Table of Contents

- 1 Research purpose
- 2 Research description
- 3 Data preprocessing
  - 3.1 Columns names correction
  - 3.2 Data and time columns creation
  - 3.3 Interim summary
- 4 Data analysis
  - 4.1 Measuring main parameters (unique users per action etc)
  - 4.2 Division by sessions
  - 4.3 Anomaly detection and elimination
  - 4.4 Interim summary
- 5 Main purpose of the study
  - 5.1 Histogram of events
  - 5.2 Division by events chains
  - 5.3 Impact of events leading to target event by all funnel scenarios
  - 5.4 Timediffer between the prevalent events
  - 5.5 Interim summary
- 6 Hypotheses statement
  - 6.1 First hypothesis check
  - 6.2 Second hypothesis check
  - 6.3 Third hypothesis check
  - 6.4 Interim summary
- 7 Project summary
- 8 Recommendations

Mobile application users behavior analysis

Atachments

- Presentation https://docs.google.com/presentation/d/1PGqQJab22Jl8T3LzK3dtvW4qYUpPVVBiHSoXqHGxvpw/edit?usp=sharing
- Dashboard https://public.tableau.com/views/Project11_DB_2022_11_24/Dashboard1?:language=en-US&publish=yes&:display_count=n&:origin=viz_share_link

Research purpose

- Engagement management enhancement
- Application growth points detection
- User experience improvement
- Profit increase

Research description

Analyze the impact of events on the completion of the target event — viewing contacts

The dataset contains data on events committed in the "Unnecessary Things" mobile application. In it, users sell their unwanted items by posting them on a bulletin board.

The dataset contains users data that was performed in the application for the first time after October 7, 2019.

Columns in *mobile_sources.csv*:

- "userId" — user ID,
- "source' — the source from which the user installed the application.

Columns in *mobile_dataset.csv*:

- "event.time' — the time of the transaction,
- "user.id " — user ID,
- "event.name " — user action.

Types of actions:

- "advert_open' — opened the ad cards,
- "photos_show' — viewed photos in the ad,
- "tips_show" — saw recommended ads,
- "tips_click" — clicked on the recommended ad,

- "contacts_show" and "show_contacts' — looked at the phone number,
- "contacts_call' — called the number from the ad,
- "map" — opened the ad map,
- "search_1"—"search_7" — various actions related to site search,
- "favorites_add" — added the ad to favorites.

Data preprocessing

In [1]:

```
#
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from datetime import datetime, timedelta
from plotly import graph_objects as go
import folium
from folium import Map, Choropleth, Marker
from folium.plugins import MarkerCluster
from folium.features import CustomIcon
from numpy import median
import re
import os
import json
import numpy as np
from plotly.subplots import make_subplots
from scipy import stats as st
import math as mth
```

In [2]:

```
pth1 = 'mobile_sources.csv'
pth2 = 'datasets/mobile_sources.csv'

if os.path.exists(pth1):
    sources = pd.read_csv(pth1, sep=',')
elif os.path.exists(pth2):
    sources = pd.read_csv(pth2, sep=',')
else:
    print('Path not found')
```

In [3]:

```
pth1 = 'mobile_dataset.csv'
pth2 = 'datasets/mobile_dataset.csv'

if os.path.exists(pth1):
    data = pd.read_csv(pth1, sep=',', parse_dates=['event.time'])
elif os.path.exists(pth2):
    data = pd.read_csv(pth2, sep=',', parse_dates=['event.time'])
else:
    print('Path not found')
```

In [4]:

```
#
def info(data):
    print('------------- First 5 lines ------------')
    display(data.sample(5))
    print('------------- Data types ------------')
    display(data.info())
    print('------------- Gaps ------------')
    for element in data.columns:
        if data[element].isna().any().mean() > 0:
            print(element, '-', data[element].isna().sum())
        else:
            print(element, '- None')
    print('------------- Duplicates ------------')
    if data.duplicated().sum() > 0:
        print(data.duplicated().sum())
    else:
        print('No Duplicates');
```

In [5]:

```
info(sources)
```

```
------------- First 5 lines ------------
```

	userId	source
4031	32668faa-0bfb-4c27-a45d-fdc494630396	yandex
2725	1e7a47ee-f831-41c6-8dde-3706b0fe6453	other
3047	bf90fc2f-501e-4709-8e4f-596d0167e1e2	google
1868	ee5216e6-36c7-4bb7-825d-bbb263e5ddc1	yandex
459	cbbf0d1b-6f03-4f40-bde0-40ca861f039e	other

```
------------- Data types ------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4293 entries, 0 to 4292
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
```

```
---  ------  --------------  -----
 0   userId  4293 non-null   object
 1   source  4293 non-null   object
dtypes: object(2)
memory usage: 67.2+ KB
None
------------- Gaps ------------
userId - None
source - None
------------- Duplicates ------------
No Duplicates
```

In [6]:

```
info(data)
```

```
------------- First 5 lines ------------
```

	event.time	event.name	user.id
61259	2019-10-30 07:32:08.253498	photos_show	e13f9f32-7ae3-4204-8d60-898db040bcfc
64917	2019-10-31 12:24:29.710670	tips_show	be1449f6-ca45-4f94-93a7-ea4b079b8f0f
2700	2019-10-08 09:16:29.858838	tips_show	6fefbd6f-7053-4c0c-818d-625c83fe5c5f
35778	2019-10-21 16:31:53.725613	tips_show	b26ccf72-bae0-45f4-8d92-8cde7598b25b
73648	2019-11-03 21:03:50.275278	tips_show	45c8980c-6840-42ec-8f23-152ad6c8e1e3

```
------------- Data types ------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74197 entries, 0 to 74196
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   event.time  74197 non-null  datetime64[ns]
 1   event.name  74197 non-null  object
 2   user.id     74197 non-null  object
dtypes: datetime64[ns](1), object(2)
memory usage: 1.7+ MB
None
------------- Gaps ------------
event.time - None
event.name - None
user.id - None
------------- Duplicates ------------
No Duplicates
```

Columns names correction

In [7]:

```
data = data.rename(columns={'event.time':'event_time', 'event.name':'event_name', 'user.id':'user_id'})
sources = sources.rename(columns={'userId':'user_id'})
display(data.sample(), sources.sample())
data.to_csv('data_clear.csv')
```

	event_time	event_name	user_id
17839	2019-10-14 17:36:41.916783	photos_show	2a6bd897-47ac-47d5-b4e0-97cccf574548

	user_id	source
201	bfd6629a-7393-41ef-abb2-7143b1ab3faf	google

Data and time columns creation

In [8]:

```
data['date'] = pd.DatetimeIndex(data['event_time']).date
data['time'] = pd.DatetimeIndex(data['event_time']).time
data.sample()
```

Out[8]:

	event_time	event_name	user_id	date	time
28911	2019-10-18 19:26:41.382996	tips_show	dc179afe-96e0-4330-a09f-8626a193e09f	2019-10-18	19:26:41.382996

In [9]:

```
display(f'First date {data["event_time"].min()}')
display(f'Last date {data["event_time"].max()}')
```

```
'First date 2019-10-07 00:00:00.431357'
'Last date 2019-11-03 23:58:12.532487'
```

Interim summary

There are no Duplicates nor gaps in both datasets.

Column names has been brought to unified style.

Observation period is nearly a month: 2019-10-07 - 2019-11-03

Data analysis

Measuring main parameters (unique users per action etc)

In [10]:
```
display(f'{data["user_id"].nunique()} Unique users ')
```
```
'4293 Unique users '
```
In [11]:
```
display(f'{data["event_name"].nunique()} Unique events ')
```
```
'16 Unique events '
```
In [12]:
```
display('Unique users per event', data.groupby("event_name")["user_id"].nunique().sort_values(ascending=False))
```
```
'Unique users per event'
event_name
tips_show        2801
map              1456
photos_show      1095
contacts_show     979
search_1          787
advert_open       751
search_5          663
search_4          474
favorites_add     351
search_6          330
tips_click        322
search_2          242
contacts_call     213
search_3          208
search_7          157
show_contacts       7
Name: user_id, dtype: int64
```
Let's interchange 7 events "show_contacts" to "contacts_show" as they seem literally single-valued.

In [13]:
```
data['event_name'] = data['event_name'].replace({'show_contacts':'contacts_show'})
display('Unique users per event', data.groupby("event_name")["user_id"].nunique().sort_values(ascending=False))
```
```
'Unique users per event'
event_name
tips_show        2801
map              1456
photos_show      1095
contacts_show     981
search_1          787
advert_open       751
search_5          663
search_4          474
favorites_add     351
search_6          330
tips_click        322
search_2          242
contacts_call     213
search_3          208
search_7          157
Name: user_id, dtype: int64
```
Division by sessions

The logic behind division by session has been taken as follows:
We assume that session is a flow of events taken by unique user with timediffer between events of one user more then 20 minutes.

In [14]:
```
sessions = (data.sort_values(['user_id','event_time']).groupby('user_id')['event_time'].diff() > pd.Timedelta('20Min')).cumsum()
data['session_id'] = data.groupby(['user_id', sessions], sort = False).ngroup() + 1
display(data.head(5))
```

	event_time	event_name	user_id	date	time	session_id
0	2019-10-07 00:00:00.431357	advert_open	020292ab-89bc-4156-9acf-68bc2783f894	2019-10-07	00:00:00.431357	1
1	2019-10-07 00:00:01.236320	tips_show	020292ab-89bc-4156-9acf-68bc2783f894	2019-10-07	00:00:01.236320	1
2	2019-10-07 00:00:02.245341	tips_show	cf7eda61-9349-469f-ac27-e5b6f5ec475c	2019-10-07	00:00:02.245341	2
3	2019-10-07 00:00:07.039334	tips_show	020292ab-89bc-4156-9acf-68bc2783f894	2019-10-07	00:00:07.039334	1
4	2019-10-07 00:00:56.319813	advert_open	cf7eda61-9349-469f-ac27-e5b6f5ec475c	2019-10-07	00:00:56.319813	2

In [15]:
```
display(f'Out of all {len(data)} events, There are {data.session_id.nunique()} sessions')
```
```
'Out of all 74197 events, There are 10975 sessions'
```
Anomaly detection and elimination

In [16]:
```
display(f'Out of {data.session_id.nunique()} sessions, There are {(data.groupby("session_id")["event_name"].count() == 1).sum()} session
```
```
'Out of 10975 sessions, There are 2328 sessions consisting of 1 event'
```
As our goal is to analyze the impact of preceding events on the target event "contacts_show",
we can eliminate sessions consisting of 2 unique events by creating following filters by sessions:

In [17]:

```
session1 = data.groupby('session_id')['event_name'].nunique().reset_index()
session1 = session1.query('event_name > 2')
len(session1)
```

Out[17]: 2296

Creation of a new dataset "data2" containing only sessions longer than 1 event and with variety of 2 or more events

In [18]:

```
data2 = data.query('session_id in @session1.session_id')
data2.head(5)
```

Out[18]:

	event_time	event_name	user_id	date	time	session_id
0	2019-10-07 00:00:00.431357	advert_open	020292ab-89bc-4156-9acf-68bc2783f894	2019-10-07	00:00:00.431357	1
1	2019-10-07 00:00:01.236320	tips_show	020292ab-89bc-4156-9acf-68bc2783f894	2019-10-07	00:00:01.236320	1
2	2019-10-07 00:00:02.245341	tips_show	cf7eda61-9349-469f-ac27-e5b6f5ec475c	2019-10-07	00:00:02.245341	2
3	2019-10-07 00:00:07.039334	tips_show	020292ab-89bc-4156-9acf-68bc2783f894	2019-10-07	00:00:07.039334	1
4	2019-10-07 00:00:56.319813	advert_open	cf7eda61-9349-469f-ac27-e5b6f5ec475c	2019-10-07	00:00:56.319813	2

In [19]:

```
display(f'Data has been reduced to {len(data2)} from {len(data)}')
display(f'Number of sessions has been reduced to {data2["session_id"].nunique()} from {data["session_id"].nunique()}')
```

```
'Data has been reduced to 29492 from 74197'
'Number of sessions has been reduced to 2296 from 10975'
```

Interim summary

10975 sessions has been allocated.

New dataset 'data2' has been created: containing only sessions longer than 2 unique events and containing target event. In result:

Data has been reduced to 29492 from 74197 and

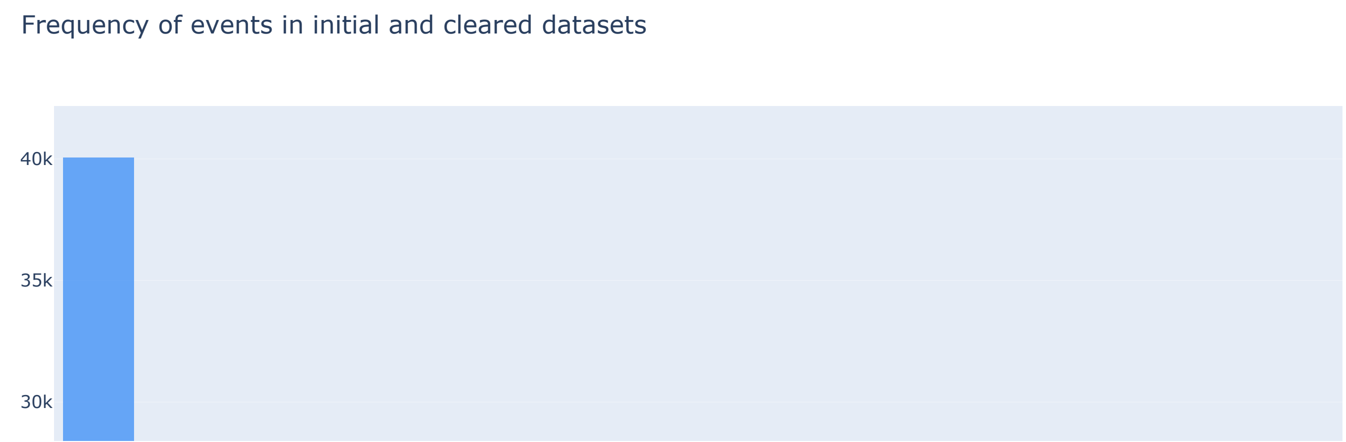
Number of sessions has been reduced to 2296 from 10975

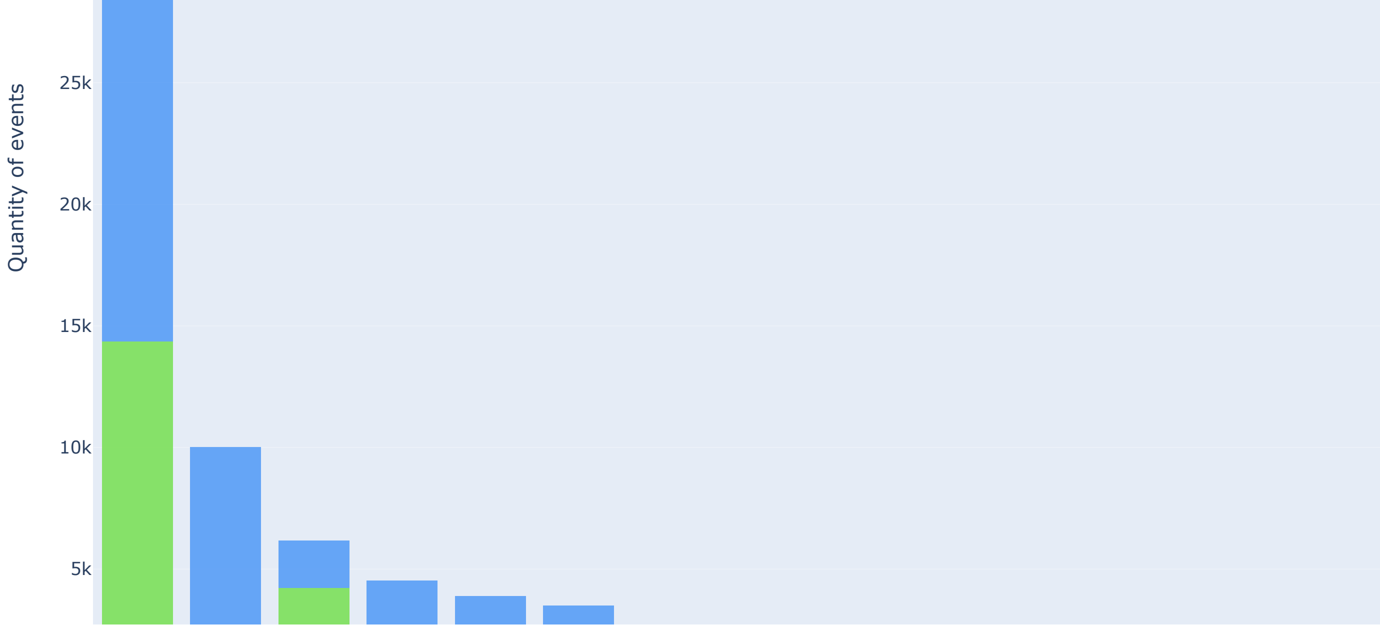
Main purpose of the study

Histogram of events

In [20]:

```
#
trace1 = go.Histogram(
    x=data['event_name'],
    opacity=0.75,
    name='Initial Data',
    marker_color='#3A8DF6')
trace2 = go.Histogram(
    x=data2['event_name'],
    opacity=0.75,
    name='Cleared Data',
    marker_color='#92f63a')

graph = [trace1, trace2]
layout = go.Layout(
    title='Frequency of events in initial and cleared datasets',
    barmode='overlay',
    height=900,
    xaxis=dict(
        title='Event name',
        categoryorder='total descending'),
    yaxis=dict(
        title='Quantity of events',
        overlaying='y'),)

fig = go.Figure(data=graph, layout=layout)
fig.show()
```





After clearing the data it appeared that the most popular events "tips_show", "photos_show" and "search_1" in 70% of the time is involved into pointless sessions conisting only of the one event.
Directed conscious actions demanding an active action from user like "advert_open", "contacts_show" and "map" has shown less reduction.

Division by events chains

In [21]:

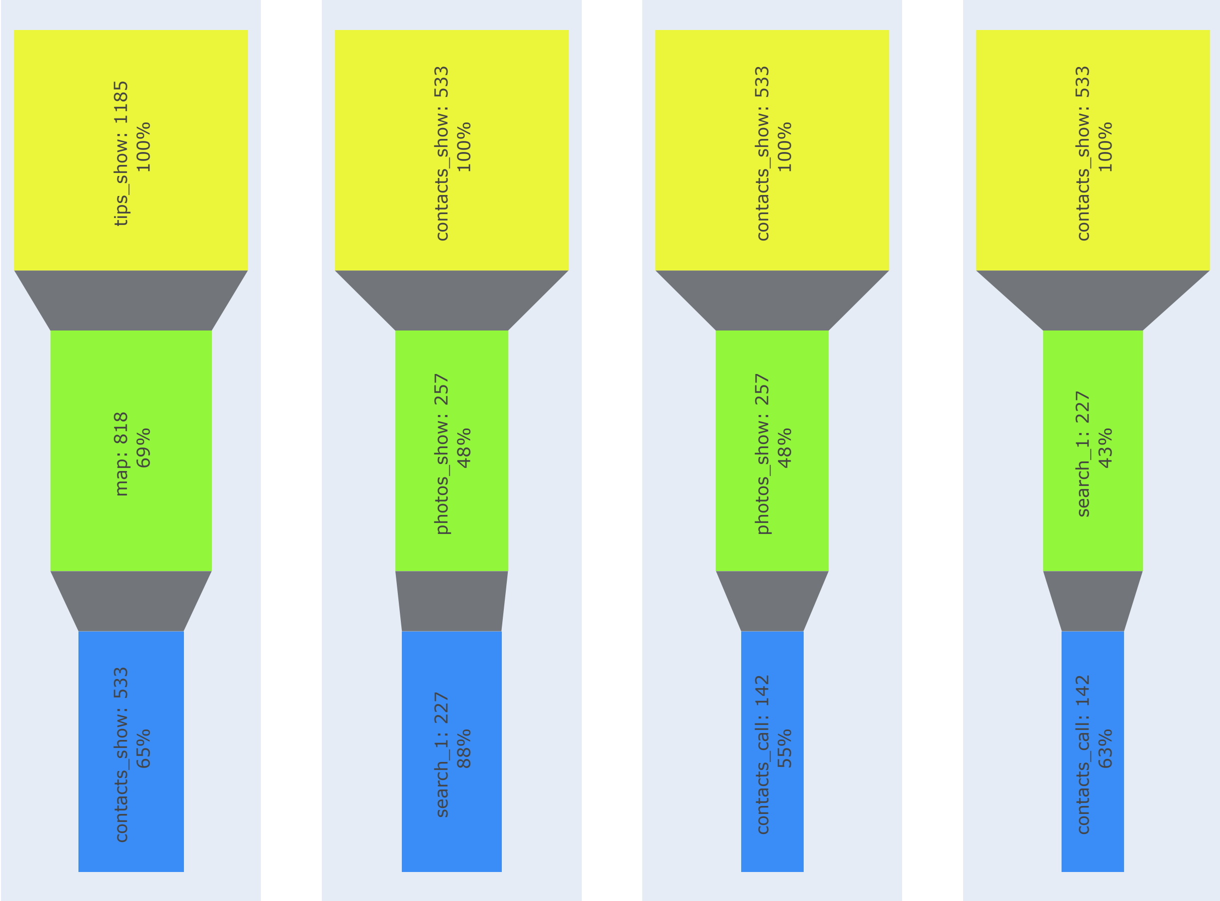
```
session2 = data2.sort_values(by='event_name').groupby('session_id', as_index=False).agg(event_flow=('event_name', 'unique'))
session2['event_flow'] =  [', '.join(map(str, i)) for i in session2['event_flow']]

funnel = session2['event_flow'].value_counts()
display('Top-10 event steps:', funnel.head(10))
```

```
'Top-10 event steps:'
advert_open, map, tips_show                   300
contacts_show, map, tips_show                 176
advert_open, map, search_3, tips_show          86
search_4, search_5, search_6, tips_show        80
contacts_show, photos_show, search_1           75
search_5, search_6, tips_show                  75
favorites_add, photos_show, search_1           64
search_4, search_5, tips_show                  63
contacts_call, contacts_show, photos_show      62
contacts_call, contacts_show, search_1         59
Name: event_flow, dtype: int64
```

In [22]:

```
#
fig = make_subplots(rows=1, cols=4)

fig.update_layout(showlegend=False,height=800, width=1000,
                    title=(f'Funnels of Top-4 successful event scenarios in cleared data'), title_font_size = 20 )

session_suc = session2.query('event_flow.str.contains("contacts_show")', engine='python')
funnel_suc = session_suc['event_flow'].value_counts()

for i in range(0,4):
    f = funnel_suc.index[i].split(', ')
    fun = data2.query('event_name == @f').groupby('event_name', as_index=False)['user_id'].nunique().sort_values(by='user_id', ascending
    fig.add_trace(go.Funnel(
    x = fun['user_id'] ,
    y = fun['event_name'],

    textposition = "inside",
    texttemplate="%{label}: %{value:} <br> %{percentPrevious}",
    textangle=-90,
    marker = {"color": ["#EBF63A", "#92f63a", "#3A8DF6"]}),
    row=1, col=i+1 )

fig.update_yaxes(visible=False)
fig.show()
```

Funnels of Top-4 successful event scenarios in cleared data



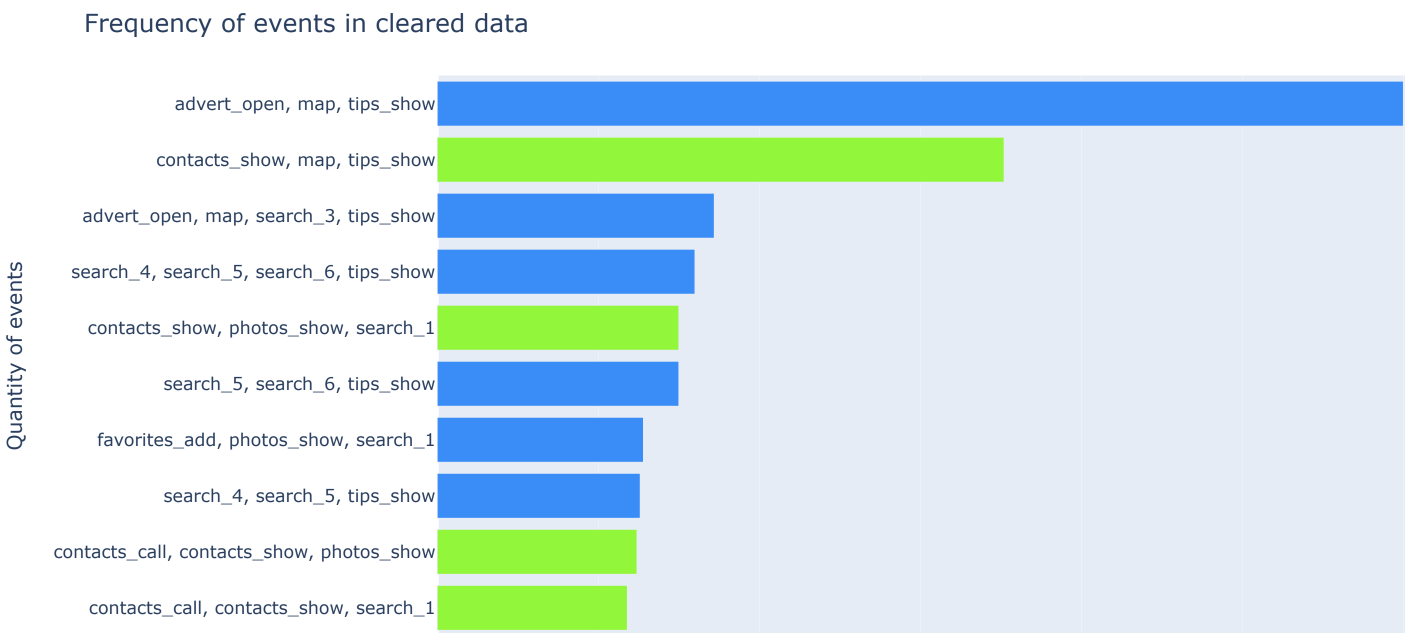
Impact of events leading to target event by all funnel scenarios

From Funnel we can see that the most common events leading to "Contacts show" are the following:

- tips_show
- map
- photos_show
- search_1

In [23]:

```
fig = px.bar( y=funnel[:10].index,  x = funnel[:10].values, color=funnel[:10].index.str.contains('contacts_show'),
                   color_discrete_sequence=['#3A8DF6', '#92f63a'])
fig.update_layout(title='Frequency of events in cleared data',
                  yaxis={'categoryorder':'total ascending'},
                  yaxis_title='Quantity of events',
                  xaxis_title='Event name',
                  legend_title="Hit Target event")
fig.show()
```



In [24]:

```
events_main = ['tips_show', 'map', 'photos_show', 'search_1', 'favorites_add', 'advert_open',  'tips_click']
for i in range(len(events_main)):
    x = events_main[i]
    target = "contacts_show"
    a = data2.query("event_name == @x")
    b = data2.query("session_id in @a.session_id and event_name == @target")
    au = a.user_id.nunique()
    bu = b.user_id.nunique()
    display(f'Percent of "{events_main[i]}" events in initial data that lead to target "Contacts show": ')
    display(f'{ bu/au :.3%}')
```

```
'Percent of "tips_show" events in initial data that lead to target "Contacts show": '
'25.063%'
'Percent of "map" events in initial data that lead to target "Contacts show": '
'26.039%'
'Percent of "photos_show" events in initial data that lead to target "Contacts show": '
'66.537%'
'Percent of "search_1" events in initial data that lead to target "Contacts show": '
'66.960%'
'Percent of "favorites_add" events in initial data that lead to target "Contacts show": '
'42.661%'
'Percent of "advert_open" events in initial data that lead to target "Contacts show": '
'16.491%'
'Percent of "tips_click" events in initial data that lead to target "Contacts show": '
'37.349%'
```

We can see that the user action "tips_show" and "map" are the most common ways to reach target event.
But this actions make only around 25% conversion to "Contacts show".

> It seems that the raise of conversion of "tips_show" and "map" will increase number of "Contacts show" events.

"Photos_show" and "search_1" make around 67% conversion to "Contacts show".

> Raise of commiting "photos_show" and "search_1" is also an option, as they show the biggest conversion.

"Favorites_add" makes 42% conversion to "Contacts show".

> Raise of commiting "Favorites_add"

Also "tips_click" is showing pretty good conversion -38%, but takes a small part of events flows.

> Raise of commiting "tips_click"

"Advert_open" makes the smallest conversion to target event - 15%

> Raise of conversion "Advert_open"

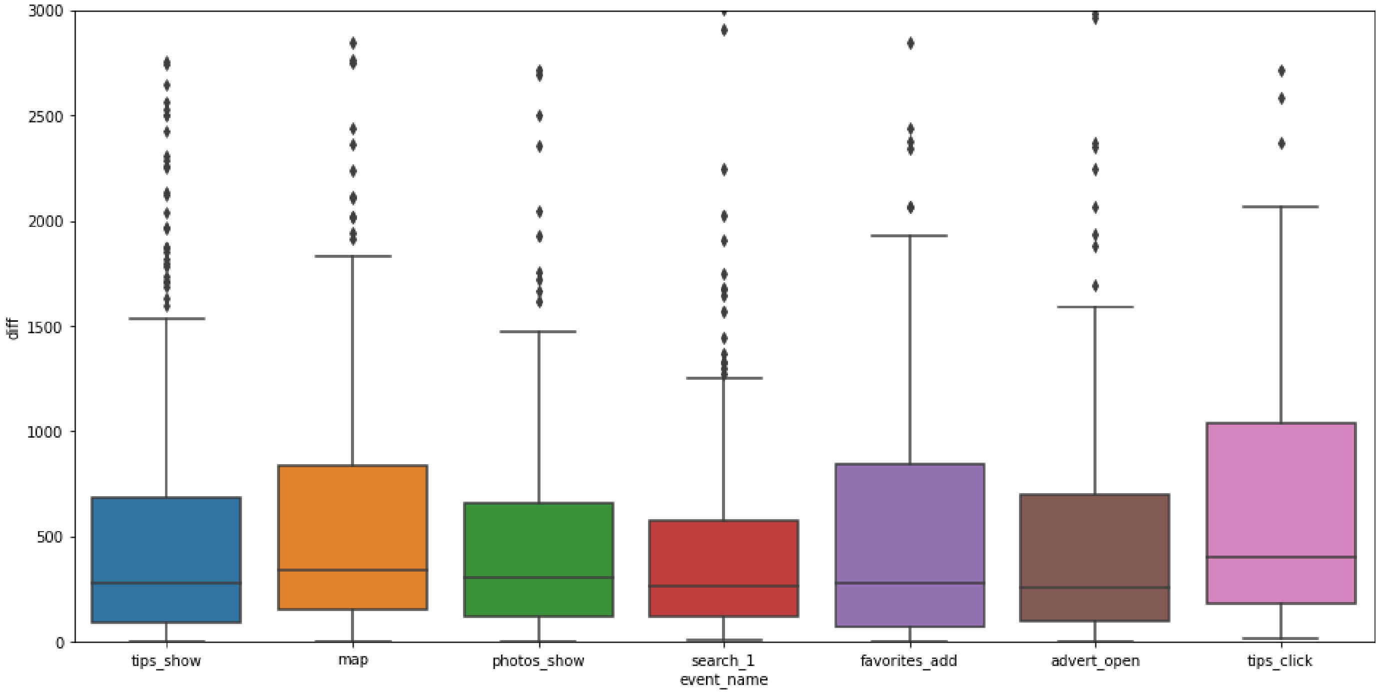
Timediffer between the prevalent events

In [25]:

```
#
session = []
for i in range(len(events_main)):
    x = events_main[i]

    session_x = data2.query('session_id in @session_suc.session_id and event_name == @x').groupby('session_id', as_index=False).agg({'ev
    session_t = data2.query('session_id in @session_x.session_id and event_name == @target').groupby('session_id', as_index=False)['even
    session_x = session_x.merge(session_t, on='session_id')
    session_x['diff'] = session_x['event_time_x'] - session_x['event_time_y']
    session_x['diff'] = session_x['diff'].dt.total_seconds().abs()

    display(f'"{events_main[i]}" and "Contacts show" median timediffer:')
    display(f'{ session_x["diff"].median() :.5}, seconds')
    session.append(session_x)
session = pd.concat(session)
display(f'Mean timediffer between prevalent events and target "Contacts show":{ session["diff"].median() :.5}, seconds')
plt.figure(figsize=(16, 8))
plt.ylim([-1,3000])
print(sns.boxplot(x='event_name', y='diff',  data=session))
```

```
'"tips_show" and "Contacts show" median timediffer:'
'278.46, seconds'
'"map" and "Contacts show" median timediffer:'
'342.5, seconds'
'"photos_show" and "Contacts show" median timediffer:'
'306.74, seconds'
'"search_1" and "Contacts show" median timediffer:'
'262.93, seconds'
'"favorites_add" and "Contacts show" median timediffer:'
'277.9, seconds'
'"advert_open" and "Contacts show" median timediffer:'
'260.23, seconds'
'"tips_click" and "Contacts show" median timediffer:'
'404.52, seconds'
```

```
'Mean timediffer between prevalent events and target "Contacts show":302.63, seconds'
AxesSubplot(0.125,0.125;0.775x0.755)
```



Interim summary

We've substracted 5 most popular sessions scenarios:

advert_open, map, tips_show 300

contacts_show, map, tips_show 176

advert_open, map, search_3, tips_show 86

search_4, search_5, search_6, tips_show 80

contacts_show, photos_show, search_1 75

We can see that the user action "tips_show" and "map" are the most common ways to reach target event.
But this actions make only around 25% conversion to "Contacts show".

> It seems that the raise of conversion of "tips_show" and "map" will increase number of "Contacts show" events.

"Photos_show" and "search_1" make around 67% conversion to "Contacts show".

> Raise of commiting "photos_show" and "search_1" is also an option, as they show the biggest conversion.

"Favorites_add" makes 42% conversion to "Contacts show".

> Raise of commiting "Favorites_add"

Also "tips_click" is showing pretty good conversion -38%, but takes a small part of events flows.

> Raise of commiting "tips_click"

"Advert_open" makes the smallest conversion to target event - 15%

> Raise of conversion "Advert_open"

Mean timediffer between prevalent events and target "Contacts show" is around 300 seconds (5 minutes)

Hypotheses statement

First hypothesis check

H0: Conversion to contact views is equal between two groups: ones who perform tips_show and tips_click actions, and ones with only tips_show.
H1: Conversion to contact views between two groups differs

In [26]:

```
def z_test(successes1, successes2, trials1, trials2, alpha=0.05):
    p1 = successes1 / trials1
    p2 = successes2 / trials2

    p_combined = (successes1 + successes2) / (trials1 + trials2)
    difference = p1 - p2

    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('p-value: ', "%.20f" % p_value )

    if (p_value < alpha):
        display('Reject the null hypothesis, there are statistically significant differences between the samples')
```

```
    else:
        display('It was not possible to reject the null hypothesis, there are no statistically significant differences in the samples')
```

In [27]:
```
session5 = data.sort_values(by='event_name').groupby('session_id', as_index=False).agg(event_flow=('event_name', 'unique'))
session5['event_flow'] =  [', '.join(map(str, i)) for i in session5['event_flow']]

target1 = 'tips_show'
target2 = 'tips_click'
success = 'contacts_show'

x1 = session5.query('event_flow.str.contains(@target1)', engine='python')
x2 = session5.query('event_flow.str.contains(@target1) and event_flow.str.contains(@target2)', engine='python')

trials1 = data.query('session_id in @x1.session_id')['user_id'].nunique()
trials2 = data.query('session_id in @x2.session_id')['user_id'].nunique()

y1 = x1.query('event_flow.str.contains(@success)', engine='python')
y2 = x2.query('event_flow.str.contains(@success)', engine='python')

successes1 = data.query('session_id in @y1.session_id')['user_id'].nunique()
successes2 = data.query('session_id in @y2.session_id')['user_id'].nunique()

z_test(successes1, successes2, trials1, trials2, alpha=0.05)
```

```
p-value:  0.15008551289817906316
'It was not possible to reject the null hypothesis, there are no statistically significant differences in the samples'
```

Second hypothesis check

H0: Conversion to contact views is equal between group of users who performed **favorites_add** and who did not.

H1: Conversion to contact views between two groups differs

In [28]:
```
target = 'favorites_add'
success = 'contacts_show'

x1 = session5.query('event_flow.str.contains(@target)', engine='python')
x2 = session5.query('~event_flow.str.contains(@target)', engine='python')

trials1 = data.query('session_id in @x1.session_id')['user_id'].nunique()
trials2 = data.query('session_id in @x2.session_id')['user_id'].nunique()

y1 = x1.query('event_flow.str.contains(@success)', engine='python')
y2 = x2.query('event_flow.str.contains(@success)', engine='python')

successes1 = data.query('session_id in @y1.session_id')['user_id'].nunique()
successes2 = data.query('session_id in @y2.session_id')['user_id'].nunique()

z_test(successes1, successes2, trials1, trials2, alpha=0.05)
```

```
p-value:  0.013361153337488821854
'Reject the null hypothesis, there are statistically significant differences between the samples'
```

Third hypothesis check

H0: Conversion to contact views is equal between group of users who used **photos_show** and who did not.

H1: Conversion to contact views between two groups differs

In [29]:
```
target = 'photos_show'
success = 'contacts_show'

x1 = session5.query('event_flow.str.contains(@target)', engine='python')
x2 = session5.query('~event_flow.str.contains(@target)', engine='python')

trials1 = data.query('session_id in @x1.session_id')['user_id'].nunique()
trials2 = data.query('session_id in @x2.session_id')['user_id'].nunique()

y1 = x1.query('event_flow.str.contains(@success)', engine='python')
y2 = x2.query('event_flow.str.contains(@success)', engine='python')

successes1 = data.query('session_id in @y1.session_id')['user_id'].nunique()
successes2 = data.query('session_id in @y2.session_id')['user_id'].nunique()

z_test(successes1, successes2, trials1, trials2, alpha=0.05)
```

```
p-value:  0.69348793142563436298
'It was not possible to reject the null hypothesis, there are no statistically significant differences in the samples'
```

Interim summary

There are no statistically significant differences in conversion to "contacts_show" between the ones who perform **tips_show and tips_click** actions, and ones with only **tips_show** .

There are statistically significant differences in conversion to "contacts_show" between the users who performed **favorites_add** and who did not.

There are no statistically significant differences in conversion to "contacts_show" between the users who performed **photos_show** and who did not.

Project summary

10975 sessions has been allocated and has been reduced to 2296 sessions after eliminating double unique event sessions.

The most common scenarios of successful event flow are:

- contacts_show, map, tips_show 176
- contacts_show, photos_show, search_1 75
- contacts_call, contacts_show, photos_show 62
- contacts_call, contacts_show, search_1 59

"Tips_show" and "map" are the most common ways to reach target event.
But this actions make only around 25% conversion to "Contacts show".

"Photos_show" and "search_1" make around 67% conversion to "Contacts show".

"Favorites_add" makes 42% conversion to "Contacts show".

Also "tips_click" is showing pretty good conversion -38%, but takes a small part of events flows.

"Advert_open" makes the smallest conversion to target event - 15%

Mean timediffer between prevalent events and target "Contacts show" is around 5 minutes

---

There are statistically significant differences in conversion to "contacts_show" between the users who performed **favorites_add** and who did not.

There are no statistically significant differences in conversion to "contacts_show" between the ones who perform **tips_show and tips_click** actions, and ones with only **tips_show** .

There are no statistically significant differences in conversion to "contacts_show" between the users who performed **photos_show** and who did not.

Recommendations

In the order of impact:

1. Raise the conversion of "tips_show" and "map"
2. Increase commission "photos_show" and "search_1"
3. Increase commission "Favorites_add"
4. Increase commission "tips_click"
5. Raise the conversion "Advert_open"