progect description

This project is dedicated to investigation of user behavior for a company's app (Let's say this is a startup that sells food products): study the sales funnel, look at the results of an A/A/B test, formulate statistical hypotheses.

Table of Contents

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 [290]:
```
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 [291]:
```
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 [292]:
```
ry:
   logs_exp = pd.read_csv('/datasets/logs_exp_us.csv', sep='\t', dtype={'EventName': 'category',
                                                                         'ExpId': 'category'})  # practic
xcept:
   try:
       logs_exp = pd.read_csv('./datasets/logs_exp_us.csv', sep='\t', dtype={'EventName': 'category',
                                                                              'ExpId': 'category'})  # lc
   except:
       try:
           logs_exp = pd.read_csv('https://code.s3.yandex.net//datasets/logs_exp_us.csv', sep='\t', dtyp

       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 [293]:
```
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
```

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

Step 2. Preprocessing the data

Renaming the columns

In [294]:
```
logs_exp.columns = ['event_name', 'user_id', 'timestamp', 'experiment_id']
```

In [295]:
```
logs_exp.user_id.nunique()
```

Out[295]:
```
7551
```

Checking for missing values and data types

In [296]:
```
logs_exp.describe(include='all')
```

Out[296]:

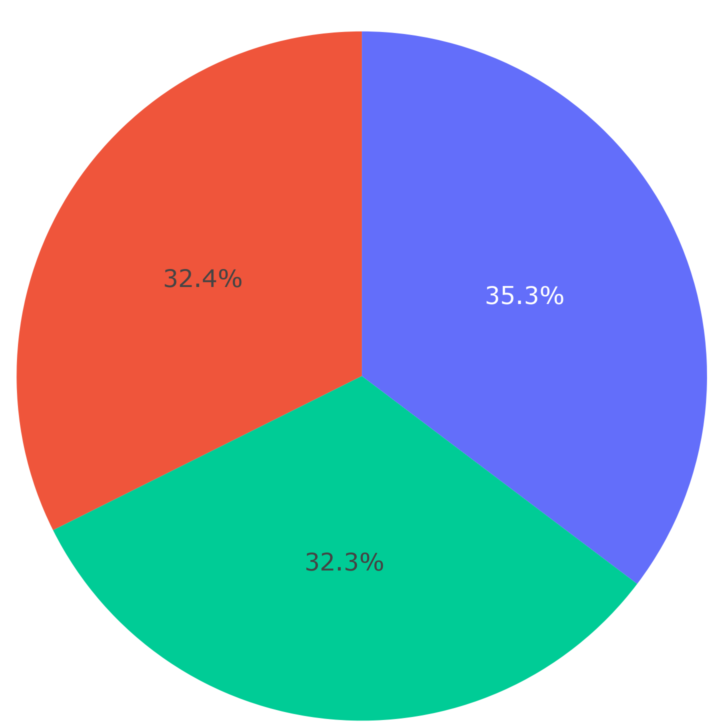
	event_name	user_id	timestamp	experiment_id
count	244126	2.441260e+05	2.441260e+05	244126
unique	5	NaN	NaN	3
top	MainScreenAppear	NaN	NaN	248
freq	119205	NaN	NaN	85747
mean	NaN	4.627568e+18	1.564914e+09	NaN
std	NaN	2.642425e+18	1.771343e+05	NaN
min	NaN	6.888747e+15	1.564030e+09	NaN
25%	NaN	2.372212e+18	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 [297]:
```
round(logs_exp.experiment_id.value_counts(normalize=True) * 100,2)
```

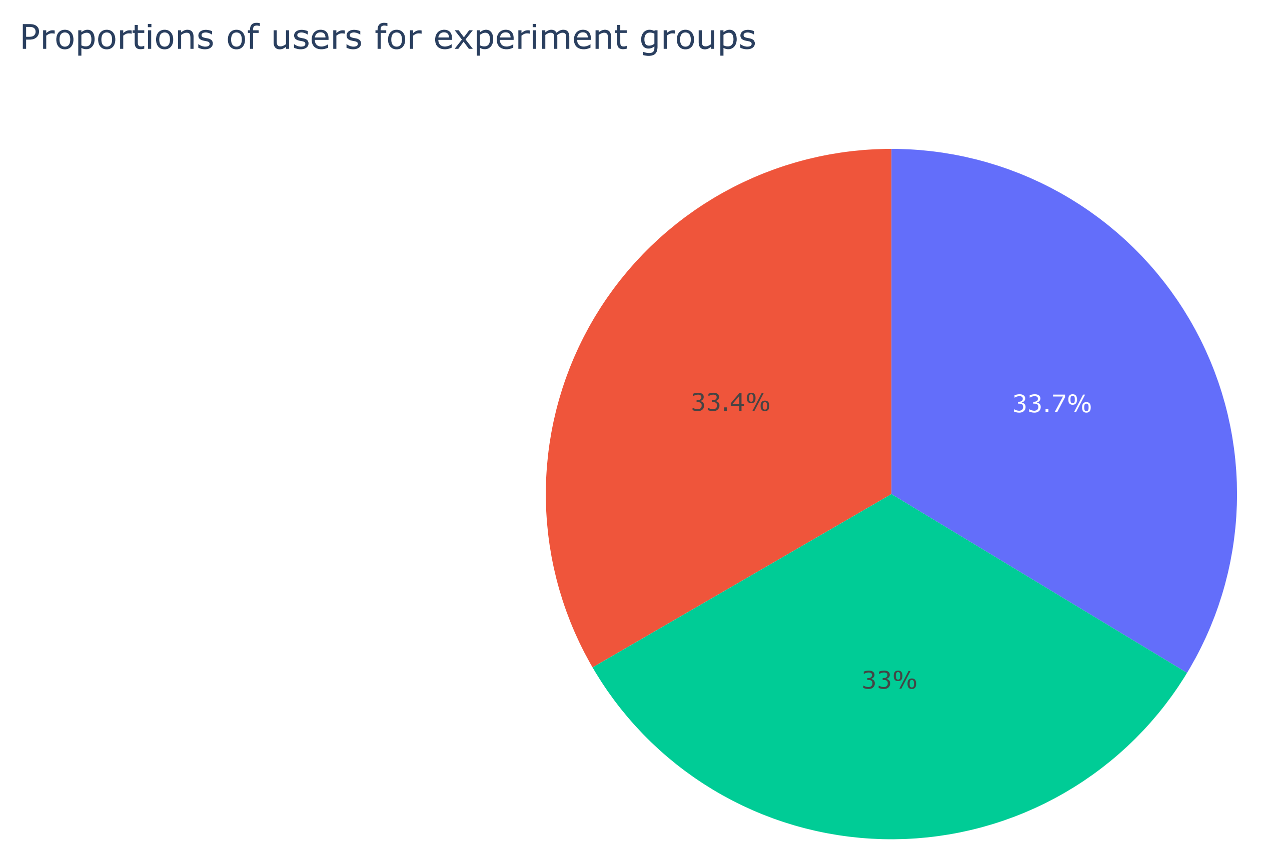
Out[297]:
```
248    35.12
246    32.89
247    31.98
Name: experiment_id, dtype: float64
```

In [298]:
```
fig = px.pie(logs_exp, values='user_id', names='experiment_id', title='Proportions of events for experim
fig.show()
```

Proportions of events for experiment groups



In [299]:
```
fig = px.pie(logs_exp.groupby('experiment_id')['user_id'].nunique().reset_index(), values='user_id', nam
fig.show()
```



Duplicates

In [300]:
```
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 [301]:
```
print(f'procentage of duplicates is {round(logs_exp.duplicated().sum() / logs_exp.shape[0] * 100,2)}%')
```

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

In [302]:
```
logs_exp = logs_exp.drop_duplicates()
```

date and time column + dates column

In [303]:
```
logs_exp['timestamp'] = logs_exp.timestamp.apply(lambda x:datetime.fromtimestamp(x))
logs_exp['date'] = logs_exp['timestamp'].astype('datetime64[D]')
```

Conclusion

We found a tiny amount of duplicated rows, it means, something is wrong with the 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 accepted naming format. The proportions for experiments look equal

Step 3. Data Discovery

How many events are in the logs?

In [304]:
```
print(f'we have {logs_exp.event_name.nunique()} unique events and {logs_exp.shape[0]} events in general
```

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

How many users are in the logs?

In [305]:
```
logs_exp.user_id.nunique()
```

Out[305]:
```
7551
```

What's the average number of events per user?

In [306]:
```
round(logs_exp.groupby('user_id')['event_name'].count().mean(),2)
```

Out[306]: 32.28

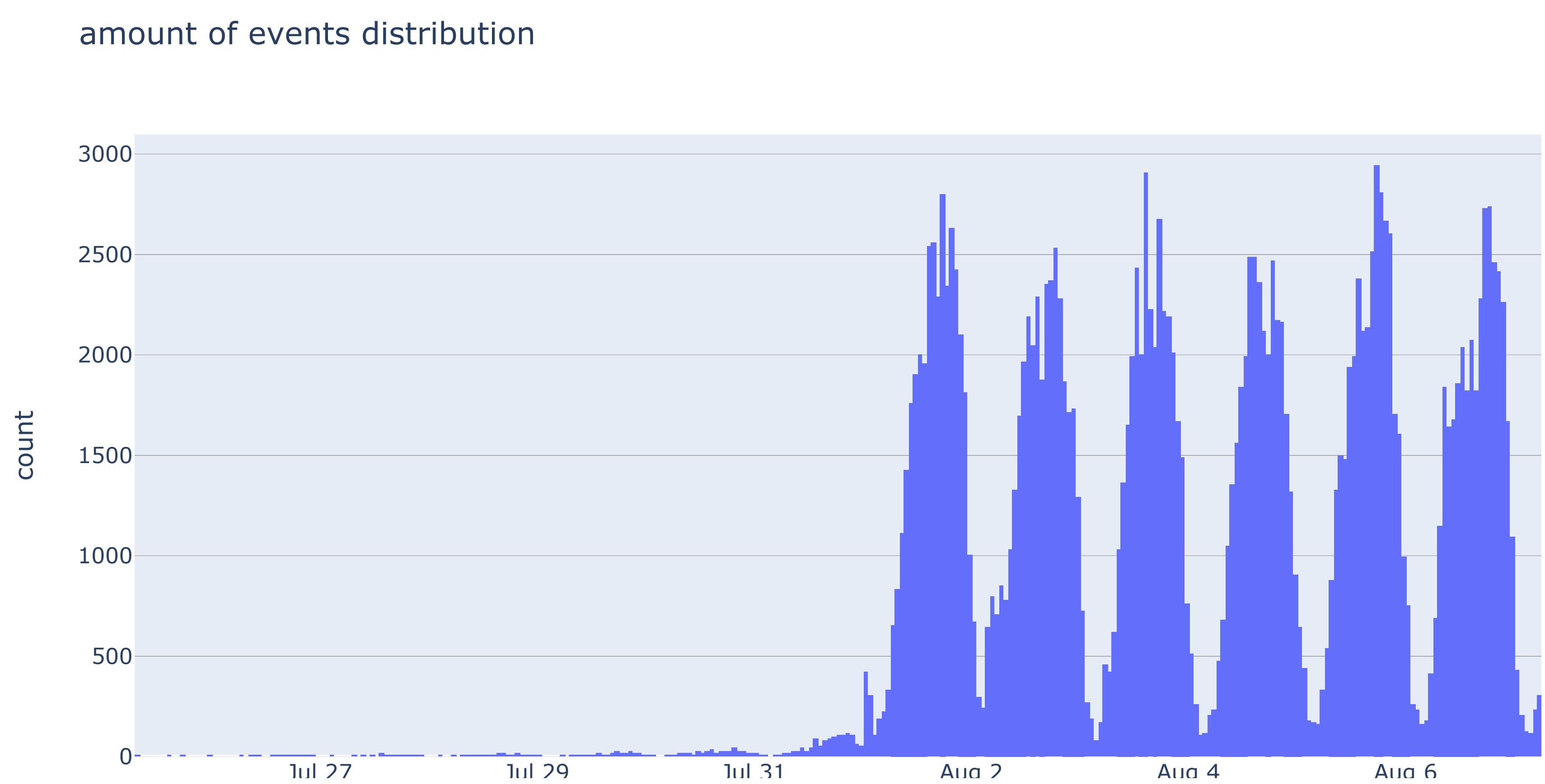
What period of time does the data cover? Find the maximum and the minimum date.

In [307]:
```
e research period is from {logs_exp.date.min()} to {logs_exp.date.max()} covering {(logs_exp.date.max()
```

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

Date and time histogram

In [308]:
```
fig = px.histogram(logs_exp, x="timestamp", title='amount of events distribution')

fig.show()
```

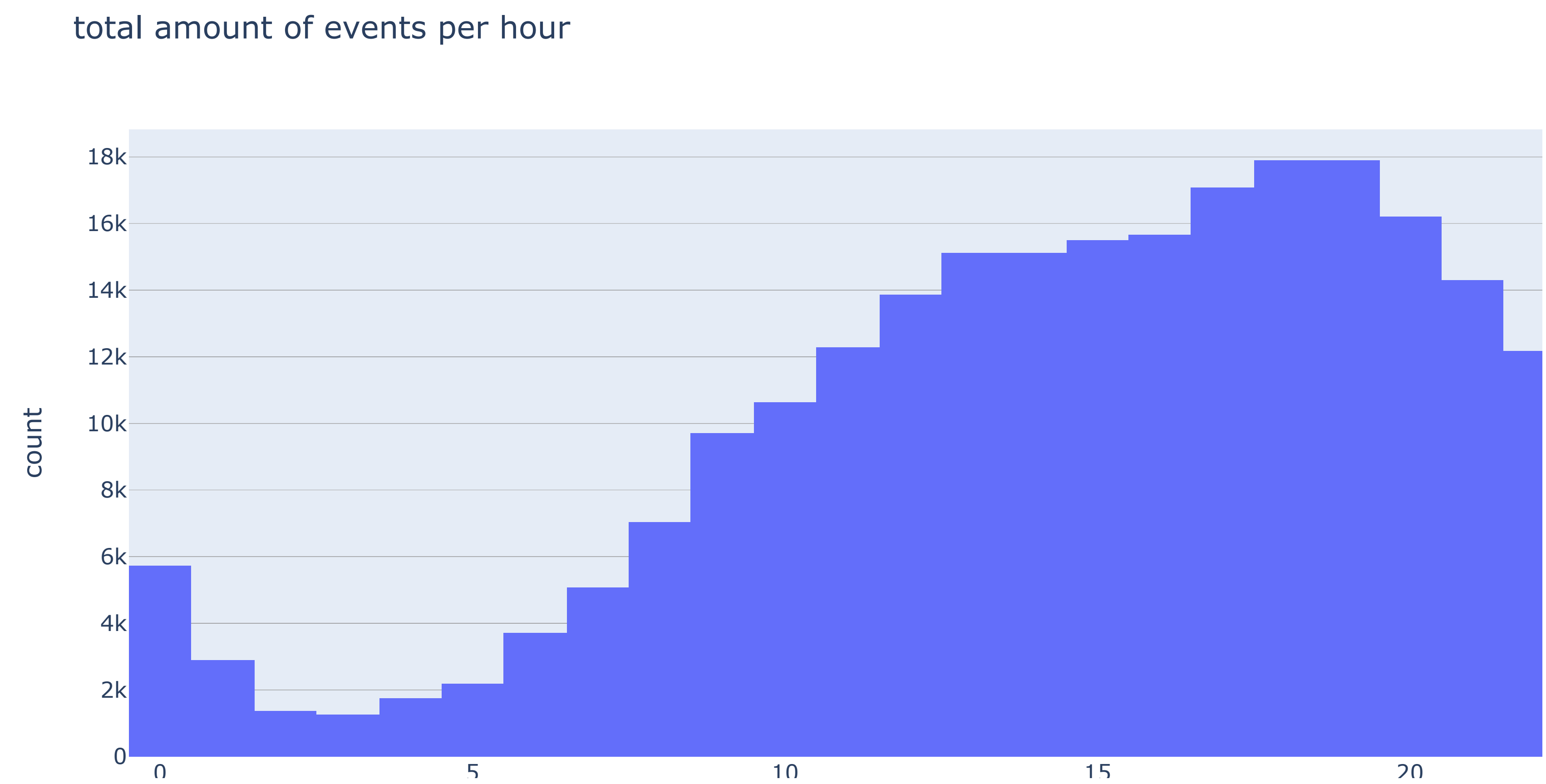


We can see really clear that August data has it's own pattern, but I want to get sure that the 1st August also belong to the pattern model. Also, I want to check the hour events distribution, because I believe that this falls happen due to night hour lower visitor flow and peaks - due to daytime visitors.

In [309]:
```
logs_exp['hour'] = logs_exp.timestamp.dt.round('H')
```

In [310]:
```
logs_exp['only_hour'] = logs_exp['hour'].dt.hour
```

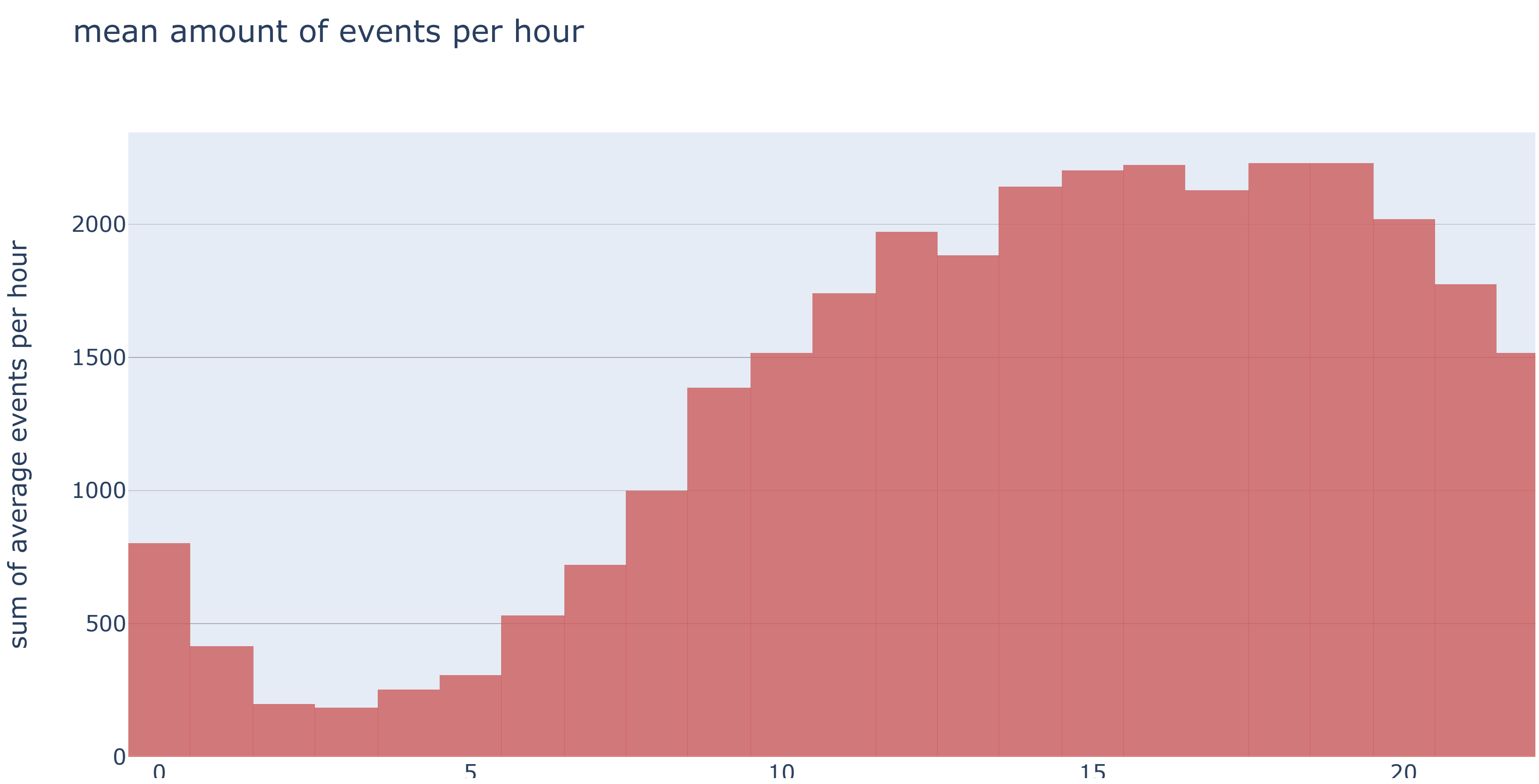
In [311]:
```
fig = px.histogram(logs_exp, x="only_hour", title='total amount of events per hour')
fig.show()
```



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

In [312]:
```
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 [313]:
```
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') # colo

fig.show()
```



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

In [314]:
```
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.head()
```

Out[314]:

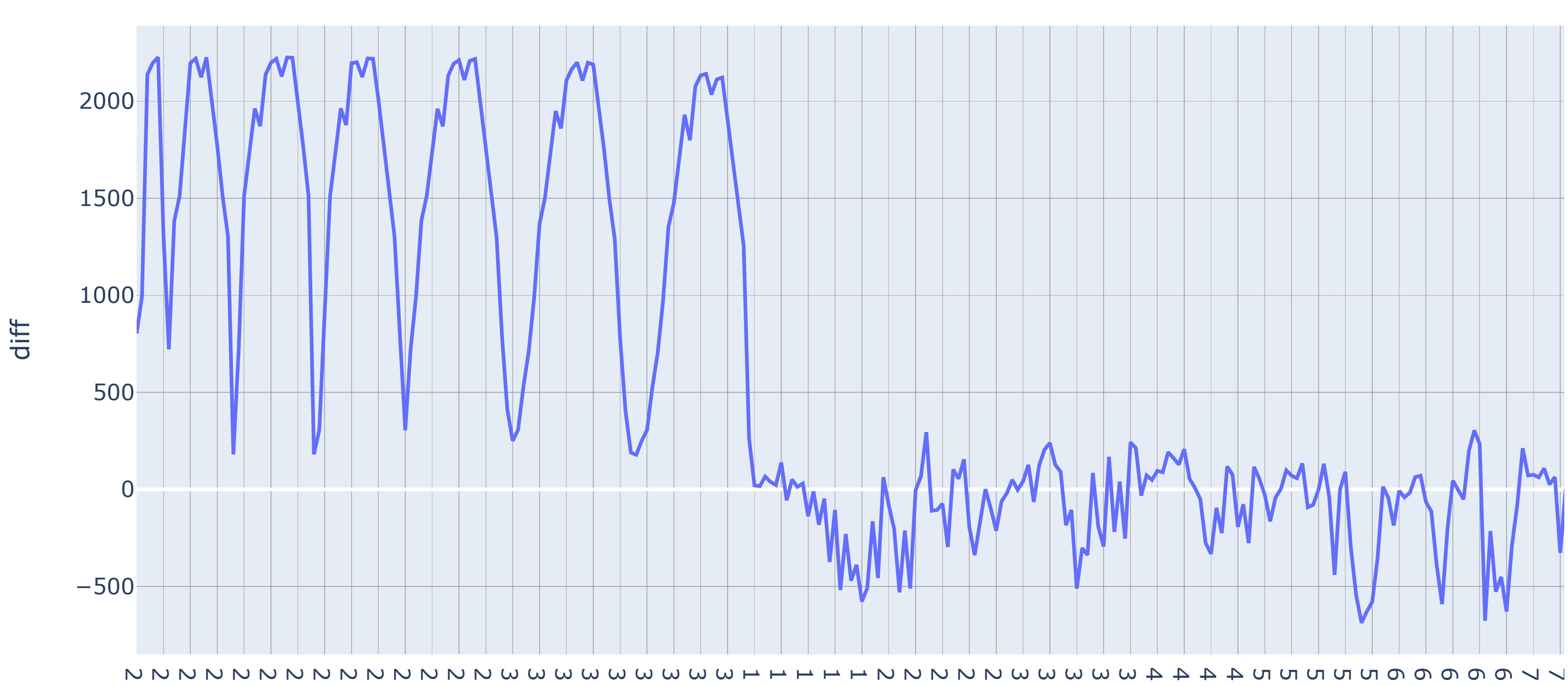
	date	hour	events_num	mean_events_num	diff
0	2019-07-25	0	1	804.285714	803.285714
1	2019-07-25	8	1	996.857143	995.857143
2	2019-07-25	14	3	2140.714286	2137.714286
3	2019-07-25	15	2	2199.000000	2197.000000
4	2019-07-25	18	1	2227.500000	2226.500000

In [315]:
```
per_day_per_hour['when'] = per_day_per_hour['date'].dt.day.astype('str') + ', ' + per_day_per_hour['hour
```

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

In [316]:
```
fig = px.line(per_day_per_hour,x='when', y="diff", title='Difference between regular event number and re
fig.show()
```

Difference between regular event number and recieved



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 received information, maybe due to technical reasons. I analyzed the data and found an average amount of events for every hour to compare with the received data. I can see on the graph that starting from 2019-08-01 data looks 'normal' and I choose this date to be the first 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.

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

In [317]:
```
= '2019-08-01'

.ogs = logs_exp[logs_exp['date'] >= good_date]

: logs_exp[logs_exp['date'] < good_date]

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

: 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 l
ost 0.82% of data
We also lost 1319 users
```

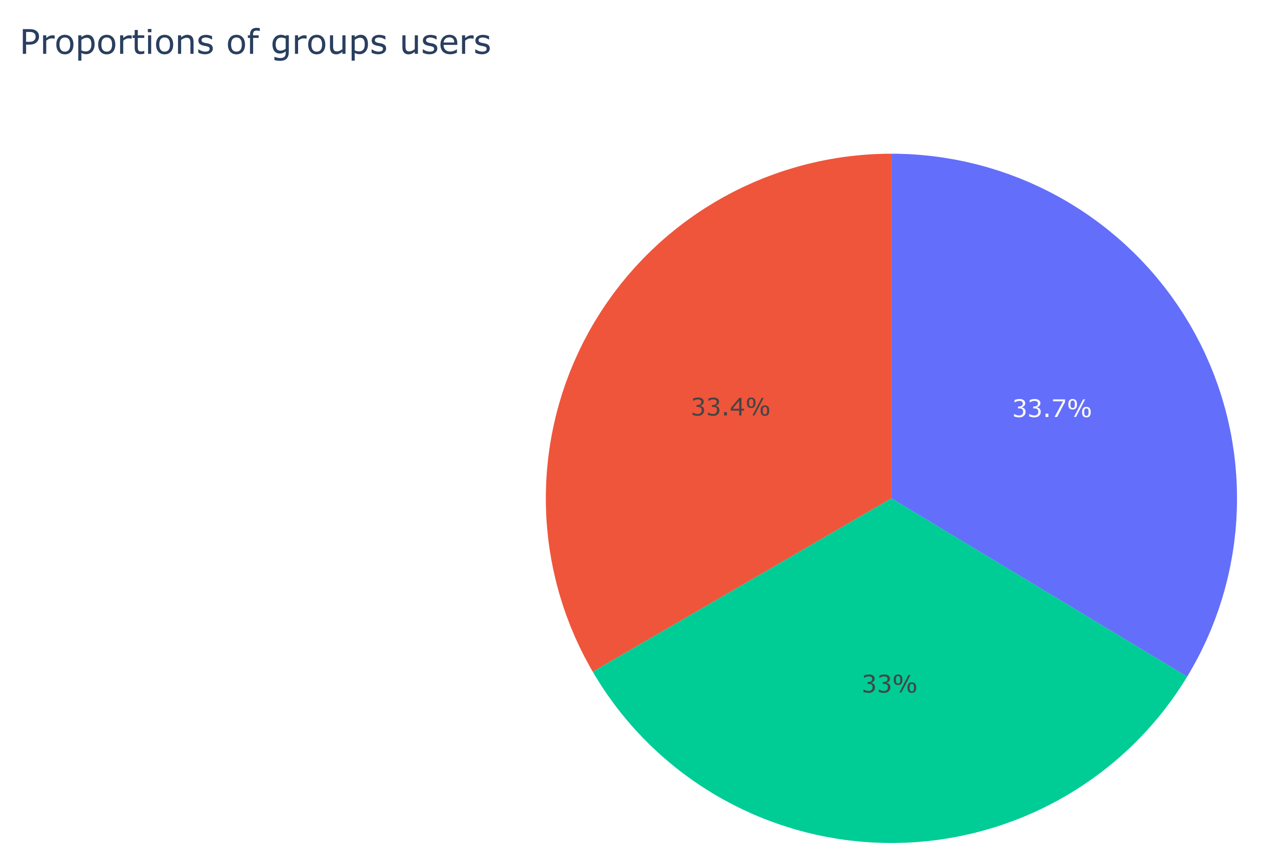
I also want to see, what are the proportions of the remained events.

In [318]:
```
bad_data.event_name.value_counts(normalize=True) * 100
```

Out[318]:
```
MainScreenAppear           60.935143
CartScreenAppear           16.339869
OffersScreenAppear         13.926596
PaymentScreenSuccessful     8.396179
Tutorial                    0.402212
Name: event_name, dtype: float64
```

Make sure you have users from all three experimental groups.

In [319]:
```
fig = px.pie(filtered_logs.groupby('experiment_id')['user_id'].nunique().reset_index(), values='user_id'
fig.show()
```



In [320]:
```
filtered_logs.groupby('experiment_id')['user_id'].nunique()
```

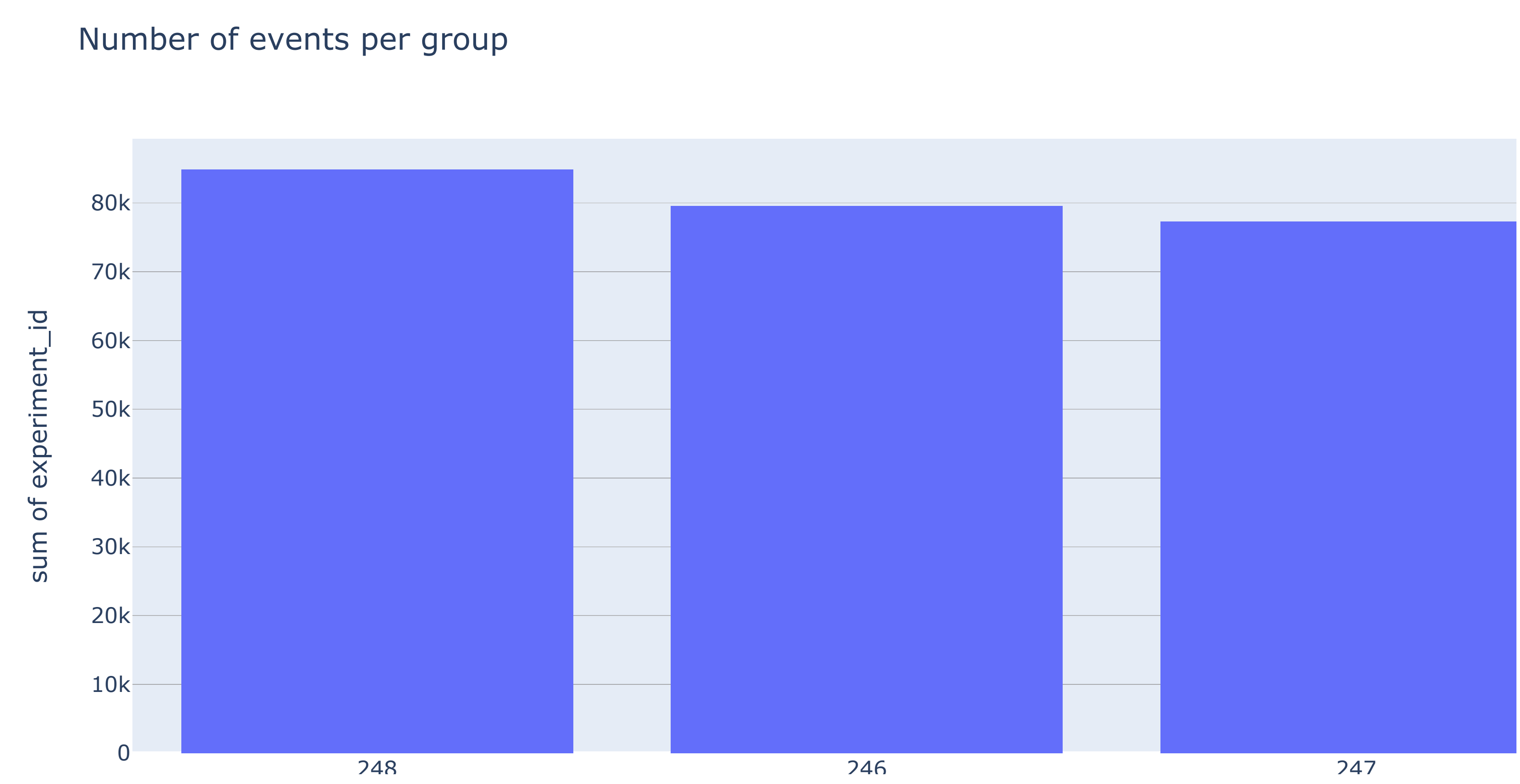
Out[320]:
```
experiment_id
246    2484
247    2517
248    2537
Name: user_id, dtype: int64
```

In [321]:
```
events_count = pd.DataFrame(filtered_logs.experiment_id.value_counts())
events_count
```

Out[321]:

	experiment_id
248	84875
246	79556
247	77293

In [322]:
```
fig = px.histogram(events_count, x=events_count.index, y = 'experiment_id',
                   title='Number of events per group')
fig.show()
```



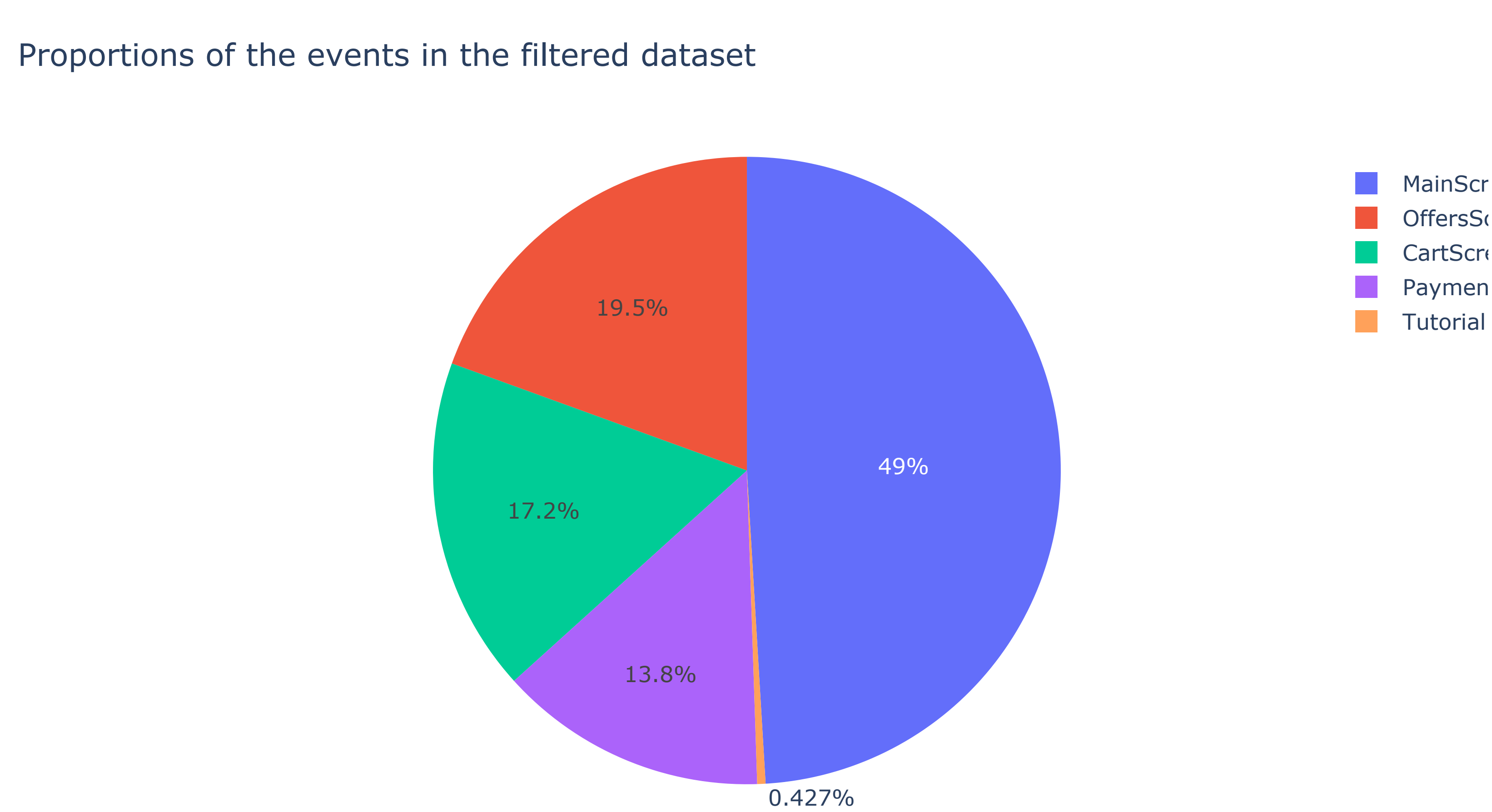
Conclusion

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

Step 4. The event funnel

Frequency of event occurrence

In [323]:
```
fig = px.pie(filtered_logs, values='user_id', names='event_name', title='Proportions of the events in th
fig.show()
```



In [324]:
```
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 [325]:
```
filtered_logs.groupby('user_id')['event_name'].apply(lambda x: x.mode()).value_counts()
```

Out[325]:
```
MainScreenAppear           6035
OffersScreenAppear         1176
CartScreenAppear            741
PaymentScreenSuccessful     173
Tutorial                     29
Name: event_name, dtype: int64
```

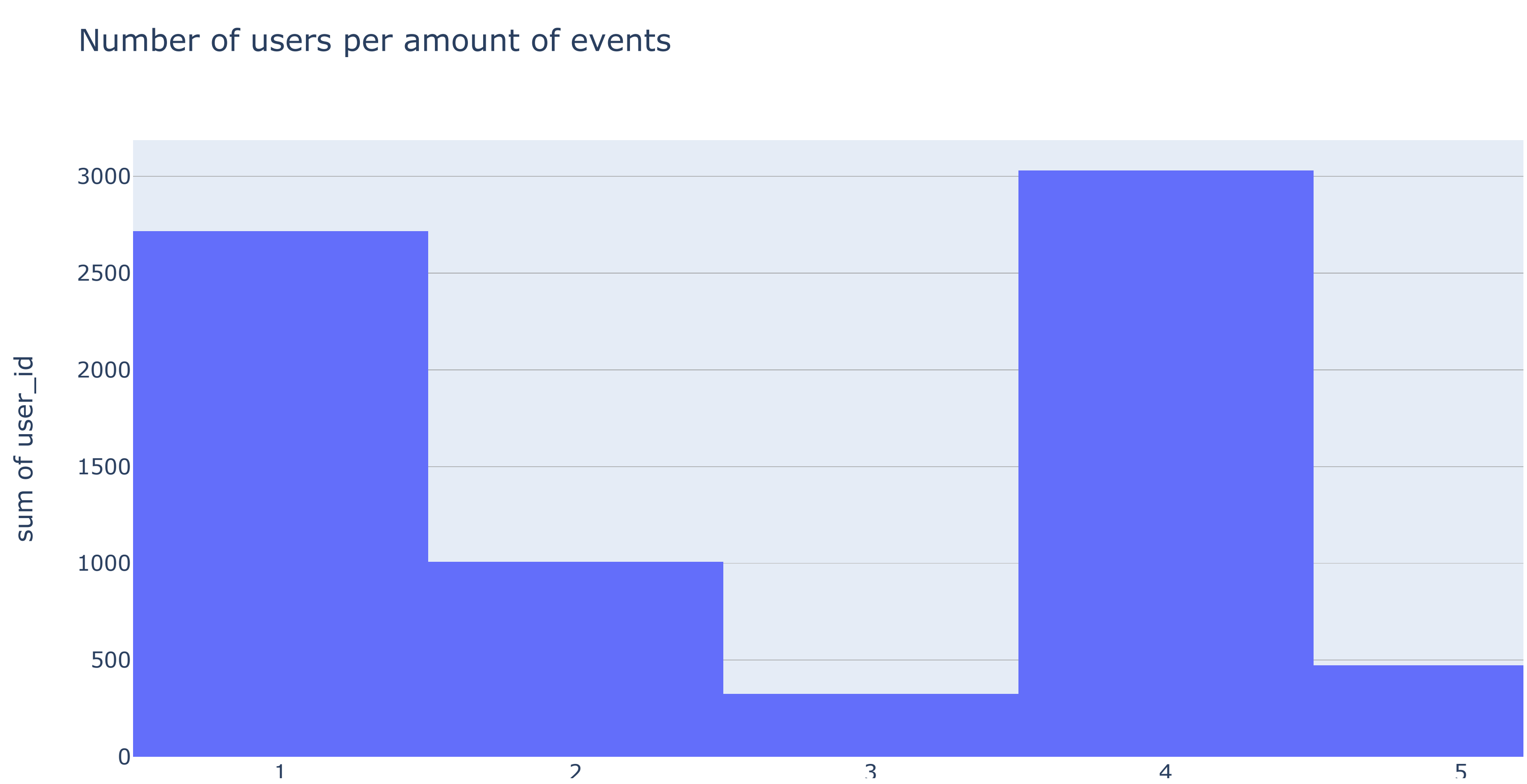
Conclusion

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

Users who performed each action

In [326]:
```
ıount_of_events = pd.DataFrame(filtered_logs.groupby('user_id')['event_name'].nunique().reset_index().grc
```

In [327]:
```
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 [328]:
```
user_events = filtered_logs.groupby('user_id')['event_name'].nunique().reset_index(drop=False)

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

In [329]:
```
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'])
```

Conclusion

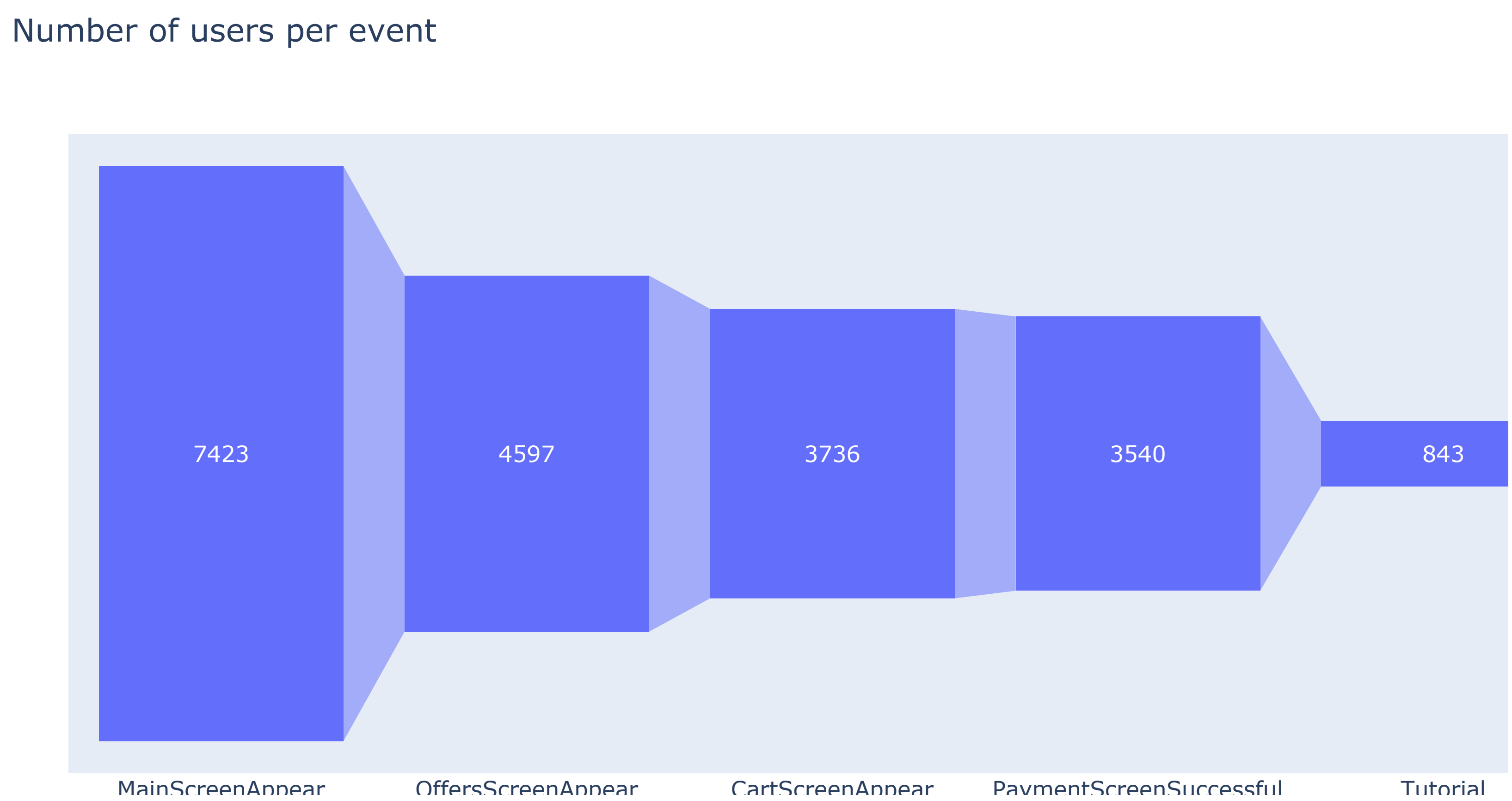
In [330]:
```
)0,2)}% of all users -  who performed all the 5 actions except for Tutorial and {len(users_5_actions)} ir
```

```
We can see that there is the highest amount of users who did only 1 action - presumably MainScreenAppe
ar and of those who did 4 actions - all the necessary ones, but not the tutorial. So we have 3027 - 4
0.16% of all users -  who performed all the 5 actions except for Tutorial and 469 including it.
```

Sort the events by the number of users

In [331]:
```
number_of_users = pd.DataFrame(filtered_logs.groupby('event_name')['user_id'].nunique().sort_values(asce
```

In [332]:
```
fig = px.funnel(number_of_users, x=number_of_users.index, y = 'user_id',  title='Number of users per eve
fig.show()
```



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 are users who make a lot of purchases

Proportion of users who performed the action at least once.

performed any action just once

In [333]:
```
users_1_action = list(user_events[user_events['number_of_events'] == 1]['user_id'])
```

In [334]:
```
print(f'We have {len(users_1_action)} users performed just one action, the MainScreenAppear presumably,
```

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

In [335]:
```
print(f' But those who performed the action at least one and more actions are {round((filtered_logs.user
```

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

Order of actions

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

In [336]:
```
order = ['MainScreenAppear', 'OffersScreenAppear', 'CartScreenAppear', 'PaymentScreenSuccessful']
```

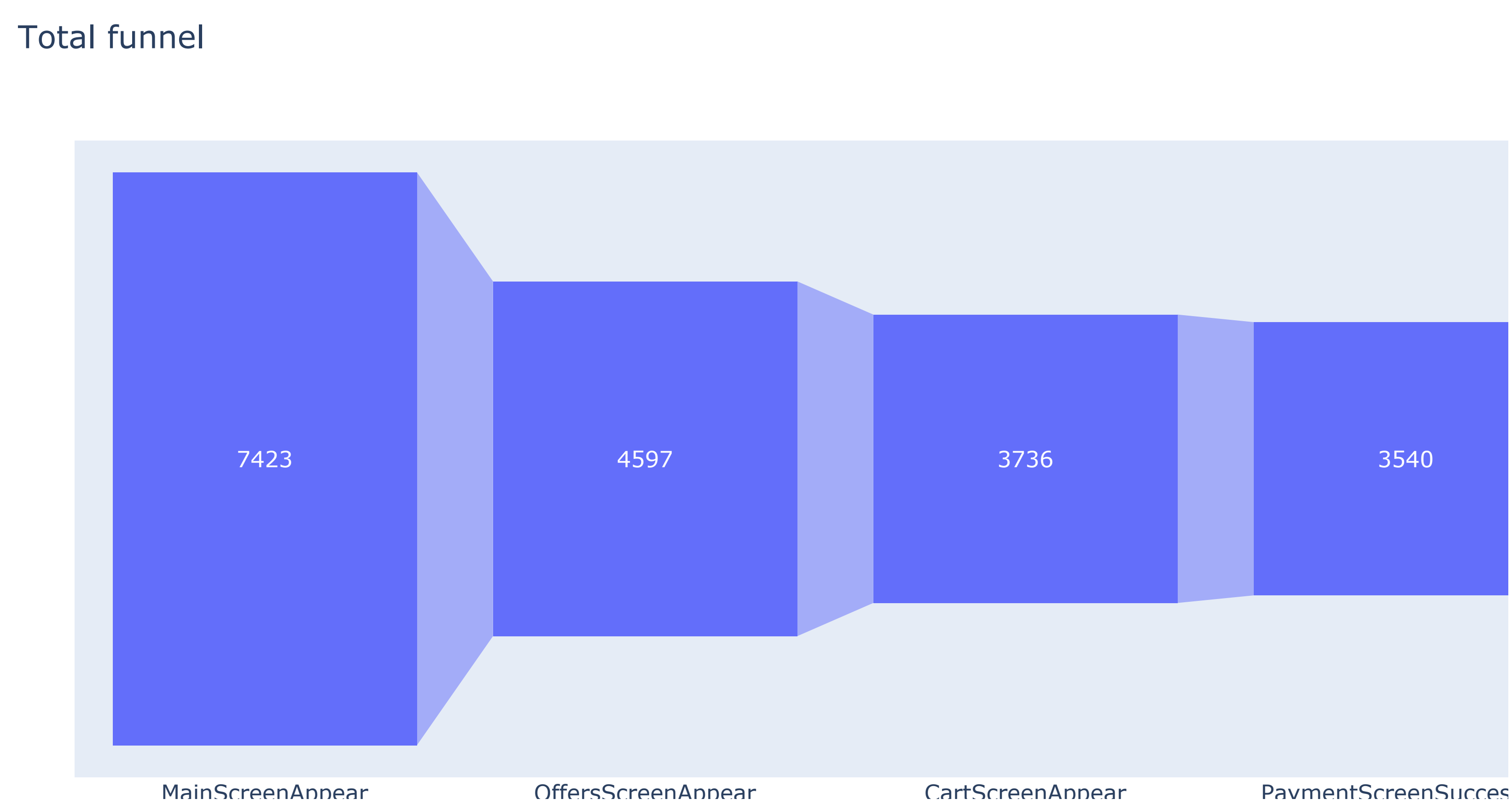
Event funnel

In [337]:
```
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[337]:

	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 [338]:
```
fig = px.funnel(funnel, x ='event_name', y = 'user_id', title='Total funnel')
fig.show()
```

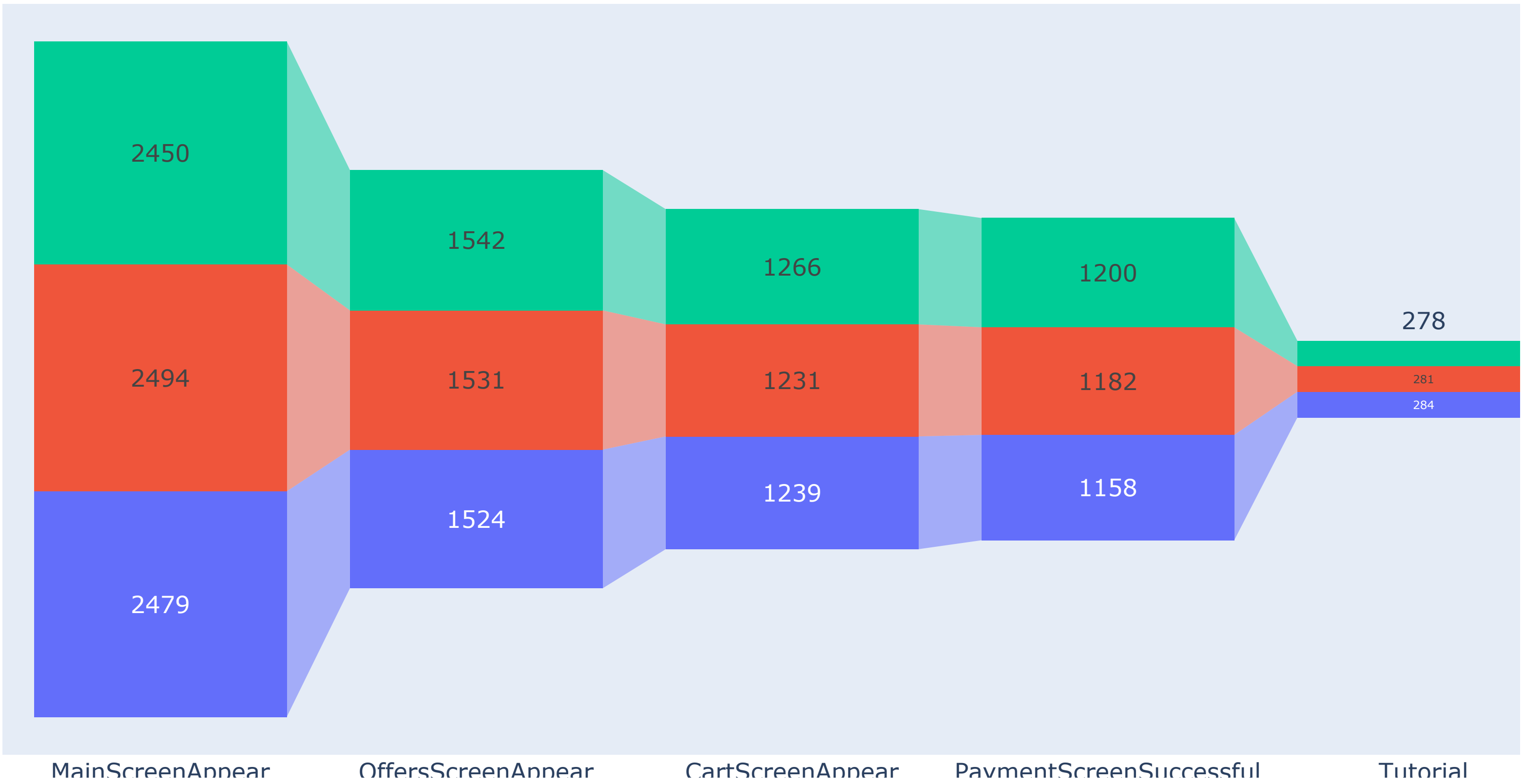


Conclusion

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

In [339]:
```
total_funnel = pd.concat(funnel_list, axis=0)
```

In [340]:
```
fig = px.funnel(total_funnel, x ='event_name', y = 'user_id', color='group')
fig.show()
```



Stage with highest lose rate

In [341]:
```
funnel[funnel.pct == funnel.pct.min()].event_name
```

Out[341]:
```
1    OffersScreenAppear
Name: event_name, dtype: category
Categories (5, object): ['CartScreenAppear', 'MainScreenAppear', 'OffersScreenAppear', 'PaymentScreenS
uccessful', 'Tutorial']
```

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

In [342]:
```
every_event_per_user = filtered_logs.pivot_table(index='user_id', columns='event_name', values='timestar

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

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

In [343]:
```
print(f'We have {every_event_per_user.shape[0]} who had this whole journey. it is {round(every_event_per
```

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

Conclusion

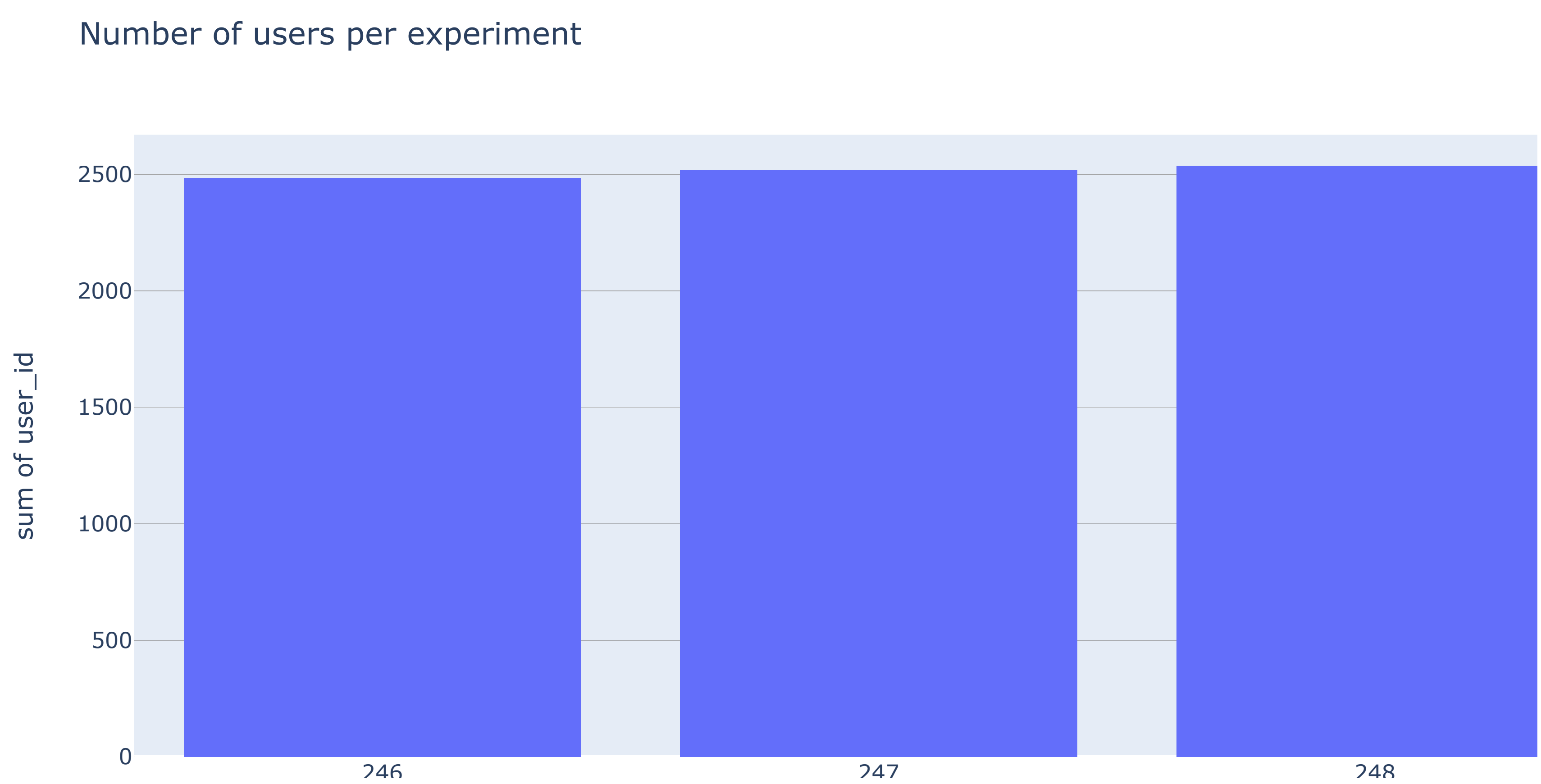
In this step we found the leader event: MainScreenAppear. The biggest gap happens between MainScreenAppear and OffersScreenAppear. interesting is that the number of users goes down on the PaymentScreenSuccessful section, but the amount of events as we saw above is still low. That mean that there are users 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 been followed by almost a half or our visitors - 45.5%. That's good.

Step 5. Study the results of the experiment

Amount of users in each group

In [344]:
```
users_per_experiment = pd.DataFrame(filtered_logs.groupby('experiment_id')['user_id'].nunique())
```

In [345]:
```
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

Statistically significant difference

In [346]:
```
proportions = filtered_logs.groupby(['experiment_id', 'event_name'])['user_id'].nunique().reset_index()
```

Statistically significant difference between group samples

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

In [347]:
```
def statistical_difference(group1, group2, alpha):
    alpha = alpha
    trials1 = filtered_logs[filtered_logs.experiment_id==group1].user_id.nunique()
    trials2 = filtered_logs[filtered_logs.experiment_id==group2].user_id.nunique()
    for event in list(filtered_logs.event_name.unique()):
        success1 = int(proportions[(proportions.event_name==event) & (proportions.experiment_id == group
        success2 = int(proportions[(proportions.event_name==event) & (proportions.experiment_id == group
        p1 = success1/trials1
        p2 = success2/trials2
        difference = p1 - p2
        p_combined = (success1 + success2) / (trials1 + trials2)
        z_value = difference / mth.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(f'Succes for {group1} is {success1}, for event: {event} out of {trials1} trials\n')
        print(f'Succes for {group2} is {success2}, for event: {event} out of {trials2} trials\n')

        print(f'p-value for {event}: ', p_value)
        if (p_value < alpha):
            print(f"Rejecting the null hypothesis for {group1} and {group2} on event {event}: there is a
        else:
            print(f"Failed to reject the null hypothesis for {group1} and {group2} on event {event}: the
```

In [348]:
```
alpha = 0.05
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)
```

```
('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
consider 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 t
o consider the proportions different
```

Coclusion

And we found no statistical difference what so ever

the most popular event

In [349]:
```
print('the most popular event is', funnel[funnel.user_id == funnel.user_id.max()].event_name[0])
```

```
the most popular event is MainScreenAppear
```

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

In [350]:
```
def find_share(group1, group2, event):
    print(
        f'share of users from {group1}: {round(pivot.loc[event, group1]/pivot.loc[event, :].sum() * 100,
    print(
        f'share of users from {group2}: {round(pivot.loc[event, group2]/pivot.loc[event, :].sum() * 100,
    print()
```

In [351]:
```
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%.
```

Conclusion

The number of users looks properly distributed and have equal proportions

The share of the number of users on every stage

In [352]:
```
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%.
```

Conclusion

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

Significance level of the statistical hypotheses

Calculate how many statistical hypothesis tests you carried out.

In [353]:
```
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 needed 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

Bonferroni

In [354]:
```
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
consider 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 t
o consider the proportions different
```

Nothing new here

Šidák

In [355]:
```
Šidák = 1 - pow((1-alpha),number_of_tests)
```

In [356]:
```
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 248 is 1231, for event: CartScreenAppear out of 2537 trials

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

p-value for CartScreenAppear:  0.08328412977507749
Rejecting the null hypothesis for 248 and 246 on event CartScreenAppear: there is a significant differ
ence between the proportions


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

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

p-value for Tutorial:  0.8964489622133207
Failed to reject the null hypothesis for 248 and 246 on event Tutorial: there is no reason to consider
the proportions different
```

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

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 contains 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 first week data we lost only lost 0.82% of data, that included visits of 1319 users.
Then 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 percentage of user loss happens between MainScreenAppear and OffersScreenAppear, and we should pay more attention to it and to find the 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.

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