'], downcast='integer')
logs_exp['EventTimestamp'] = pd.to_numeric(logs_exp['EventTimestamp'], downcast='integer')

logs_exp.info(memory_usage='deep')
```

```
<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  category
 1   DeviceIDHash    244126 non-null  int64
 2   EventTimestamp  244126 non-null  int32
 3   ExpId           244126 non-null  category
dtypes: category(2), int32(1), int64(1)
memory usage: 3.3 MB
```

2.0.1 Conclusion

We successfully opened the dataset. The dataset contains 244126 lines, 2 category columns, 2 integer columns. Let's se how we can preprocess it

3 Step 2. Preprocessing the data

3.0.1 Renaming the columns

In [117]:
```
logs_exp.columns = ['event_name', 'user_id', 'timestamp', 'experiment_id']
```

In [118]:
```
logs_exp.user_id.nunique()
```

Out[118]:
```
7551
```

3.0.2 Checking for missing values and data types

In [119]:
```
logs_exp.describe(include='all')
```

Out[119]:

	event_name	user_id	timestamp	experiment_id
count	244126	2.441260e+05	2.441260e+05	244126
unique	5	NaN	NaN	3
top	MainScreenAppear	NaN	NaN	248
freq	119205	NaN	NaN	85747
mean	NaN	4.627568e+18	1.564914e+09	NaN
std	NaN	2.642425e+18	1.771343e+05	NaN
min	NaN	6.888747e+15	1.564030e+09	NaN
25%	NaN	2.372212e+18	1.564757e+09	NaN
50%	NaN	4.623192e+18	1.564919e+09	NaN
75%	NaN	6.932517e+18	1.565075e+09	NaN
max	NaN	9.222603e+18	1.565213e+09	NaN

In [120]:
```
round(logs_exp.experiment_id.value_counts(normalize=True) * 100,2)
```

Out[120]:
```
248    35.12
246    32.89
247    31.98
Name: experiment_id, dtype: float64
```

3.0.3 Duplicates

In [121]:
```
for i in logs_exp[logs_exp.duplicated()].columns:
    print(i, ':', logs_exp[logs_exp.duplicated()][i].nunique())
```

```
event_name : 5
user_id : 237
timestamp : 352
experiment_id : 3
```

In [122]:
```
print(f'procentage of duplicates is {round(logs_exp.duplicated().sum() / logs_exp.shape[0] * 100,2)}%')
```

```
procentage of duplicates is 0.17%
```

In [123]:
```
logs_exp = logs_exp.drop_duplicates()
```

3.0.4 date and time column + dates column

In [124]:
```
logs_exp['timestamp'] = logs_exp.timestamp.apply(lambda x:datetime.fromtimestamp(x))
logs_exp['date'] = logs_exp['timestamp'].astype('datetime64[D]')
```

3.0.5 Conclusion

We found really small amount of duplicated lines, in real worlв we should report this, cause it means, something is wrong obtained data. We found no missing values, but created the 'timestamp' and 'date' columns which will help us later on. Also we renamed column names to officially acceped naming format. Also the proportions for experiments look equal

4 Step 3. Data Discovery

4.0.1 How many events are in the logs?

In [125]:
```
1 print(f'we have {logs_exp.event_name.nunique()} unique events and {logs_exp.shape[0]} events in general in t
```

we have 5 unique events and 243713 events in general in the logs dataset

4.0.2 How many users are in the logs?

In [126]:
```
1 logs_exp.user_id.nunique()
```

Out[126]: 7551

4.0.3 What's the average number of events per user?

In [127]:
```
1 round(logs_exp.groupby('user_id')['event_name'].count().mean(),2)
```

Out[127]: 32.28

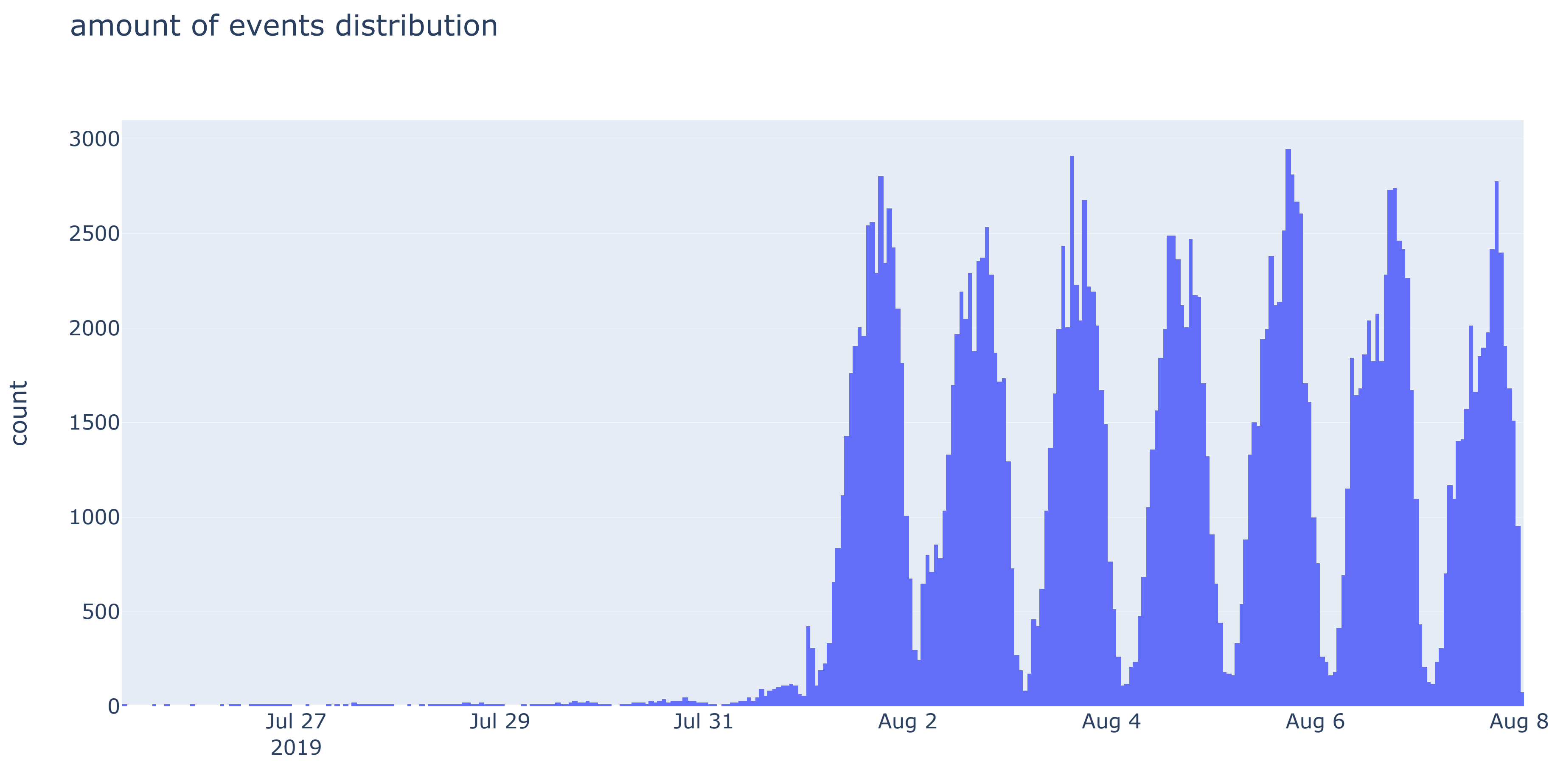
4.0.4 What period of time does the data cover? Find the maximum and the minimum date.

In [128]:
```
1 print(f"The research period is from {logs_exp.date.min()} to {logs_exp.date.max()} covering {(logs_exp.date
```

The research period is from 2019-07-25 00:00:00 to 2019-08-08 00:00:00 covering 15.0 days

4.0.5 Date and time histogram

In [129]:
```
1 fig = px.histogram(logs_exp, x="timestamp", title='amount of events distribution')
2
3 fig.show()
```

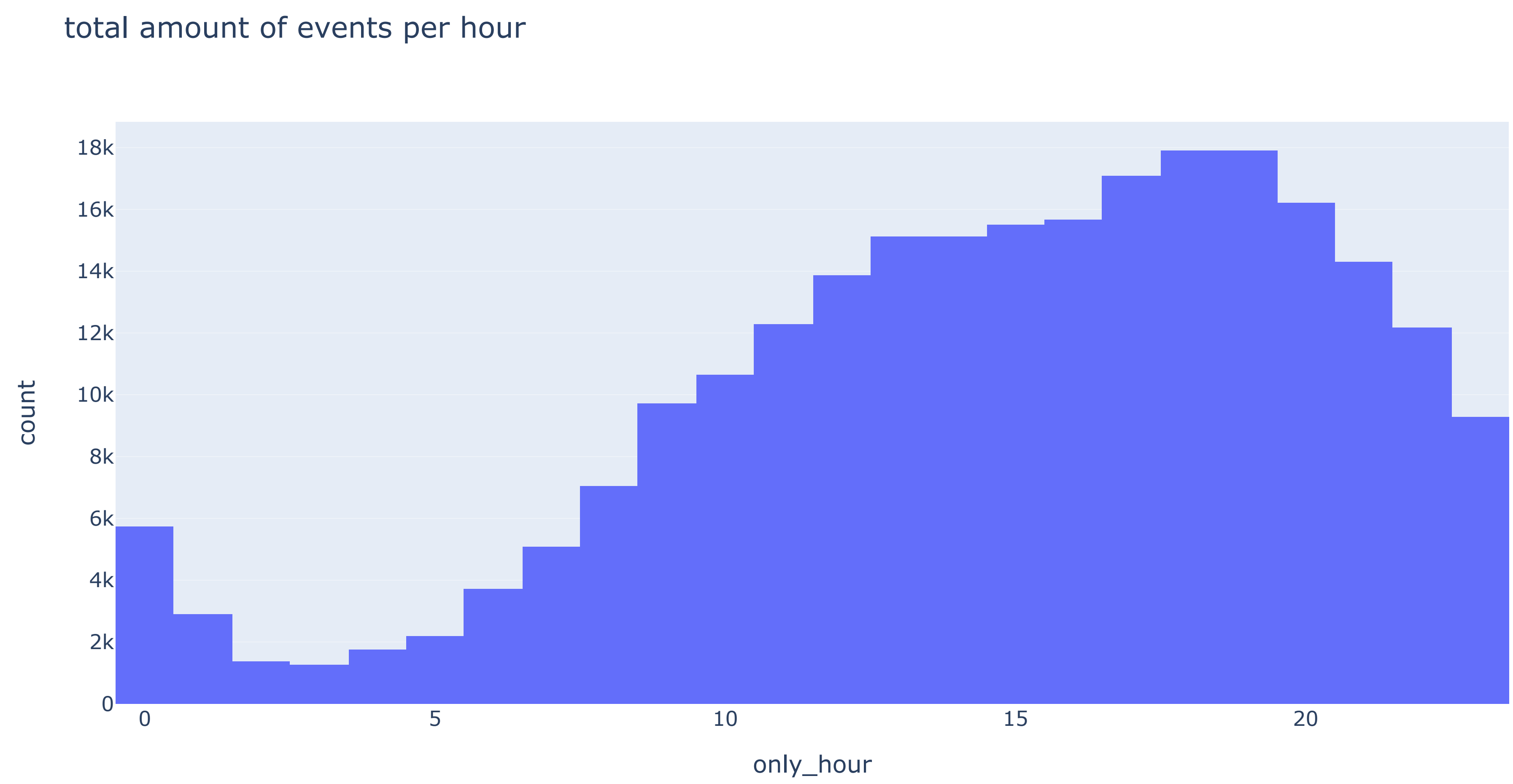


We can see really clear that August data has it's own pattern, but I want to get sure that the 1st August also belong to the pattern model. Also I want to check the hour events distribution, because I beleive that this falls happen due to night hour lower visitor flow and peaks - due to daytime visitors. What wa the reason for having such low values before August - possibly technical problems.

In [130]:
```
1 logs_exp['hour'] = logs_exp.timestamp.dt.round('H')
```

In [131]:
```
1 logs_exp['only_hour'] = logs_exp['hour'].dt.hour
```

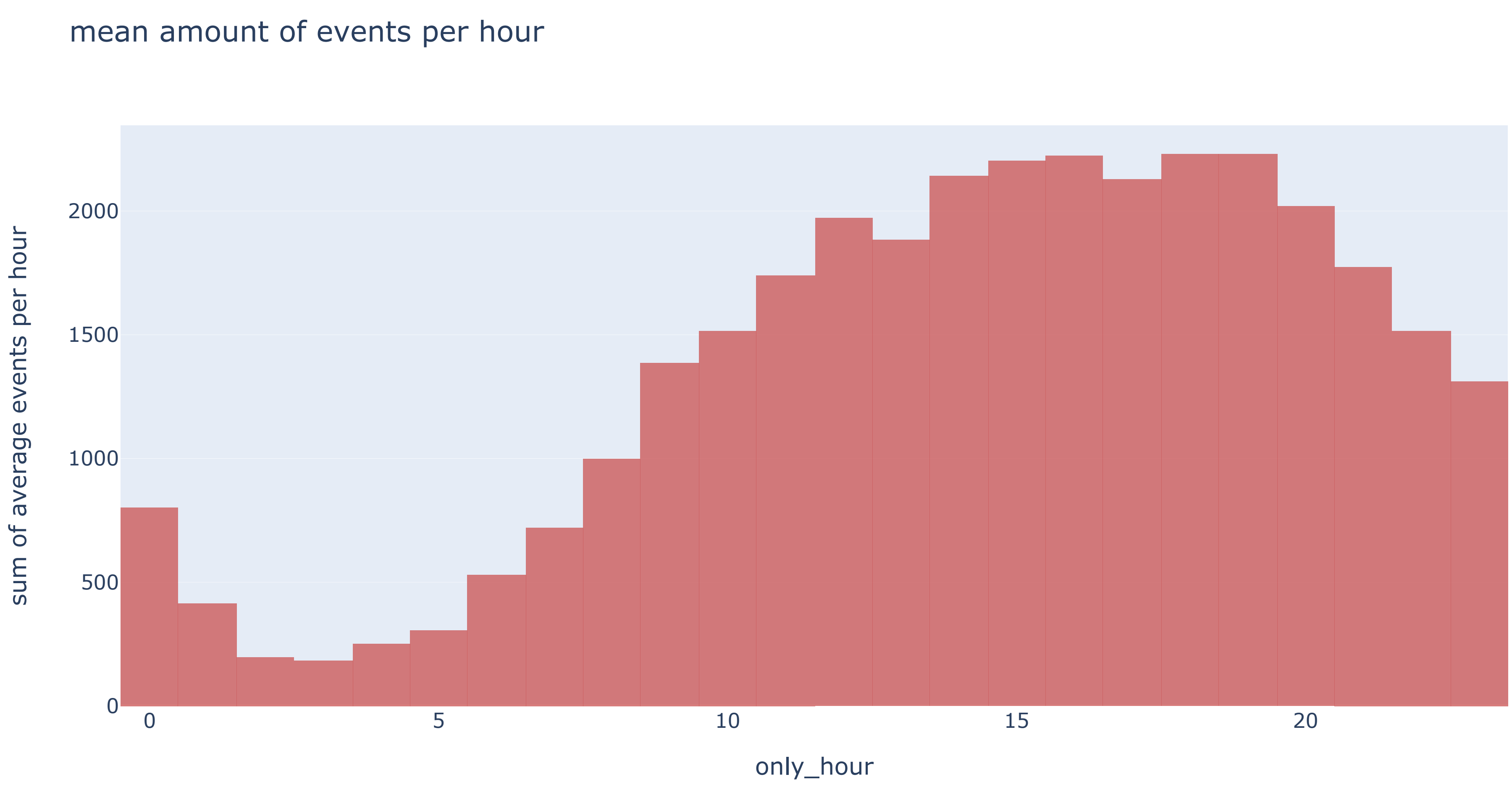
In [132]:
```
fig = px.histogram(logs_exp, x="only_hour", title='total amount of events per hour')
fig.show()
```



As I expected, the night hours have the lowest amount of events. But I want to get to the mean amountnt of events when the system recieved data correctly, not the whole sum of events.

In [133]:
```
min_amount= 81
normal_hours = logs_exp.groupby(['date','only_hour'])['event_name'].count().unstack()
event_per_hour = pd.DataFrame(normal_hours[normal_hours > min_amount].mean(axis=0))
event_per_hour.columns = ['amount']
```

In [134]:
```
fig = px.histogram(event_per_hour, x=event_per_hour.index, y='amount', nbins=24,
                   labels={'amount':'average events per hour'}, # can specify one label per df column
                   opacity=0.8,
                   color_discrete_sequence=['indianred'], title='mean amount of events per hour') # color o

fig.show()
```



Now I have awerage amount of events for eventsery hour and can compare it to the recieved values.

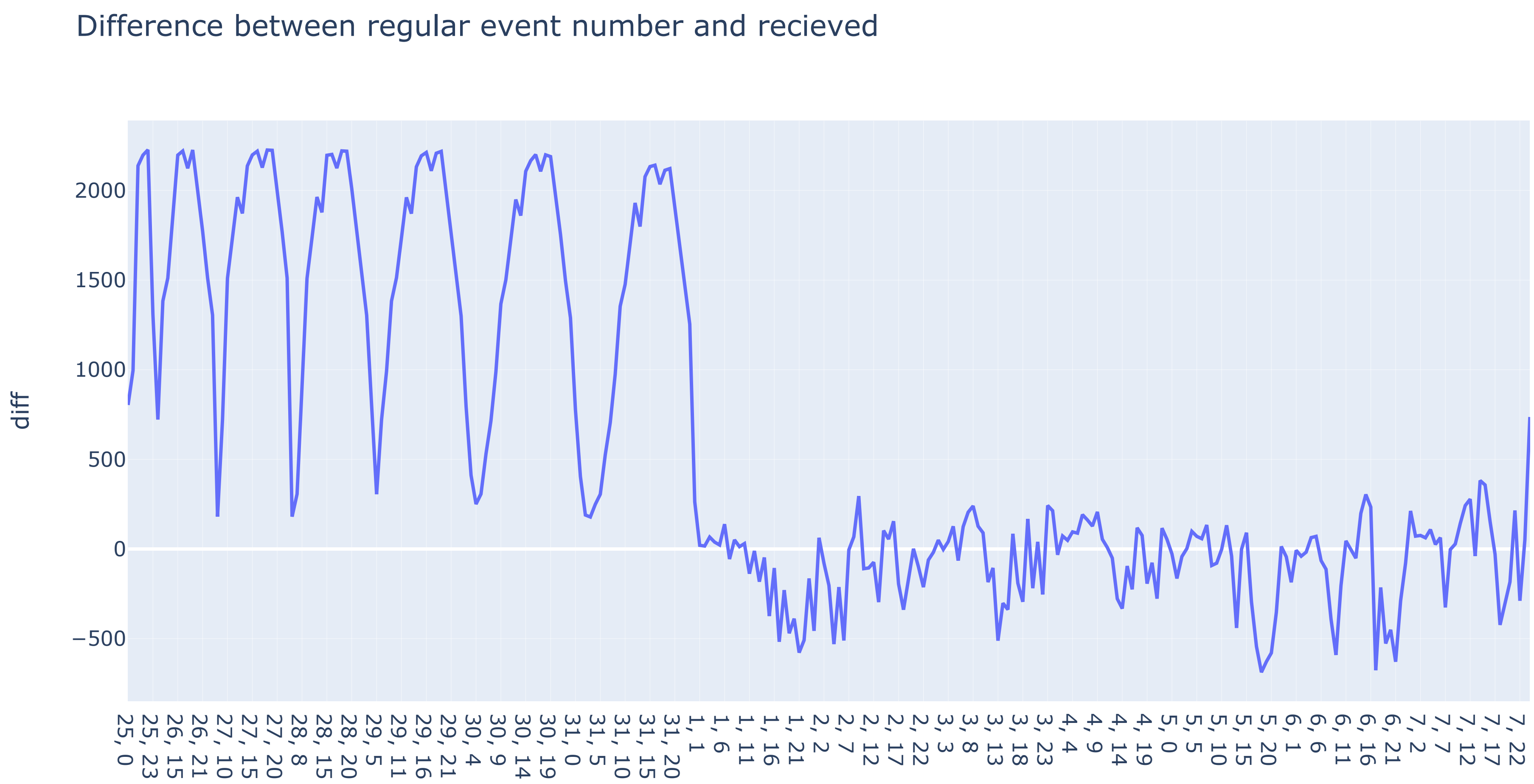
In [135]:
```
per_day_per_hour = logs_exp.groupby(['date','only_hour'])['event_name'].count().reset_index(drop=False)

per_day_per_hour = per_day_per_hour.merge(
    event_per_hour, how='left', left_on='only_hour', right_on=event_per_hour.index)

per_day_per_hour.columns = ['date', 'hour', 'events_num', 'mean_events_num']
per_day_per_hour['diff'] = per_day_per_hour['mean_events_num'] -per_day_per_hour['events_num']

per_day_per_hour
```

99	2019-07-31	9	30	1384.000000	1354.000000
100	2019-07-31	10	38	1514.142857	1476.142857
101	2019-07-31	11	38	1740.285714	1702.285714
102	2019-07-31	12	38	1968.285714	1930.285714
103	2019-07-31	13	83	1881.625000	1798.625000
104	2019-07-31	14	64	2140.714286	2076.714286
105	2019-07-31	15	66	2199.000000	2133.000000
106	2019-07-31	16	80	2220.857143	2140.857143
107	2019-07-31	17	95	2129.375000	2034.375000
108	2019-07-31	18	115	2227.500000	2112.500000
109	2019-07-31	19	106	2227.750000	2121.750000
110	2019-07-31	20	121	2017.375000	1896.375000

In [136]:
```
per_day_per_hour['when'] = per_day_per_hour['date'].dt.day.astype('str') + ', ' + per_day_per_hour['hour'].a
```

Let's plot a line that would how how the recieved values varied from average

In [137]:
```
fig = px.line(per_day_per_hour,x='when', y="diff", title='Difference between regular event number and recie
fig.show()
```



4.0.6 Conclusion

This dataset contains have 5 unique events and 243713 events in general in the logs dataset with 7551 having nearly 32 events per one. The data in the dataset describes 2 weeks, but not all the days contain properly recieved information, maybe due to technical reasons. I analysed the data and founf an average hamount of events for every hour to compare with the recieved data. I can see on the graph that starting from 2019-08-01 data looks 'normal' and I choose this date ti be the fist point of properly distributed data. When the data was close to the 'average' in July, in was due to regularly low numbers events that is proven by other plots. So the data really represents just the period from 2019-08-01 to 2019-08-08.

4.0.7 Did you lose many events and users when excluding the older data?

In [138]:
```
good_date = '2019-08-01'

filtered_logs = logs_exp[logs_exp['date'] >= good_date]

bad_data = logs_exp[logs_exp['date'] < good_date]

print(f'After getting rid of bad data (but recived during the half of the whole research time), we lost only

print(f'We also lost {bad_data.user_id.nunique()} users')
```

```
After getting rid of bad data (but recived during the half of the whole research time), we lost only lost 0.8
2% of data
We also lost 1319 users
```

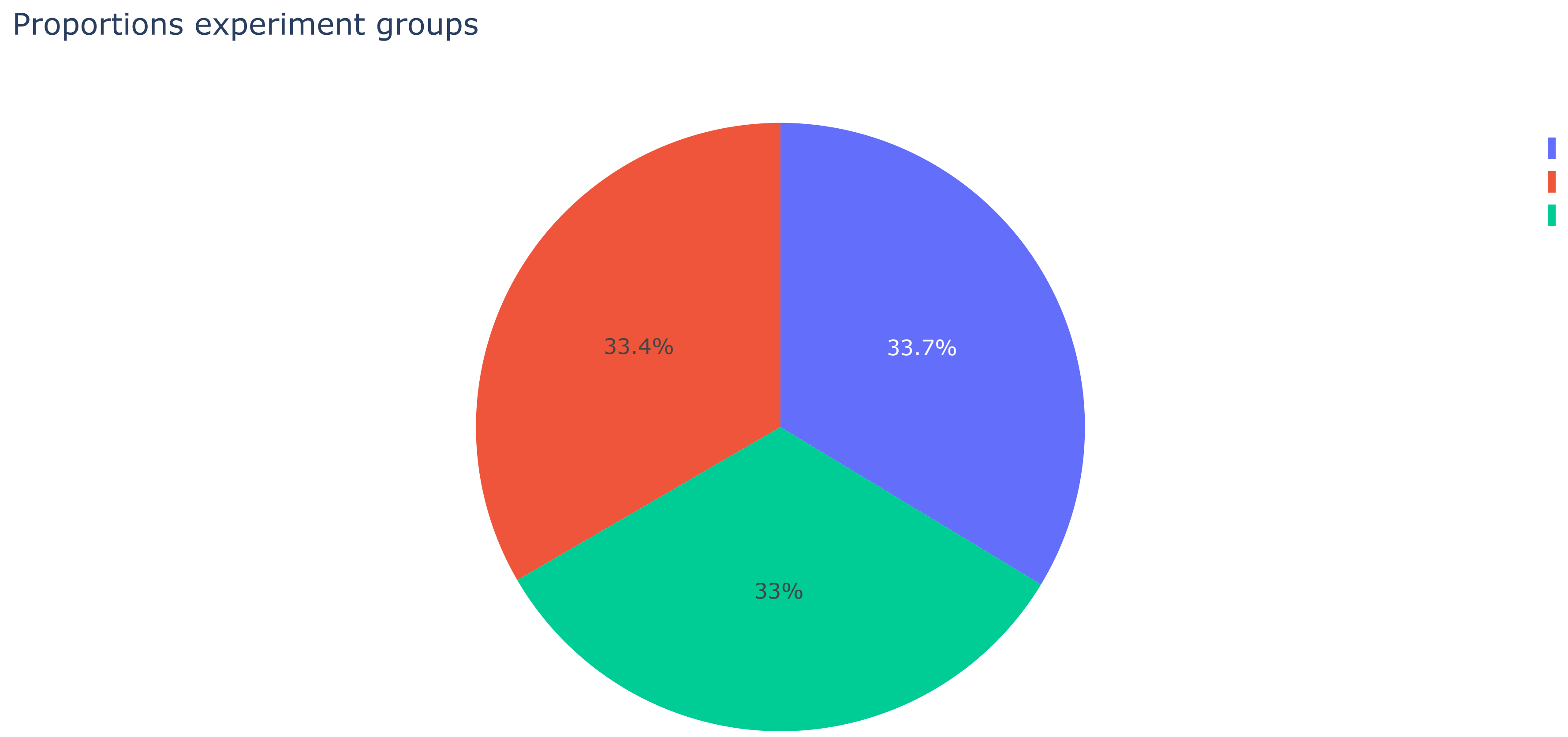
I also want to see, what are the proportions of the remained events.

In [139]:
```
bad_data.event_name.value_counts(normalize=True) * 100
```

Out[139]:
```
MainScreenAppear           60.935143
CartScreenAppear           16.339869
OffersScreenAppear         13.926596
PaymentScreenSuccessful     8.396179
Tutorial                    0.402212
Name: event_name, dtype: float64
```

4.0.8 Make sure you have users from all three experimental groups.

In [140]:
```
users_in_group = pd.DataFrame(filtered_logs.groupby('experiment_id')['user_id'].nunique())

fig = px.pie(users_in_group,  values='user_id', names=users_in_group.index, title='Proportions experiment g
fig.show()
```

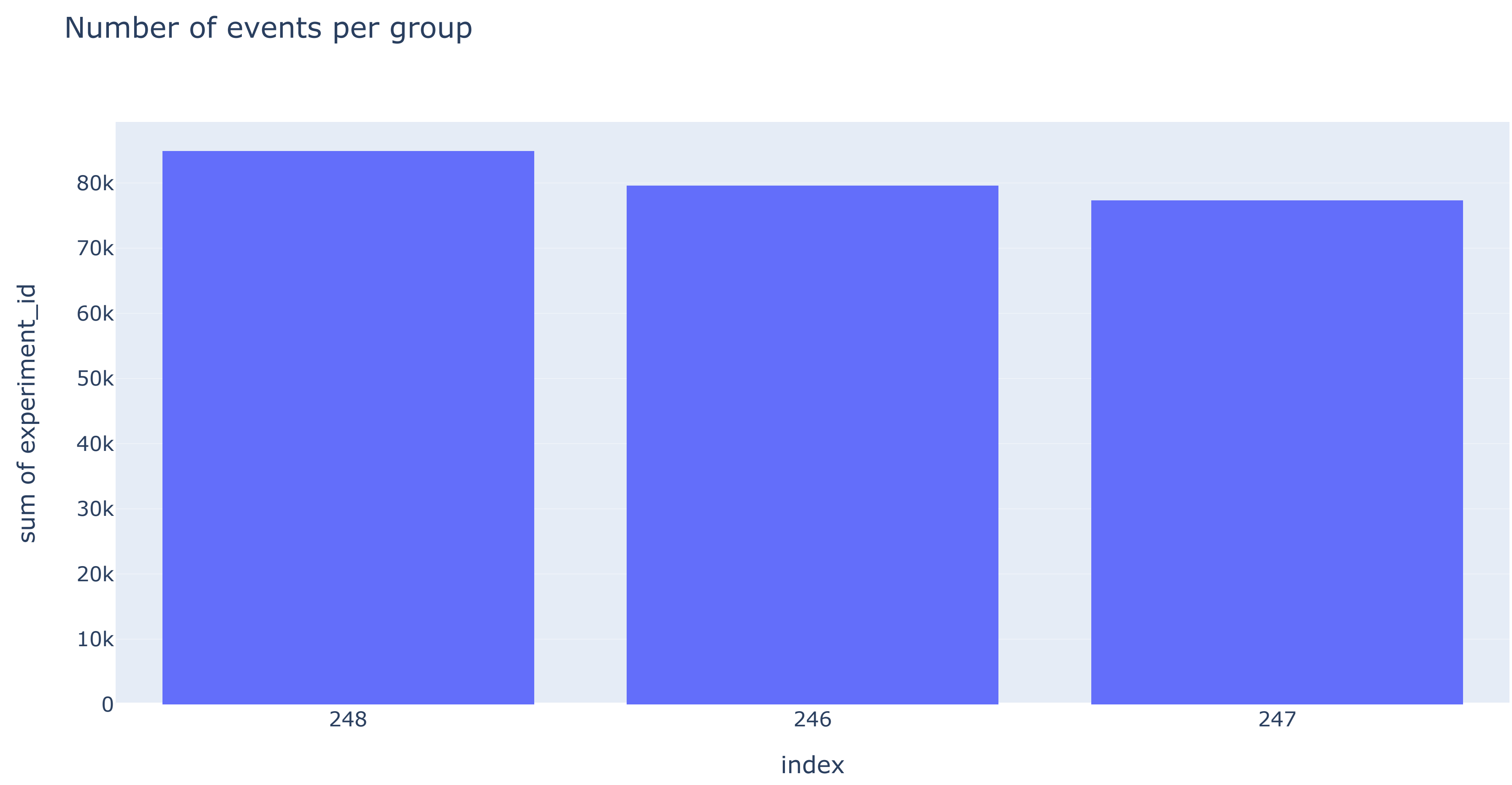


In [141]:
```
filtered_logs.groupby('experiment_id')['user_id'].nunique()
```

Out[141]:
```
experiment_id
246    2484
247    2517
248    2537
Name: user_id, dtype: int64
```

In [142]:
```
events_count = pd.DataFrame(filtered_logs.experiment_id.value_counts())
```

In [143]:
```
fig = px.histogram(events_count, x=events_count.index, y = 'experiment_id',
                   title='Number of events per group')
fig.show()
```



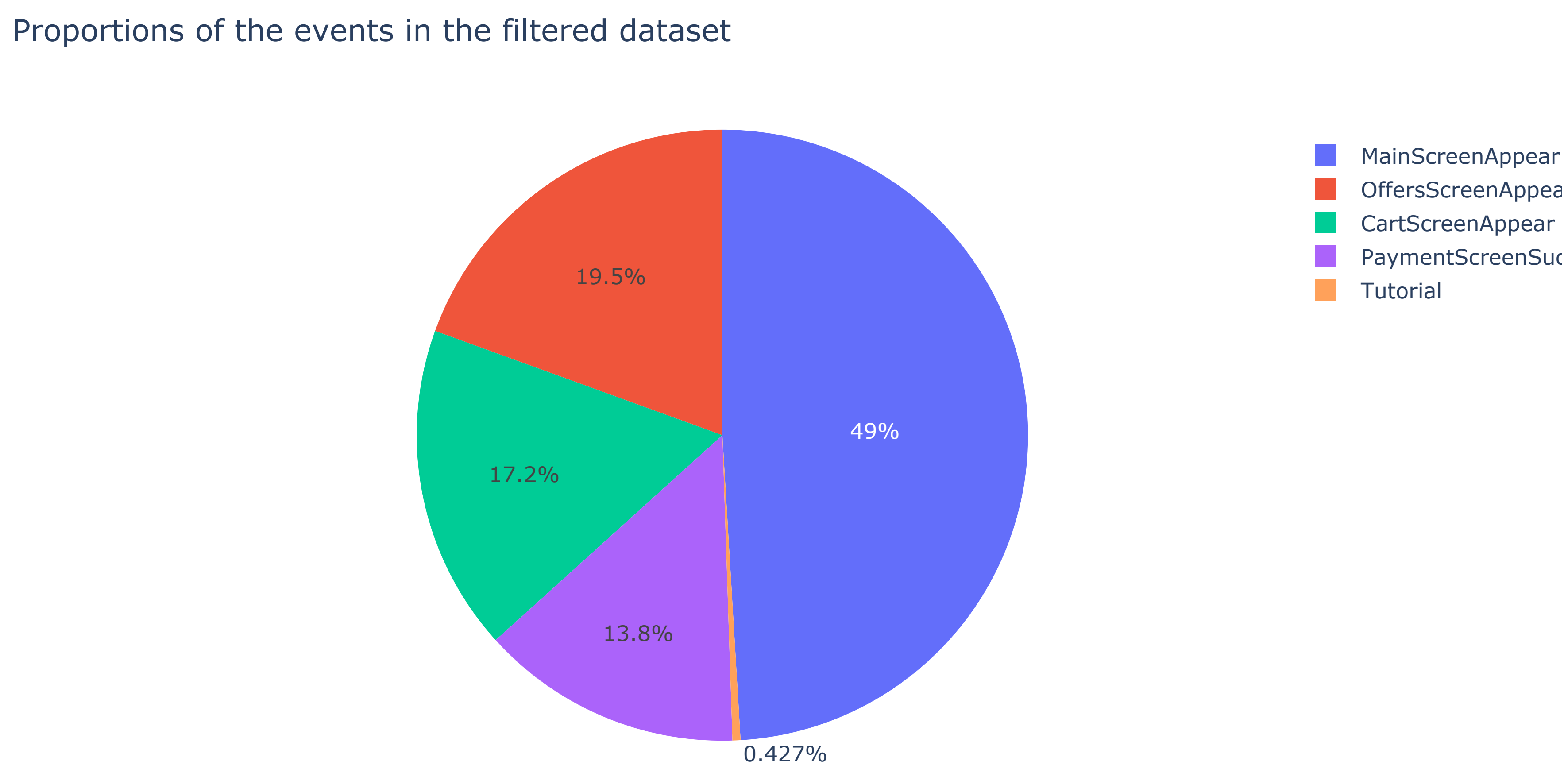
4.0.9 Conclusion

After getting rid of bad data (but recived during the half of the whole research time), we lost only lost 0.82% of data, that included visits of 1319 users. Also I checked the proportions of the event types and their order remained the same. Also I checked how many users are left in the filtered data, but they are still nicely distributed.

5 Step 4. The event funnel

5.0.1 Frequency of event occurrence

In [144]:
```
fig = px.pie(filtered_logs, values='user_id', names='event_name', title='Proportions of the events in the fi
fig.show()
```



In [145]:
```
print('Amount of events:\n',filtered_logs.event_name.value_counts())
```

```
Amount of events:
 MainScreenAppear          117889
OffersScreenAppear          46531
CartScreenAppear            42343
PaymentScreenSuccessful     33951
Tutorial                     1010
Name: event_name, dtype: int64
```

In [146]:
```
filtered_logs.groupby('user_id')['event_name'].apply(lambda x: x.mode()).value_counts()
```

Out[146]:
```
MainScreenAppear          6035
OffersScreenAppear        1176
CartScreenAppear           741
PaymentScreenSuccessful    173
Tutorial                    29
Name: event_name, dtype: int64
```

5.0.2 Conclusion

We can see that MainScreenAppear is an undoubtle leader followed by OffersScreenAppear and CartScreenAppear that both have 2 time fewer users. It is also the most frequent event for 6035 users

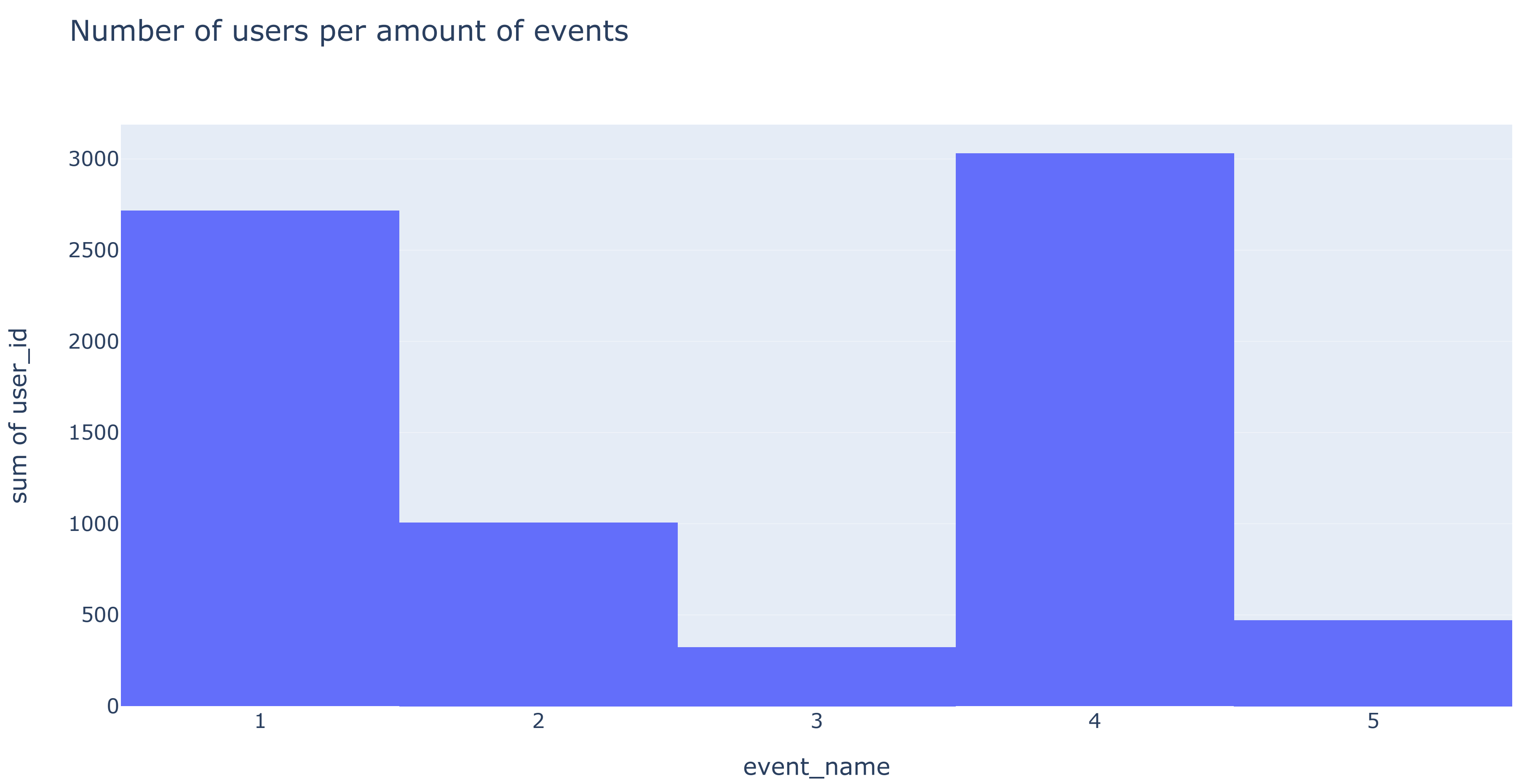
5.0.3 Users who performed each action

In [147]:
```
amount_of_events = pd.DataFrame(filtered_logs.groupby('user_id')['event_name'].nunique().reset_index().grou
amount_of_events
```

Out[147]:

event_name	user_id
1	2717
2	1006
3	319
4	3027
5	469

In [148]:
```
fig = px.histogram(amount_of_events, x=amount_of_events.index, y = 'user_id', nbins=5,
                   title='Number of users per amount of events')
fig.show()
```



In [149]:
```
user_events = filtered_logs.groupby('user_id')['event_name'].nunique().reset_index(drop=False)

user_events.columns = ['user_id', 'number_of_events']
```

In [150]:
```
users_5_actions = list(user_events[user_events['number_of_events'] == 5]['user_id'])

users_4_actions = list(user_events[user_events['number_of_events'] == 4]['user_id'])
```

5.0.4 Conclusion

In [151]:
```
print(f'We can see that there is the highest amount of users who did only 1 action - presumably MainScreenAp
```

```
We can see that there is the highest amount of users who did only 1 action - presumably MainScreenAppear and
ofthose who did 4 actions - all the neccasiare ones, but not the tutorial. So we have 3027 - 40.16% of all us
ers -  who performed all the 5 actions except for Tutorial and 469 including it.
```

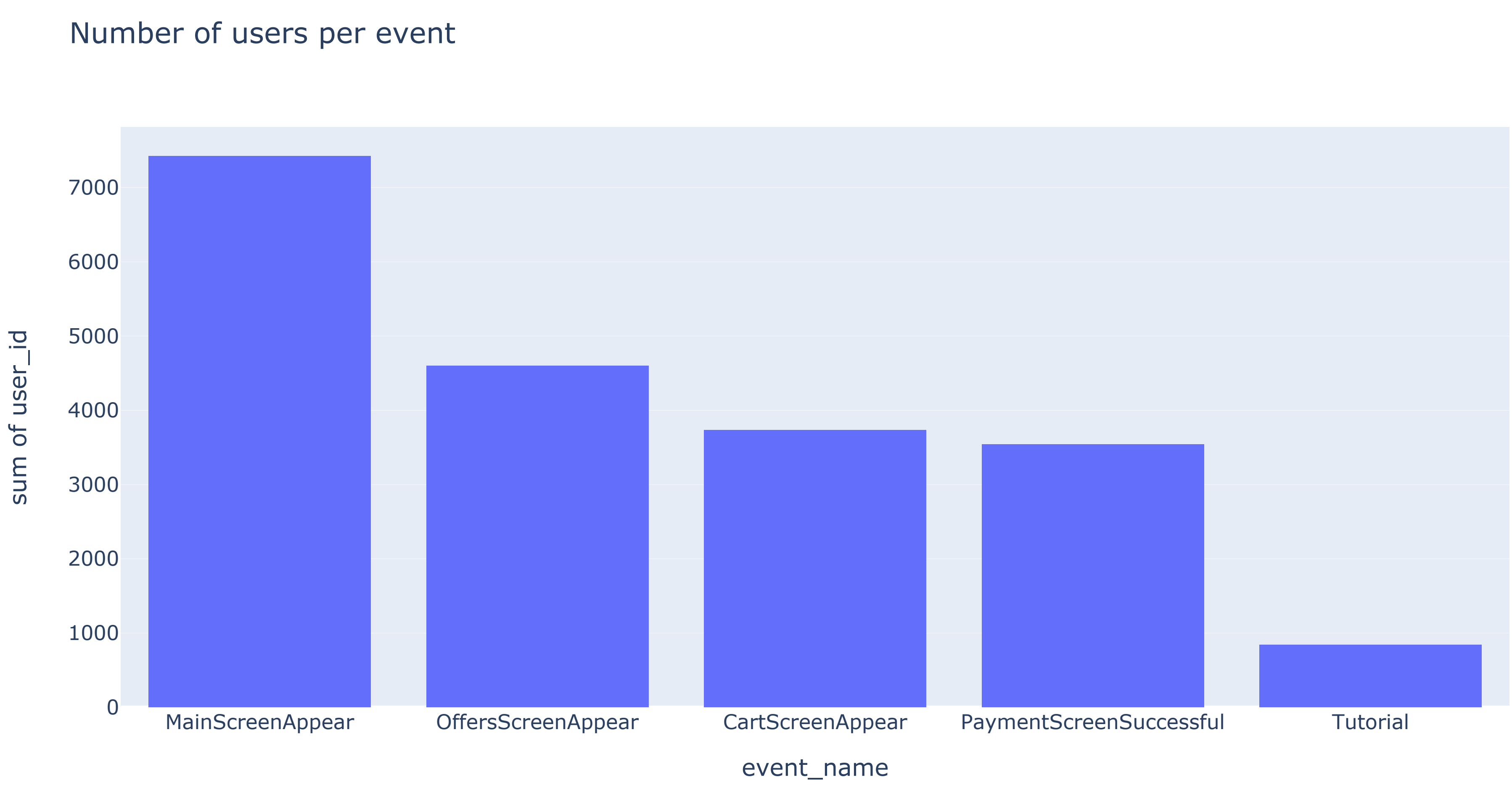
5.0.5 Sort the events by the number of users

In [152]:
```
number_of_users = pd.DataFrame(filtered_logs.groupby('event_name')['user_id'].nunique().sort_values(ascending
number_of_users
```

Out[152]:

	user_id
event_name	
MainScreenAppear	7423
OffersScreenAppear	4597
CartScreenAppear	3736
PaymentScreenSuccessful	3540
Tutorial	843

In [153]:
```
fig = px.histogram(number_of_users, x=number_of_users.index, y = 'user_id', nbins=5,
                   title='Number of users per event')
fig.show()
```



5.0.6 Conclusion

We can see that the number of users goes down on the PaymentScreenSuccessful section, but the amount of events as we saw above is still low. That mean that there areusers who make a lot of purchases

5.0.7 Proportion of users who performed the action at least once.

performed any action just once

In [154]:
```
users_1_action = list(user_events[user_events['number_of_events'] == 1]['user_id'])
```

In [155]:
```
print(f'We have {len(users_1_action)} users performed just one action, the MainScreenAppear presumably, its
```

```
We have 2717 users performed just one action, the MainScreenAppear presumably, its 36.04% of all users
```

In [156]:
```
(f1 But those who performed the action at least one and more actions are {round((filtered_logs.user_id.nunique()
```

```
 But those who performed the action at least one and more actions are 93.78%
```

5.0.8 Order of actions

I beleive that tutorial has such low values because it is not the necessaire part of events, it's extra. So the order is as follows

In [157]:
```
order = ['MainScreenAppear', 'OffersScreenAppear', 'CartScreenAppear', 'PaymentScreenSuccessful']
```

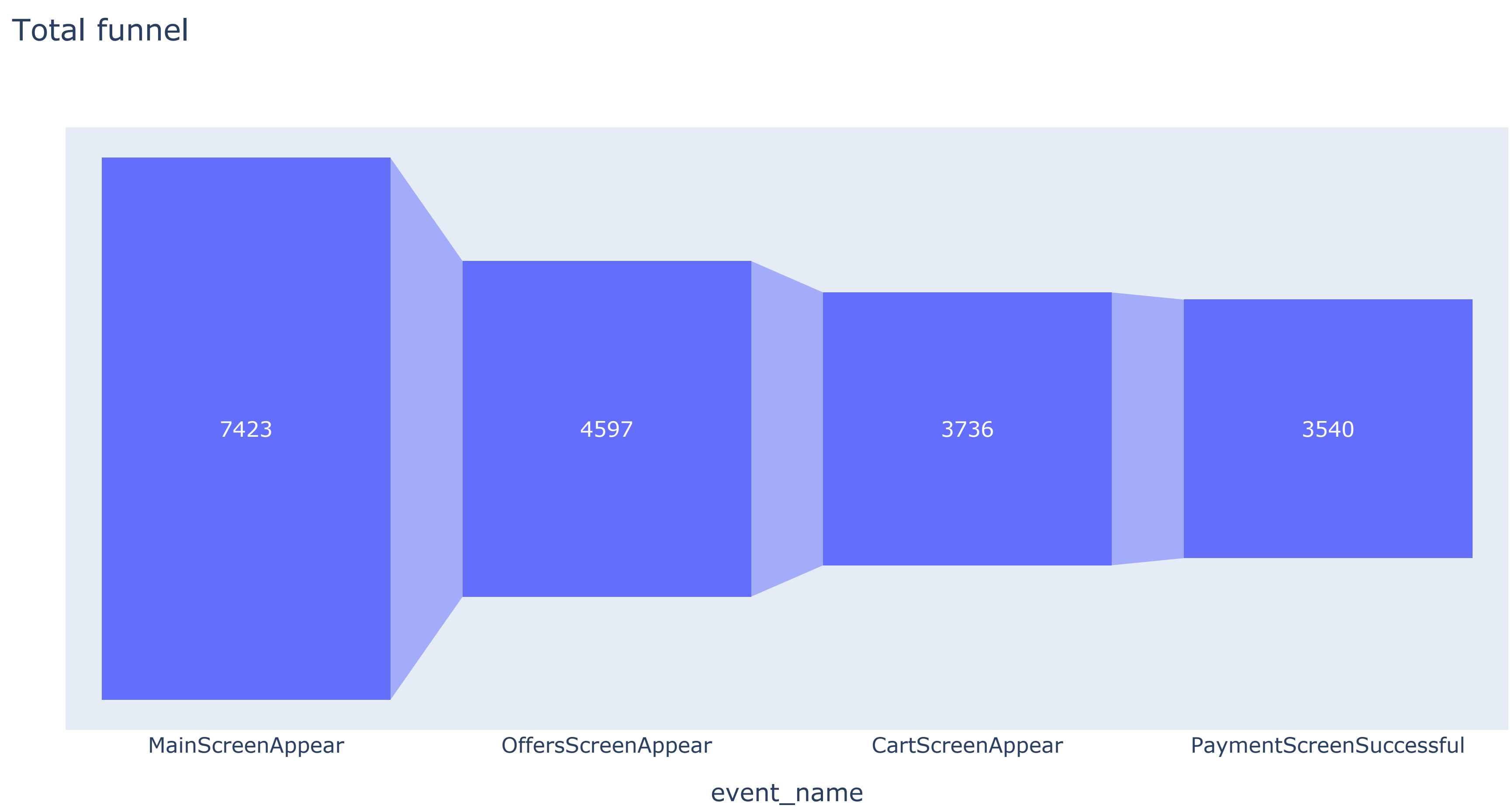
5.0.9 Event funnel

In [158]:
```
funnel = filtered_logs.groupby(
    'event_name')['user_id'].nunique().sort_values(ascending=False).reset_index()

funnel['pct'] = funnel.user_id.pct_change()

funnel = funnel.drop(funnel[funnel.event_name =='Tutorial'].index)

funnel
```

Out[158]:

	event_name	user_id	pct
0	MainScreenAppear	7423	NaN
1	OffersScreenAppear	4597	-0.380709
2	CartScreenAppear	3736	-0.187296
3	PaymentScreenSuccessful	3540	-0.052463

In [159]:
```
fig = px.funnel(funnel, x ='event_name', y = 'user_id', title='Total funnel')
fig.show()
```



In [160]:
```
nel_list = []

: exp in filtered_logs.experiment_id.unique():
print(exp)
funnel_ = filtered_logs[filtered_logs.experiment_id == exp]
funnel_ = pd.DataFrame(funnel_.groupby('event_name')['user_id'].nunique().sort_values( ascending=False).reset_:
funnel_['pct'] = funnel_.user_id.pct_change()
funnel_['group'] = exp
print(funnel_)
funnel_list.append(funnel_)
```

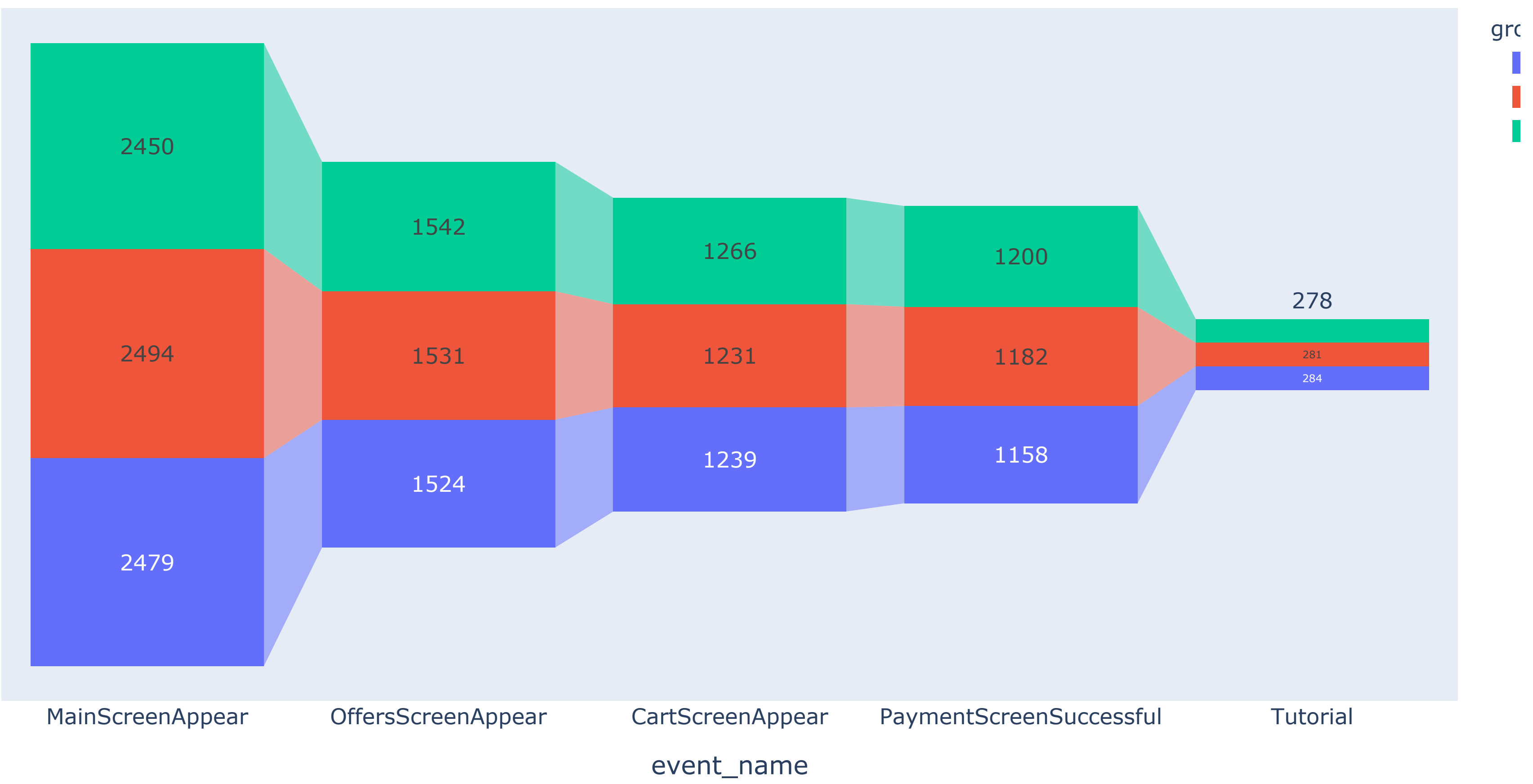
```
247
                event_name  user_id       pct  group
0          MainScreenAppear     2479       NaN    247
1        OffersScreenAppear     1524 -0.385236    247
2          CartScreenAppear     1239 -0.187008    247
3   PaymentScreenSuccessful     1158 -0.065375    247
4                  Tutorial      284 -0.754750    247
248
                event_name  user_id       pct  group
0          MainScreenAppear     2494       NaN    248
1        OffersScreenAppear     1531 -0.386127    248
2          CartScreenAppear     1231 -0.195950    248
3   PaymentScreenSuccessful     1182 -0.039805    248
4                  Tutorial      281 -0.762267    248
246
                event_name  user_id       pct  group
0          MainScreenAppear     2450       NaN    246
1        OffersScreenAppear     1542 -0.370612    246
2          CartScreenAppear     1266 -0.178988    246
3   PaymentScreenSuccessful     1200 -0.052133    246
4                  Tutorial      278 -0.768333    246
```

5.0.10 Conclusion

OffersScreenAppear has almost 38% lower events than MainScreenAppear in general and for every group. But further values varies a little

In [161]:
```
total_funnel = pd.concat(funnel_list, axis=0)
```

In [162]:
```
fig = px.funnel(total_funnel, x ='event_name', y = 'user_id', color='group')
fig.show()
```



5.0.11 Stage with highest lose rate

In [163]:
```
funnel[funnel.pct == funnel.pct.min()].event_name
```

Out[163]:
```
1    OffersScreenAppear
Name: event_name, dtype: category
Categories (5, object): ['CartScreenAppear', 'MainScreenAppear', 'OffersScreenAppear', 'PaymentScreenSuccessf
ul', 'Tutorial']
```

5.0.12 Share of users who make the entire journey from their first event to payment

In [164]:
```
every_event_per_user = filtered_logs.pivot_table(index='user_id', columns='event_name', values='timestamp',

every_event_per_user = every_event_per_user.drop('Tutorial', axis=1)

every_event_per_user = every_event_per_user.dropna(how='any')
```

In [165]:
```
print(f'We have {every_event_per_user.shape[0]} who had this whole journey. it is {round(every_event_per_use
```

```
We have 3430 who had this whole journey. it is 45.5% of all users
```

5.0.13 Conclusion

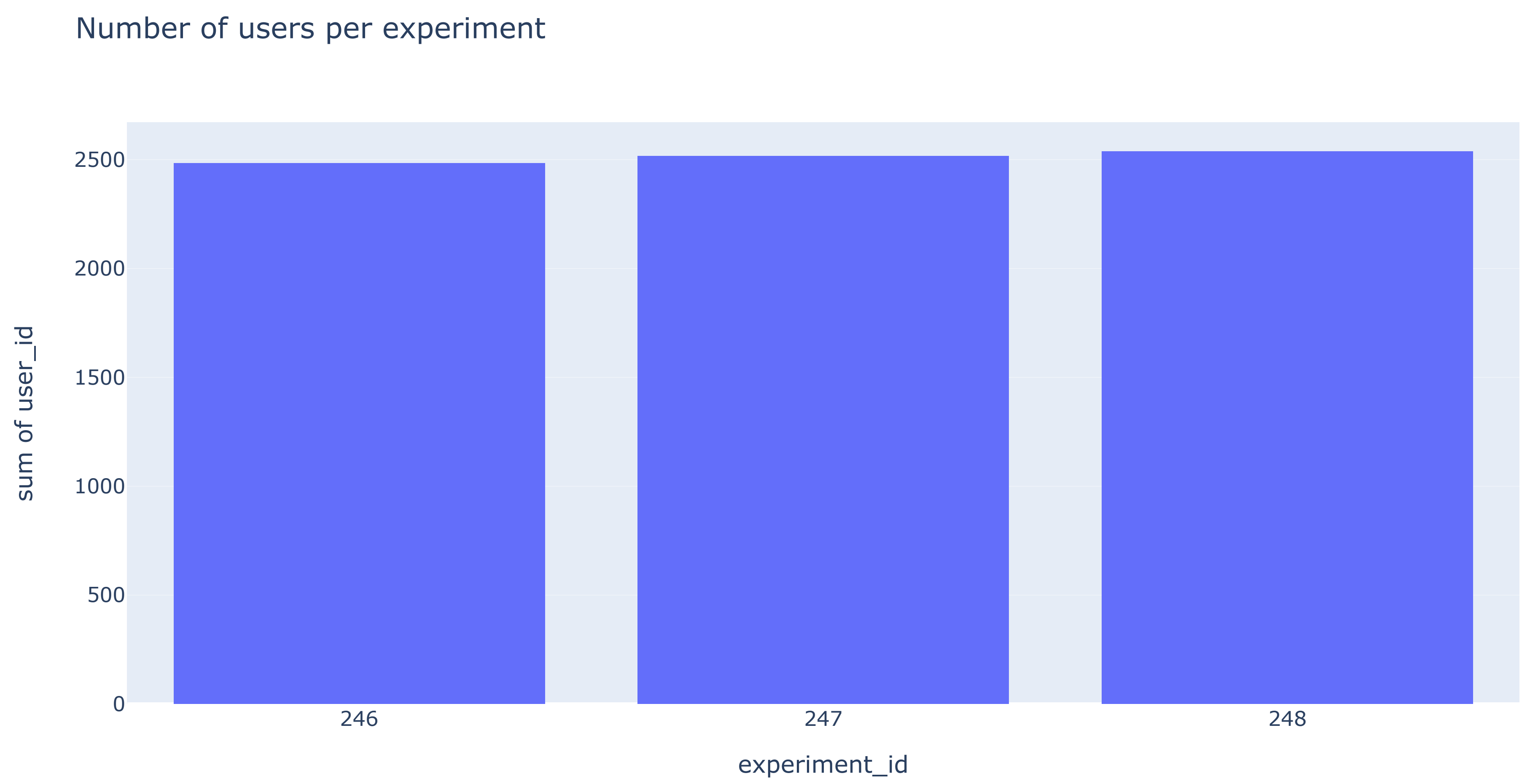
In this step we found the leader event: MainScreenAppear. The biggest gap happends between MainScreenAppear and OffersScreenAppear. interesting is that the number of users goes down on the PaymentScreenSuccessful section, but the amount of events as we saw above is still low. That mean that there areusers who make a lot of purchases. Also see that there is the highest amount of users with 1 action and with 4 actions. 3027 - 40.16% of all users performed all the 5 actions except for Tutorial and 469 including it. We decided to establish the order:'MainScreenAppear', 'OffersScreenAppear', 'CartScreenAppear', 'PaymentScreenSuccessful' and found that is was followed by almost a half or our visitors - 45.5%. That's good We really need to check why half people find nothing interesting on the MainScreenAppear page and just go away. Maybe we can drug their attention with customized products they would like, maybe show them good discounts, or maybe make them interact by other way: possibly put a little 2-d game where a user who get's to the end, can choose discount for any types of products. This can drag people's attnetion.

6 Step 5. Study the results of the experiment

6.0.1 Amount of users in each group

In [166]:
```
users_per_experiment = pd.DataFrame(filtered_logs.groupby('experiment_id')['user_id'].nunique())
```

In [167]:
```
fig = px.histogram(users_per_experiment, x=users_per_experiment.index, y = 'user_id', nbins=3,
                   title='Number of users per experiment')
fig.show()
```



It is still almost equal

6.0.2 Statistically significant difference

Let's see how many users we have per evety eent per every group

In [168]:
```
filtered_logs.pivot_table(values='user_id', index='event_name', columns='experiment_id',aggfunc=lambda x: x.nun

pivot.sort_values(by='246', ascending=False)

```

Out[168]:

experiment_id event_name	246	247	248
MainScreenAppear	2450	2479	2494
OffersScreenAppear	1542	1524	1531
CartScreenAppear	1266	1239	1231
PaymentScreenSuccessful	1200	1158	1182
Tutorial	278	284	281

6.0.2.1 Statistically significant difference between samples 246 and 247.

I have a H0 hypothesis that there is no statistically significant difference between 246 and 247 groups. Aternative hypothesis is that there is one.

In [169]:
```
ical_difference(group1, group2, alpha):
 alpha
 =filtered_logs[filtered_logs.experiment_id==group1].user_id.nunique()
 =filtered_logs[filtered_logs.experiment_id==group2].user_id.nunique()
nt in list(filtered_logs.event_name.unique()):
cess1 = pivot.loc[event, group1]
cess2 = pivot.loc[event, group2]
= success1/trials1
= success2/trials2
ference = p1 - p2
ombined = (success1 + success2) / (trials1 + trials2)
alue = difference / mth.sqrt(p_combined * (1 - p_combined) * (1/trials1 + 1/trials2))
tr = st.norm(0, 1)
alue = (1 - distr.cdf(abs(z_value))) * 2
nt(f'Succes for {group1} is {success1}, for event: {event} out of {trials1} trials\n')
nt(f'Succes for {group2} is {success2}, for event: {event} out of {trials2} trials\n')

nt(f'p-value for {event}: ', p_value)
(p_value < alpha):
 print(f"Rejecting the null hypothesis for {group1} and {group2} on event {event}: there is a significant differ
e
 print(f"Failed to reject the null hypothesis for {group1} and {group2} on event {event}: there is no reason to
```

In [170]:
```
alpha = 0.05

statistical_difference('246', '247', alpha)
```

```
Succes for 246 is 2450, for event: MainScreenAppear out of 2484 trials

Succes for 247 is 2479, for event: MainScreenAppear out of 2517 trials

p-value for MainScreenAppear:  0.6756217702005545
Failed to reject the null hypothesis for 246 and 247 on event MainScreenAppear: there is no reason to conside
r the proportions different


Succes for 246 is 1542, for event: OffersScreenAppear out of 2484 trials

Succes for 247 is 1524, for event: OffersScreenAppear out of 2517 trials

p-value for OffersScreenAppear:  0.26698769175859516
Failed to reject the null hypothesis for 246 and 247 on event OffersScreenAppear: there is no reason to consi
der the proportions different


Succes for 246 is 1200, for event: PaymentScreenSuccessful out of 2484 trials

Succes for 247 is 1158, for event: PaymentScreenSuccessful out of 2517 trials

p-value for PaymentScreenSuccessful:  0.10298394982948822
Failed to reject the null hypothesis for 246 and 247 on event PaymentScreenSuccessful: there is no reason to
consider the proportions different


Succes for 246 is 1266, for event: CartScreenAppear out of 2484 trials

Succes for 247 is 1239, for event: CartScreenAppear out of 2517 trials

p-value for CartScreenAppear:  0.2182812140633792
Failed to reject the null hypothesis for 246 and 247 on event CartScreenAppear: there is no reason to conside
r the proportions different


Succes for 246 is 278, for event: Tutorial out of 2484 trials

Succes for 247 is 284, for event: Tutorial out of 2517 trials

p-value for Tutorial:  0.9182790262812368
Failed to reject the null hypothesis for 246 and 247 on event Tutorial: there is no reason to consider the pr
oportions different
```

6.0.2.2 Coclusion

We found no signinficant difference between groups 246 and 247

6.0.2.3 Statistically significant difference between all samples

I have a H0 hypothesis that there is no statistically significant difference between groups that I check. Aternative hypothesis is that there is one.

In [171]:
```
for i in list(itertools.combinations(filtered_logs.experiment_id.unique(),2)):
    print(f'{i} test: \n')
    statistical_difference(i[0], i[1], alpha)
    print('*'*90)
```

```
consider the proportions different


Succes for 247 is 1239, for event: CartScreenAppear out of 2517 trials

Succes for 248 is 1231, for event: CartScreenAppear out of 2537 trials

p-value for CartScreenAppear:  0.6169517476996997
Failed to reject the null hypothesis for 247 and 248 on event CartScreenAppear: there is no reason to conside
r the proportions different


Succes for 247 is 284, for event: Tutorial out of 2517 trials

Succes for 248 is 281, for event: Tutorial out of 2537 trials

p-value for Tutorial:  0.8151967015119994
Failed to reject the null hypothesis for 247 and 248 on event Tutorial: there is no reason to consider the pr
oportions different
```

6.0.2.4 Coclusion

And we found no statistical difference what so ever

6.0.3 the most popular event

In [172]:
```
funnel[funnel.user_id == funnel.user_id.max()].event_name
```

Out[172]:
```
0    MainScreenAppear
Name: event_name, dtype: category
Categories (5, object): ['CartScreenAppear', 'MainScreenAppear', 'OffersScreenAppear', 'PaymentScreenSuccessf
ul', 'Tutorial']
```

6.0.4 The share of the number of users who performed the most popular action

In [173]:
```
def find_share(group1, group2, event):
    print(
        f'share of users from {group1}: {round(pivot.loc[event, group1]/pivot.loc[event, :].sum() * 100,2)}
    print(
        f'share of users from {group2}: {round(pivot.loc[event, group2]/pivot.loc[event, :].sum() * 100,2)}
    print()
```

In [174]:
```
for i in list(itertools.combinations(filtered_logs.experiment_id.unique(),2)):
    print(f'Share of MainScreenAppear event for groups {i}')
    find_share(i[0],i[1], 'MainScreenAppear')
```

```
Share of MainScreenAppear event for groups ('247', '248')
share of users from 247: 33.4%.
share of users from 248: 33.6%.

Share of MainScreenAppear event for groups ('247', '246')
share of users from 247: 33.4%.
share of users from 246: 33.01%.

Share of MainScreenAppear event for groups ('248', '246')
share of users from 248: 33.6%.
share of users from 246: 33.01%.
```

6.0.5 Conclusion

The number of users looks properly distributed and have equal proportions

6.0.6 The share of the number of users on every stage

In [175]:
```
for i in list(itertools.combinations(filtered_logs.experiment_id.unique(),2)):
    for event in list(filtered_logs.event_name.unique()):
        print(f'Share of {event} event for groups {i}')
        find_share(i[0],i[1], event)
```

```
Share of MainScreenAppear event for groups ('247', '248')
share of users from 247: 33.4%.
share of users from 248: 33.6%.

Share of OffersScreenAppear event for groups ('247', '248')
share of users from 247: 33.15%.
share of users from 248: 33.3%.

Share of PaymentScreenSuccessful event for groups ('247', '248')
share of users from 247: 32.71%.
share of users from 248: 33.39%.

Share of CartScreenAppear event for groups ('247', '248')
share of users from 247: 33.16%.
share of users from 248: 32.95%.

Share of Tutorial event for groups ('247', '248')
share of users from 247: 33.69%.
share of users from 248: 33.33%.

Share of MainScreenAppear event for groups ('247', '246')
share of users from 247: 33.4%.
share of users from 246: 33.01%.

Share of OffersScreenAppear event for groups ('247', '246')
share of users from 247: 33.15%.
share of users from 246: 33.54%.

Share of PaymentScreenSuccessful event for groups ('247', '246')
share of users from 247: 32.71%.
share of users from 246: 33.9%.

Share of CartScreenAppear event for groups ('247', '246')
share of users from 247: 33.16%.
share of users from 246: 33.89%.

Share of Tutorial event for groups ('247', '246')
share of users from 247: 33.69%.
share of users from 246: 32.98%.

Share of MainScreenAppear event for groups ('248', '246')
share of users from 248: 33.6%.
share of users from 246: 33.01%.

Share of OffersScreenAppear event for groups ('248', '246')
share of users from 248: 33.3%.
share of users from 246: 33.54%.

Share of PaymentScreenSuccessful event for groups ('248', '246')
share of users from 248: 33.39%.
share of users from 246: 33.9%.

Share of CartScreenAppear event for groups ('248', '246')
share of users from 248: 32.95%.
share of users from 246: 33.89%.

Share of Tutorial event for groups ('248', '246')
share of users from 248: 33.33%.
share of users from 246: 32.98%.
```

6.0.7 Conclusion

And this is true for every stage and every group. No significat difference.

6.0.8 Significance level of the statistical hypotheses

6.0.9 Calculate how many statistical hypothesis tests you carried out.

In [176]:
```
number_of_tests = filtered_logs.event_name.nunique() * filtered_logs.experiment_id.nunique()
```

Since I really have a lot of tests I need to be sure and needd to adjust the value of alpha for it. I checked the Bonferroni method and got to the same conclusions (see below), but since we should really care about all the error types I also used the Šidák approach, because it offers higher power and got some new answer

6.0.10 Bonferroni

In [177]:
```
bonferroni = alpha / number_of_tests

for i in list(itertools.combinations(filtered_logs.experiment_id.unique(),2)):
    print(f'{i} test: \n')
    statistical_difference(i[0], i[1], bonferroni)
    print('*'*90)
```

```
('247', '248') test:

Succes for 247 is 2479, for event: MainScreenAppear out of 2517 trials

Succes for 248 is 2494, for event: MainScreenAppear out of 2537 trials

p-value for MainScreenAppear:  0.6001661582453706
Failed to reject the null hypothesis for 247 and 248 on event MainScreenAppear: there is no reason to conside
r the proportions different


Succes for 247 is 1524, for event: OffersScreenAppear out of 2517 trials

Succes for 248 is 1531, for event: OffersScreenAppear out of 2537 trials

p-value for OffersScreenAppear:  0.8835956656016957
Failed to reject the null hypothesis for 247 and 248 on event OffersScreenAppear: there is no reason to consi
der the proportions different
```

Nothing new here

6.0.11 Šidák

In [178]:
```
Šidák = 1 - pow((1-alpha),number_of_tests)
```

In [179]:
```
for i in list(itertools.combinations(filtered_logs.experiment_id.unique(),2)):
    print(f'{i} test: \n')
    statistical_difference(i[0], i[1], Šidák)
    print('*'*90)
```

```
Succes for 246 is 1266, for event: CartScreenAppear out of 2484 trials

p-value for CartScreenAppear:  0.2182812140633792
Rejecting the null hypothesis for 247 and 246 on event CartScreenAppear: there is a significant difference be
tween the proportions


Succes for 247 is 284, for event: Tutorial out of 2517 trials

Succes for 246 is 278, for event: Tutorial out of 2484 trials

p-value for Tutorial:  0.9182790262812368
Failed to reject the null hypothesis for 247 and 246 on event Tutorial: there is no reason to consider the pr
oportions different


******************************************************************************************
('248', '246') test:
```

We found a statistically significant difference between 247 and 246 and between 248 and 246 on the events: OffersScreenAppear, PaymentScreenSuccessful, CartScreenAppear. But that's weird because 246 and 247 are the control groups, and 248 is the test group

7 General Conclusion

This dataset have 5 unique events and 243713 events in general with 7551 user having nearly 32 events per one. Even though the data containes information on 2 weeks period, we can consider only the second week worth - since 01-08, because prior to it the data was incomplete. I found that the amount of events is naturally higher at day and naturally lower at night. After getting rid of the fist week data we lost only lost 0.82% of data, that included visits of 1319 users.
Thean I found the leader event: MainScreenAppear and establish customer journey: 'MainScreenAppear', 'OffersScreenAppear', 'CartScreenAppear', 'PaymentScreenSuccessful'. It was followed by almost a half or our visitors - 45.5%. Funnel showed me that the biggest procentage of user loss happens between MainScreenAppear and OffersScreenAppear, and we should pay more attention to it and to find te reasons why. Also interesting is that the number of users goes down on the PaymentScreenSuccessful section, but the amount of events as we saw above is still low. That mean that there are users who make a lot of purchases, our loyal customers. Never the less we have its 36.04% of all users who performed just one action, but 93.78%, who performed it all. We really need to check why half of users finds nothing interesting on the MainScreenAppear page and just goes away. Maybe we can drug their attention with customized products they would like, maybe show them good discounts, or maybe make them interact by other way: possibly put a little 2-d game where a user who get's to the end, can choose discount for any types of products.

I found no difference in proportions for all the groups. Also using both Šidák and Bonferroni approach I found no statistically significant diggerence between the test group and control groups. So we can say that the test failed to bring us more conversion.