Table of Contents

A/B тестирование

Даны 4 датасета с информамцией о пользователях, их действиях, разбивке на группы A/B-тестов и календарь маркетингов мероприятий. **Необходимо проанализировать A/B-тестирование.**
Основные задачи:

- оценка корректности проведения теста
- проверка статистической разницы долей z-критерием
- анализ результатов теста

Техническое задание

- **Название теста:** recommender_system_test ;
- **группы:** A — контрольная, B — новая платёжная воронка;
- **дата запуска:** 2020-12-07;
- **дата остановки набора новых пользователей:** 2020-12-21;
- **дата остановки:** 2021-01-04;
- **аудитория:** в тест должно быть отобрано 15% новых пользователей из региона EU;
- **назначение теста:** тестирование изменений, связанных с внедрением улучшенной рекомендательной системы;
- **ожидаемое количество участников теста:** 6000.
- **ожидаемый эффект:** за 14 дней с момента регистрации пользователи покажут улучшение каждой метрики не менее, чем на 10%: конверсии в просмотр карточек товаров — событие product_page , просмотры корзины — product_cart , покупки — purchase .

Описание данных

1. ab_project_marketing_events.csv — календарь маркетинговых событий на 2020 год. Структура файла:

- name — название маркетингового события;
- regions — регионы, в которых будет проводиться рекламная кампания;
- start_dt — дата начала кампании;
- finish_dt — дата завершения кампании.

2. final_ab_new_users.csv — пользователи, зарегистрировавшиеся с 7 до 21 декабря 2020 года. Структура файла:

- user_id — идентификатор пользователя;
- first_date — дата регистрации;
- region — регион пользователя;
- device — устройство, с которого происходила регистрация

3. final_ab_events.csv — действия новых пользователей в период с 7 декабря 2020 по 4 января 2021 года. Структура файла:

- user_id — идентификатор пользователя;
- event_dt — дата и время события;
- event_name — тип события;
- details — дополнительные данные о событии. Например, для покупок, purchase, в этом поле хранится стоимость покупки в долларах.

4. final_ab_participants.csv — таблица участников тестов. Структура файла:

- user_id — идентификатор пользователя;
- ab_test — название теста;
- group — группа пользователя.

План работы

1. Импорт необходимых библиотек, открытие датасетов
2. Предобработка данных:

- обработка пропусков, дубликатов, приведение данных к корректным типам

3. Оценка корректности проведения теста в соответствии с ТЗ и общими принципами корректности. Проверка:
   - аудитории теста
     - пересечение с конкурирующим тестом
     - пересечениепо группам теста
     - равномерность распределения учсатников по группам
   - дат
     - нет ли пользователей, пришедших после даты окончания набора новых
     - нет ли действий, совершенных до старта теста и после его окончания
   - распределения пользователей по регионам
     - достаточное ли количество участников из EU
     - есть ли пересечение с региональными маркетинговыми активностями
   - наличия совершенных действий у каждого пользователя
   - лайфтайм
     - все ли пользователи успели "прожить" 14 дней?
     - есть ли действия, совершенные после 14-го лайфтайма?
   - оценка оставшихся участников тестов
   - характеристика корректности проведения теста
4. Исследовательский анализ данных:
   - как распрделены события по пользователям в выборках
   - как число событий в выборках распределено по дням
   - как отличаются конверсии двух групп, наблюдается ли относительный прирост группы B по отношению к A
   - какие особенности данных нужно учесть, прежде чем приступать к A/B-тестированию
5. Оценка результатов A/B-тестирования
   - что можно сказать про результаты A/B-тестирования
   - проверка статистической разницы долей z-критерием
   - выводы и общее заключение о корректности проведения теста.

Импорт библиотек, открытие файлов

In [1]:
```
import pandas as pd

import datetime as dt
from datetime import timedelta

import numpy as np

from scipy import stats as st

import math as mth

import matplotlib.pyplot as plt
plt.style.use('ggplot')
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import plotly
import plotly.express as px
from plotly import graph_objects as go
from plotly.subplots import make_subplots
import plotly.io as io
io.templates.default = 'ggplot2'
```

```
io.renderers.default = "png"
svg_renderer = io.renderers["png"]
svg_renderer.scale = 1.2
```

In [2]:
```
try:
    project = pd.read_csv(r'\Users\Hp\Desktop\Практикум\FINAL\ab_project_marketing_events.csv
    new_users = pd.read_csv(r'\Users\Hp\Desktop\Практикум\FINAL\final_ab_new_users.csv')
    participants = pd.read_csv(r'\Users\Hp\Desktop\Практикум\FINAL\final_ab_participants.csv'
    events = pd.read_csv(r'\Users\Hp\Desktop\Практикум\FINAL\final_ab_events.csv')
except:
    project = pd.read_csv('https://code.s3.yandex.net/datasets/ab_project_marketing_events.cs
    new_users = pd.read_csv('https://code.s3.yandex.net/datasets/final_ab_new_users.csv')
    participants = pd.read_csv('https://code.s3.yandex.net/datasets/final_ab_participants.csv
    events = pd.read_csv('https://code.s3.yandex.net/datasets/final_ab_events.csv')
```

In [3]:
```
display(project.head(3))
display(new_users.head(3))
display(participants.head(3))
events.head(3)
```

	name	regions	start_dt	finish_dt
0	Christmas&New Year Promo	EU, N.America	2020-12-25	2021-01-03
1	St. Valentine's Day Giveaway	EU, CIS, APAC, N.America	2020-02-14	2020-02-16
2	St. Patric's Day Promo	EU, N.America	2020-03-17	2020-03-19

	user_id	first_date	region	device
0	D72A72121175D8BE	2020-12-07	EU	PC
1	F1C668619DFE6E65	2020-12-07	N.America	Android
2	2E1BF1D4C37EA01F	2020-12-07	EU	PC

	user_id	group	ab_test
0	D1ABA3E2887B6A73	A	recommender_system_test
1	A7A3664BD6242119	A	recommender_system_test
2	DABC14FDDFADD29E	A	recommender_system_test

Out[3]:

	user_id	event_dt	event_name	details
0	E1BDDCE0DAFA2679	2020-12-07 20:22:03	purchase	99.99
1	7B6452F081F49504	2020-12-07 09:22:53	purchase	9.99
2	9CD9F34546DF254C	2020-12-07 12:59:29	purchase	4.99

Все таблицы загружены

Знакомство с данными. Предобработка

На этом этапе мы посмотрим на то, какие данные хранят таблицы, обработаем пропуски и дубликаты при необходимости, приведем данные к корректным типам. **Пересечения групп, соответствие дат и другие характеристики для оценки корректности проведения A/B-теста будут проанализированы в соответствующем блоке**

project

In [4]:
```
display(project)
project.info()
```

	name	regions	start_dt	finish_dt
0	Christmas&New Year Promo	EU, N.America	2020-12-25	2021-01-03
1	St. Valentine's Day Giveaway	EU, CIS, APAC, N.America	2020-02-14	2020-02-16
2	St. Patric's Day Promo	EU, N.America	2020-03-17	2020-03-19
3	Easter Promo	EU, CIS, APAC, N.America	2020-04-12	2020-04-19
4	4th of July Promo	N.America	2020-07-04	2020-07-11
5	Black Friday Ads Campaign	EU, CIS, APAC, N.America	2020-11-26	2020-12-01
6	Chinese New Year Promo	APAC	2020-01-25	2020-02-07
7	Labor day (May 1st) Ads Campaign	EU, CIS, APAC	2020-05-01	2020-05-03
8	International Women's Day Promo	EU, CIS, APAC	2020-03-08	2020-03-10
9	Victory Day CIS (May 9th) Event	CIS	2020-05-09	2020-05-11
10	CIS New Year Gift Lottery	CIS	2020-12-30	2021-01-07
11	Dragon Boat Festival Giveaway	APAC	2020-06-25	2020-07-01
12	Single's Day Gift Promo	APAC	2020-11-11	2020-11-12
13	Chinese Moon Festival	APAC	2020-10-01	2020-10-07

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   name       14 non-null     object
 1   regions    14 non-null     object
 2   start_dt   14 non-null     object
 3   finish_dt  14 non-null     object
dtypes: object(4)
memory usage: 576.0+ bytes
```

`Таблица включает всего 14 строк,` пропусков и дубликатов нет, столбцы имеют корректное название

- приведем столбцы с датой к корректному типу
- заменим 'N.America' на общепринятую аббревиатуру NA

In [5]:
```
# заменяем все 'N.America' в столбце 'regions'
project['regions'] = [elem.replace("N.America", "NA", 1) for elem in project['regions']]
#приводим тип данных в столбцах с датой к корректному:
project['start_dt'] = pd.to_datetime(project['start_dt'])
project['finish_dt'] = pd.to_datetime(project['finish_dt'])
```

Таблица project готова

new_users

In [6]:
```
display(new_users.head())
new_users.info()
```

	user_id	first_date	region	device
0	D72A72121175D8BE	2020-12-07	EU	PC
1	F1C668619DFE6E65	2020-12-07	N.America	Android
2	2E1BF1D4C37EA01F	2020-12-07	EU	PC
3	50734A22C0C63768	2020-12-07	EU	iPhone
4	E1BDDCE0DAFA2679	2020-12-07	N.America	iPhone

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 61733 entries, 0 to 61732
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   user_id     61733 non-null  object
 1   first_date  61733 non-null  object
 2   region      61733 non-null  object
 3   device      61733 non-null  object
dtypes: object(4)
memory usage: 1.9+ MB
```

`Таблица содержит 61733 наблюдения,` пропусков нет, столбцы имеют корректные названия

- проверим наличие строк-дубликатов, дубликатов в столбце 'user_id'
- приведем столбец с датой к корректному типу
- выявим временной промежуток
- проверим значения в столбцах с категориальными данными, обработаем неявные дубликаты при наличии

In [7]:
```
print('Число строк-дубликатов:', new_users.duplicated().sum())
print("Число дубликатов в столбце 'user_id':", new_users['user_id'].duplicated().sum())
```

```
Число строк-дубликатов: 0
Число дубликатов в столбце 'user_id': 0
```

In [8]:
```
# приводим столбец с датой к корректному типу, выявляем временной промежуток:
new_users['first_date'] = pd.to_datetime(new_users['first_date'])
print('Минимальная дата:', new_users['first_date'].min())
print('Максимальная дата:', new_users['first_date'].max())
print('Временной диапазон:', new_users['first_date'].max()-new_users['first_date'].min())
```

```
Минимальная дата: 2020-12-07 00:00:00
Максимальная дата: 2020-12-23 00:00:00
Временной диапазон: 16 days 00:00:00
```

Минимальная дата соответствует дате запуска теста, `максимальная превышает дату остановки набора новых пользователей.` На этом этапе просто фиксируем временной промежуток, главное, что не наблдюдаем каких-то слишком выбивающихся дат.

`Проверим категориальные данные`

In [9]:
```
print("Уникальные значения в столбце 'region':", new_users['region'].sort_values().unique().t
```

```
Уникальные значения в столбце 'region': ['APAC', 'CIS', 'EU', 'N.America']
```

`Неявных дубликатов не обнаружили,` заменим "N.America" по аналогии с предыдущим пунктом и посмотрим как распределены данные по регионам.

In [10]:
```
# заменяем все 'N.America' в столбце 'region'
new_users['region'] = [elem.replace("N.America", "NA", 1) for elem in new_users['region']]
new_users['region'].value_counts()
```

Out[10]:
```
EU      46270
NA       9155
CIS      3155
APAC     3153
Name: region, dtype: int64
```

Большинство наших пользователей из `европейского региона.`

In [11]:
```
print("Уникальные значения в столбце 'device':",new_users['device'].sort_values().unique().to
new_users['device'].value_counts()
```

```
Уникальные значения в столбце 'device': ['Android', 'Mac', 'PC', 'iPhone']
```

Out[11]:
```
Android    27520
PC         15599
iPhone     12530
Mac         6084
Name: device, dtype: int64
```

`Неявных дубликатов нет,` большинство клиентов используют Android.

Таблица 'new_users' готова

participants

In [12]:
```
display(participants.head())
participants.info()
```

	user_id	group	ab_test
0	D1ABA3E2887B6A73	A	recommender_system_test
1	A7A3664BD6242119	A	recommender_system_test
2	DABC14FDDFADD29E	A	recommender_system_test
3	04988C5DF189632E	A	recommender_system_test
4	482F14783456D21B	B	recommender_system_test

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18268 entries, 0 to 18267
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   user_id  18268 non-null  object
 1   group    18268 non-null  object
 2   ab_test  18268 non-null  object
dtypes: object(3)
memory usage: 428.3+ KB
```

`Таблица содержит 18268 наблдюдений,` пропусков нет, типы данных корректные, столбцы имеют корректные названия.

- проверим наличие строк-дубликатов, дубликатов в столбце 'user_id'
- проверим какие значения хранятся в столбцах с категориальными данными

In [13]:
```
print('Число строк-дубликатов:',participants.duplicated().sum())
print('Число дубликатов "user_id":', participants['user_id'].duplicated().sum())
print('Число дубликатов в связке "user_id"-"ab_test":', participants[['user_id', "ab_test"]].
```

```
Число строк-дубликатов: 0
Число дубликатов "user_id": 1602
Число дубликатов в связке "user_id"-"ab_test": 0
```

Видим 1602 дубликатов в столбце 'user_id' - `пользователи участвуют в обоих тестах одновременно.` На этом этапе фиксируем наличие таких пользователей, детально будем

рассматривать при оценке корректности теста. Строк-дубликатов нет.

Проверим категориальные данные

In [14]:
```
print("Уникальные значения в столбце 'group':",participants['group'].sort_values().unique().t
print("Уникальные значения в столбце 'ab_test':",participants['ab_test'].sort_values().unique
```

```
Уникальные значения в столбце 'group': ['A', 'B']
Уникальные значения в столбце 'ab_test': ['interface_eu_test', 'recommender_system_test']
```

Здесь никаких сюрпризов, `пользователи разбиты на две группы, встречается два теста:` исследуемый ('recommender_system_test) и второй, конкурирующий ('interface_eu_test'). `Посмотрим как распределены пользователи в исследуемом тесте по группам:`

In [15]:
```
participants.query('ab_test  == "recommender_system_test"')['group'].value_counts()
```

Out[15]:
```
A    3824
B    2877
Name: group, dtype: int64
```

`Есть пользователи в обеих группах,` группа A имеет численное преимущество.

Таблица готова к работе

events

In [16]:
```
display(events.head())
events.info()
```

	user_id	event_dt	event_name	details
0	E1BDDCE0DAFA2679	2020-12-07 20:22:03	purchase	99.99
1	7B6452F081F49504	2020-12-07 09:22:53	purchase	9.99
2	9CD9F34546DF254C	2020-12-07 12:59:29	purchase	4.99
3	96F27A054B191457	2020-12-07 04:02:40	purchase	4.99
4	1FD7660FDF94CA1F	2020-12-07 10:15:09	purchase	4.99

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 440317 entries, 0 to 440316
Data columns (total 4 columns):
 #   Column      Non-Null Count   Dtype  
---  ------      --------------   -----  
 0   user_id     440317 non-null  object 
 1   event_dt    440317 non-null  object 
 2   event_name  440317 non-null  object 
 3   details     62740 non-null   float64
dtypes: float64(1), object(3)
memory usage: 13.4+ MB
```

`Таблица включает 440317 наблюдений,` столбец 'details' содержит пропуски, столбец с датой хранит данные с некорректным типом.

- проверим наличие строк-дубликатов
- приведем столбец с датой к корректному типу данных
- добавим столбец 'date' с датой без времени
- выявим временной промежуток
- изучим значения, хранящиеся в столбце 'event_name'
- изучим пропуски в столбце 'details'
- сверим совпдание user_id с одноименным столбцом в таблице 'new_users'

In [17]:
```
print('Число строк-дубликатов:',events.duplicated().sum())
print('Число дубликатов в связке "user_id"-"event_dt":', events[['user_id', "event_dt"]].dupl
```

```
Число строк-дубликатов: 0
Число дубликатов в связке "user_id"-"event_dt": 151233
```

Явных дубликатов не обнаружено, видим большое количество дубликатов в связке id-время события, объясняется тем, что если в браузере у пользователя отключена часть необходимого тесту функционала события могут записываться с одинаковым временем завершения сессии.

In [18]:
```
# приводим столбец с датой к корректному типу, добавляем столбец с датой, выявляем временной
events['event_dt'] = pd.to_datetime(events['event_dt'])
events['date'] = events['event_dt'].astype('datetime64[D]')
print('Минимальная дата:', events['event_dt'].min())
print('Максимальная дата:',events['event_dt'].max())
print('Временной диапазон:', events['event_dt'].max() - events['event_dt'].min())
```

```
Минимальная дата: 2020-12-07 00:00:33
Максимальная дата: 2020-12-30 23:36:33
Временной диапазон: 23 days 23:36:00
```

Минимальныя и максимальная даты вписываются во временой промежуток проведения A/B-теста.

In [19]:
```
print("Уникальные значения в столбце 'event_name':", events['event_name'].unique().tolist())
events['event_name'].value_counts()
```

```
Уникальные значения в столбце 'event_name': ['purchase', 'product_cart', 'product_page', 'log
in']
```

Out[19]:
```
login           189552
product_page    125563
purchase         62740
product_cart     62462
Name: event_name, dtype: int64
```

Всего встречается 4 типа событий. Видим, что событие "просмотр корзины" случается реже, чем "покупка", что сразу наталкивает на мысль, что "product_cart" - необязательное событие перед "purchase", скорее всего, возможна оплата без добавления в корзину. В остальном данные распределены нормально.

Изучим столбец 'details'

In [20]:
```
events['details'].value_counts()
```

Out[20]:
```
4.99      46362
9.99       9530
99.99      5631
499.99     1217
Name: details, dtype: int64
```

Всего 4 вида значений, суммы покупок фиксированные. Посмотрим, по каким типам событий встречаются пропуски в 'details'

In [21]:
```
print("Пропуски в 'details' характерны для событий:", events.query('details.isna()')['event_n
print("Столбец 'details' заполнен для события:", events.query('details.notna()')['event_name'
```

```
Пропуски в 'details' характерны для событий: ['product_cart', 'product_page', 'login']
Столбец 'details' заполнен для события: ['purchase']
```

Все логично, столбец заполнен данными о стоимости покупки, для остальных событий встречаются пропуски.

Таблица готова к работе

Выводы

- обработаны 4 датасета
- встречаются пропущенные значения в столбце 'details' таблицы 'events' - объясняется типом совершенного действия (в столбце заполнены данные только для покупок)
- явных дубликатов нет ни в одном датасете
- встречаются дубликаты в таблице 'events' в связке столбцов "user_id"-"event_dt" - объясняется техническими особенностями
- встречаются дублирующиеся 'user_id' таблице 'participants' - пользователи участвуют в двух тестах одновременно, изучим детальнее при проведении оценки корректности
- в исследуемом тесте есть пользователи из обеих групп
- все даты приведены к корректному типу

Все 4 датасета готовы к дальнейшему анализу

Оценка корректности проведения теста

Оценим корректность проведения теста. Проверим соответствие техническому заданию и общие признаки корректности разбивки на группы. **Определим следующий план оценки корректности:**

1. Проверка аудитории теста:

- пересечение пользователей:
  - с конкурирующим тестом: как распределены участники конкурирующего теста по группам исследуемого теста, корректно ли оставлять их при анализе исследуемого A/B-теста
  - по группам исследуемого теста: нет ли пользователей, входящих в две группы одновременно
- оценка равномерности распределения пользователей по группам

2. Проверка дат:

- первая дата пользователя должна попадать в диапазон даты запуска теста и даты остановки набора новых пользователей
- дата события не должна превышать дату остановки теста
- совпадает ли время проведения теста с маркетинговыми и другими активностями

3. Распределение пользователей по регионам

- Проверка пересечения времени теста с региональными маркетинговыми активностями

4. Все ли пользователи соврешали действия?

5. Лайфтайм

- все ли пользователи успели "прожить" 14 дней?
- есть ли действия, совершенные после 14-го лайфтайма?

6. Оценка оставшихся участников тестов

выполняется после всех предудщих проверок, т.к. возможны исключения некоторых пользователей из теста

- соответствует ли количество участников теста ожидаемому (6000 пользователей)
- соответсвует ли доля пользователей из EU сожидаемой (15%)

7. Характеристика корректности проведения теста

Проверка аудитории теста

Пересечение с конкурирующим тестом

На этапе предобработки данных мы обратили внимание, что 1602 пользователя участвуют в обоих тестах одновременно. Изучим это пересечение внимательнее.

In [22]:
```
# считаем по каждому пользователю количество тестов
test_cnt = participants.groupby('user_id')['ab_test'].nunique().sort_values(ascending=False).
# сохраняем в список пользователей, участвующих в двух тестах
double_test_users = test_cnt.query('ab_test > 1')['user_id'].tolist()
# В 'double_test_competing' сохраняем информацию по таким пользователям, оставляем только кон
double_test_competing = participants.query('ab_test == "interface_eu_test" and user_id in @do

print('Всего пользователей участвующих в тестах:', participants['user_id'].nunique())
print('Пользователей, участвующих в исследуемом тесте:', participants.query('ab_test == "reco
      ['user_id'].nunique())
print('Пользователей, участвующих в обеих тестах одновременно:', len(double_test_users))
```

```
Всего пользователей участвующих в тестах: 16666
Пользователей, участвующих в исследуемом тесте: 6701
Пользователей, участвующих в обеих тестах одновременно: 1602
```

Большинство новых пользователей участвуют в конкурирующем тесте, в исследуемом **"recommender_system_test" участвует всего 6701 пользователь, среди них 1602 человека участвует в двух тестах одновременно.**

Два теста на одних и тех же пользователей в целом проводить не следует, предпочтительнее исключить "пересекающихся" пользователей, но тогда мы потеряем слишком много участников. **Посмотрим как распределены участники двух тестов одновременно по группам конкурирующего теста:**

In [23]:
```
print('Пользователи, участвующих в обоих тестах: распределение по группам конкурирующего тест
double_test_competing['group'].value_counts()
```

Out[23]:
```
Пользователи, участвующих в обоих тестах: распределение по группам конкурирующего теста:
A    819
B    783
Name: group, dtype: int64
```

Пользователей, участвующих в группе А конкурирующего теста можно оставить. В тесте "interface_eu_test" для них не транслировались никакие изменения и результаты конкурирующего теста не должны повлиять на результаты нашего теста.

С пользователями, участвующими в группе В конкурирующего теста, ситуация сложнее. В этом тесте для них транслировались изменения и мы не можем определить что именно влияло на их поведение, соответственно, не можем корректно проанализировать результаты исследуемого теста. **Посмотрим, равномерно ли распределены такие пользователи по группам теста "recommender_system_test"**

In [24]:
```
# собираем пользователей, участвующих в обеих теста и входящих в группу B в конкурирующем тес
# фильтруем по ним ''participants:
double_test_competing_b = double_test_competing.query('group=="B"')['user_id'].tolist()
participants.query('user_id in @double_test_competing_b and ab_test == "recommender_system_te
```

Out[24]:
```
A    439
B    344
Name: group, dtype: int64
```

В группе А таких пользователей больше, однако мы помним, что в исследуемом тесте в принципе численное преимещество за группой А. Единственный способ оценить равномерны ли доли таких участников в обеих группах: **провести z-тест и оценить разницу долей.**

Используем самостоятельно созданную функцию **'get_z_test'** - она пригодится нам в дальнейшем для оценки конверсий. Функция принимает количество "успехов" в обеих группах, количество "попыток" и уровень стат.значимости. **Передадим ей число участников группы В конкурирующего теста, попавших в группы А и В исследуемого теста, а так же общее число участников исследумого теста по группам.**

В нашем случае под "конверсией" понимаем долю участников

In [25]:
```
# подсчитываем количество участников группы B конкурирующего теста в каждой из групп исследуе
competing_b_recommender_A = participants.query('user_id in @double_test_competing_b \
                                                and ab_test == "recommender_system_test" and g
competing_b_recommender_B = participants.query('user_id in @double_test_competing_b \
                                                and ab_test == "recommender_system_test" and g
#подсчитываем общее количество участников по группам исследуемого теста:
recommender_A = participants.query('ab_test == "recommender_system_test" and group=="A"')['us
recommender_B = participants.query('ab_test == "recommender_system_test" and group=="B"')['us
```

- **H0** Участники из группы В конкурирующего теста **равномерно** распределены по группам исследуемого теста (группы равны)
- **H1** Участники из группы В конкурирующего теста **неравномерно** распределены по группам исследуемого теста (группы не равны)

Уровень статистической значимости: 0.01

In [26]:
```
'''Функция, принимает количество успехов и количество попыток двух групп, уровень стат.значим
рассчитывает пропорции успехов в обеих групах, пропорцию успехов в комбинированном датасете,
проводит z-тест,
возвращает средние конверсии для каждой группы, p_value, результат z-теста
'''

def get_z_test(success_1, success_2, total_1, total_2, alpha):
        # пропорция успехов в первой группе:
    p1 = success_1 / total_1
        # пропорция успехов во второй группе:
    p2 = success_2 / total_2

    print(success_1, success_2, total_1, total_2)
        # пропорция успехов в комбинированном датасете:
    p_combined = ((success_1 + success_2)/
                      (total_1 + total_2))
        # разница пропорций в датасетах:
    difference = p1 - p2
        # считаем статистику в ст.отклонениях стандартного нормального распределения
    z_value = difference / mth.sqrt(p_combined * (1 - p_combined) *
                                        (1/total_1 + 1/total_2))
        # задаем стандартное нормальное распределение (среднее 0, ст.отклонение 1)
    distr = st.norm(0, 1)
    p_value = (1 - distr.cdf(abs(z_value)))*2
    print('Конверсия первой группы:', "{0:.2f}%".format(success_1/total_1*100))
    print('Конверсия второй группы:', "{0:.2f}%".format(success_2/total_2*100))
    print('p-value: {}'.format(p_value))
    if (p_value < alpha):
        print('Отвергаем нулевую гипотезу, между группами есть статистически значимые различи
    else:
```

```
        print('Не отвергаем нулевую гипотезу, между группами нет статистически значимых разли
    print()

get_z_test(competing_b_recommender_A, competing_b_recommender_B, recommender_A, recommender_B
```

```
439 344 3824 2877
Конверсия первой группы: 11.48%
Конверсия второй группы: 11.96%
p-value: 0.5475925169519402
Не отвергаем нулевую гипотезу, между группами нет статистически значимых различий
```

Что ж, разница в долях таких пользователей статистически не значима, можем сказать, **что пользователи группы B конкурирующего теста распределены равномерно по группам A и B исследуемого теста.** Оставим пользователей, участвующих в обоих тестах одновременно.

In [27]:
```
testig_participants = participants.query('ab_test == "recommender_system_test"')
print('Пользователей, участвующих в "recommender_system_test":', len(testig_participants))
```

```
Пользователей, участвующих в "recommender_system_test": 6701
```

Пока что мы соответсвуем требованиям ТЗ по ожидаемому количеству участников, однако мы уже можем сформировать первую особенность данных для оценки корректности теста. **Фиксируем:**

Встречаются пользователи, участвующие в двух тестах одновременно.

- **Пользователи из группы A конкурирующего теста не должны повлиять на результаты исследуемого теста** (для них не транслировались изменения в тесте "interface_eu_test")
- **Пользователям из группы B конкурирующего теста изменения в "interface_eu_test" транслировались, но в исследуемом тесте такие пользователи распределены равномерно.**

Проверка на вхождение в обе группы, равномерность распределения по группам

In [28]:
```
if testig_participants['user_id'].duplicated().sum() == 0:
    print("Нет дубликатов в 'user_id', каждый пользователь входит только в одну группу")
else:
    print("Встречаются дубликаты в 'user_id', необходима дополнительная проверка")
print()
print('Распределение пользователей по группам теста:')
testig_participants['group'].value_counts()
```

```
Нет дубликатов в 'user_id', каждый пользователь входит только в одну группу

Распределение пользователей по группам теста:
```

Out[28]:
```
A    3824
B    2877
Name: group, dtype: int64
```

Здесь сюрпризов нет: каждый участник теста входит только в одну группу, но численно пользователи распределены не равномерно. **Фиксируем:**

- каждый участник входит только в одну группу, **нет пересечений по группам**
- **в группе A на 947 участников больше, чем в группе B**

Проверка дат

Основные требования:

- первая дата пользователя должна попадать в диапазон даты запуска теста и даты остановки набора новых пользователей

- дата события не должна превышать дату остановки теста
- время проведения теста не должно совпадать с маркетинговыми и другими активностями

Сохраним все интересующие нас даты в отдельных переменных и приступим к оценке.

In [29]:
```
start_date = dt.datetime.strptime('2020-12-07', "%Y-%m-%d").date()      # начало теста
end_date = dt.datetime.strptime('2021-01-04', "%Y-%m-%d").date()        # окончание теста
end_user_set =  dt.datetime.strptime('2020-12-21', "%Y-%m-%d").date()  # окончание набора нов
```

Пользователи, пришедшие после окончания набора новых пользователей

In [30]:
```
# сохраним в отдельном списке отобранных пользователей:
testig_user = testig_participants['user_id'].tolist()
# в 'testing_new_user' сохраняем только нужные id
testing_new_user = new_users.query('user_id in @testig_user').sort_values(by='first_date')
```

In [31]:
```
print('Минимальная первая дата пользователя', testing_new_user['first_date'].min())
print('Максимальная первая дата пользователя', testing_new_user['first_date'].max())

if testing_new_user['first_date'].min() >= start_date and testing_new_user['first_date'].max(
    print('Все пользователи набраны после старта теста и до дня остановки набора новых пользо
else:
    print('Встречаются пользователи, набранные за пределами ожидаемого временного диапазона')
```

```
Минимальная первая дата пользователя 2020-12-07 00:00:00
Максимальная первая дата пользователя 2020-12-21 00:00:00
Все пользователи набраны после старта теста и до дня остановки набора новых пользователей (вк
лючительно)
```

```
C:\Users\Hp\AppData\Local\Temp\ipykernel_9712\2322478741.py:4: FutureWarning:

Comparison of Timestamp with datetime.date is deprecated in order to match the standard libra
ry behavior. In a future version these will be considered non-comparable. Use 'ts == pd.Times
tamp(date)' or 'ts.date() == date' instead.
```

Тут все хорошо, **фиксируем:**

Все пользователи собраны в нужный период.

События, совершенные до старта теста или после его окончания

In [32]:
```
# оставляем в testing_events только нужных пользователей
testing_events = events.query('user_id in @testig_user')

print('Минимальная первая дата пользователя', testing_events['date'].min())
print('Максимальная первая дата пользователя', testing_events['date'].max())

if testing_events['date'].min() >= start_date and testing_events['date'].max() <= end_date:
    print('Все события совершены после старта теста и до дня его остановки (включительно)')
else:
     print('Встречаются события, совершенные за пределами ожидаемого временного диапазона')
```

```
Минимальная первая дата пользователя 2020-12-07 00:00:00
Максимальная первая дата пользователя 2020-12-30 00:00:00
Все события совершены после старта теста и до дня его остановки (включительно)
```

```
C:\Users\Hp\AppData\Local\Temp\ipykernel_9712\2980710955.py:7: FutureWarning:

Comparison of Timestamp with datetime.date is deprecated in order to match the standard libra
ry behavior. In a future version these will be considered non-comparable. Use 'ts == pd.Times
tamp(date)' or 'ts.date() == date' instead.
```

В целом, все хорошо, но последнее событие совершено до даты остановки теста, **фиксируем:**

- **Выходящих за ожидаемый период событий нет**.
- **Нет событий, произошедших с 31.12 по 04.01**

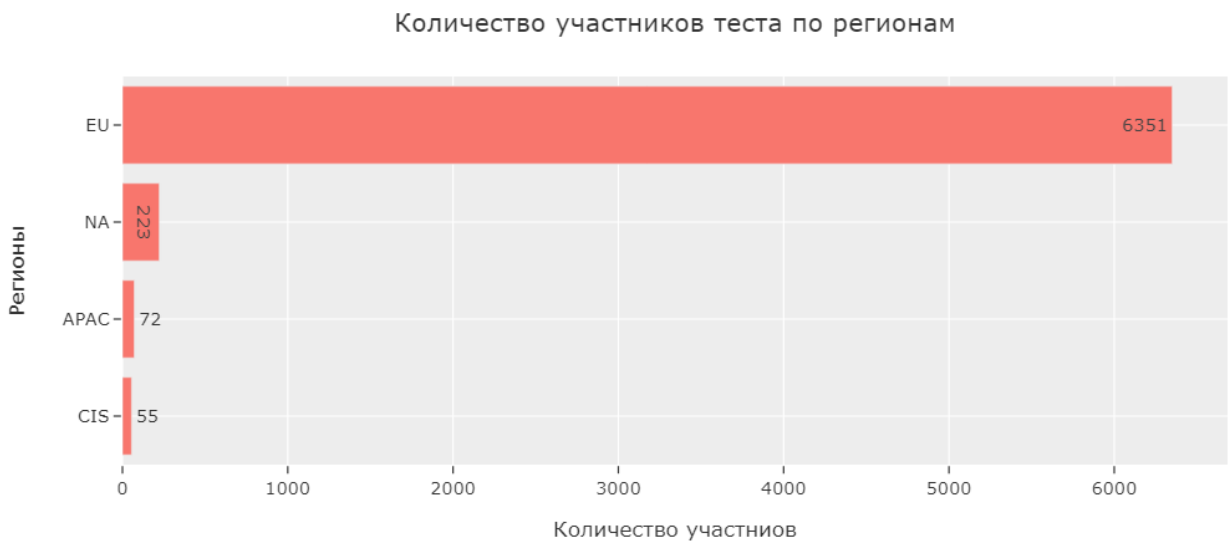
Распределение пользователей по регионам

Как мы помним, маркетинговые акции могут распространяться на разные регионы. `Чтобы понять с какими именно акциями нам нежелательно пересекаться, посмотрим по каким регионам распространены участники теста.`

In [33]:
```
testing_new_user_region = testing_new_user.groupby('region')['user_id'].count().sort_values()
```

In [34]:
```
fig = px.bar(testing_new_user_region,
             y='region', x='user_id', text='user_id', orientation='h')
fig.update_layout(title='Количество участников теста по регионам',
        title_x = 0.5,
        height=400, width=900)
fig.update_yaxes(title_text = "Регионы")
fig.update_xaxes(title_text = "Количество участниов")

fig.show()
```



Большинство пользователей из европейского региона, их количество отвечает требованию ТЗ. Т.к. нас интереесуют изменения конверсий именно европейских пользователей, **удалим из исследования пользователей из других регионов.**

In [35]:
```
# Сохраним новый корректный список пользователей
eu_testing_user = testing_new_user.query('region == "EU"')['user_id'].tolist()
```

In [36]:
```
# Сохраним количество  новых пользователей из EU,
# зарегистрированных в период с даты старта теста до даты окончания набора новых пользовател
total_ue_new_user = new_users.query('region =="EU" and  @end_user_set >= first_date  >= @star
```

In [37]:
```
print('Доля новых пользователей из EU, участвующих в тесте:',
      "{0:.2f}%".format(len(eu_testing_user)/len(total_ue_new_user)*100))
```

```
Доля новых пользователей из EU, участвующих в тесте: 15.00%
```

На этом этапе мы все еще соответствуем ТЗ. **Фиксируем:**

Доля новых пользователей из Европы соответсвует ожидаемому в ТЗ (15%)

Проверка пересечения времени теста с региональными маркетинговыми активностями

Теперь посмотрим на маркетинговые мероприятия в таком ключе:

- распространяется на европейский регион
- дата начала или дата окончания мероприятия больше даты старта теста

In [38]:
```
eu_project = project[project['regions'].str.contains('EU')]
eu_project.query('start_dt > @start_date or finish_dt>@start_date')
```

Out[38]:

	name	regions	start_dt	finish_dt
0	Christmas&New Year Promo	EU, NA	2020-12-25	2021-01-03

Во время проведения теста проходила одна промоакция: "Christmas&New Year Promo". Маркетинговое мероприятие может исказить результаты теста, **крайне нежелательно проводить подобные активности в период проведения тестирования.** Также отметим в принципе не самый удачный выбранный период - в предновогодние дни покупательская активность обычно и так выше, чем в остальные.

Посмотрим, **сколько событий успели совершить пользователи из разных групп во время проведения промо.** Объединим все таблицы в 'eu_data', сохранив только подходящих нам пользователей

In [39]:
```
# сохраняем в eu_testing_participants только пользователей из EU
eu_testing_participants = testig_participants.query('user_id in @eu_testing_user')
```

Таблица с распределением по группам для нас - основная, в первую очередь нам важно сохранить именно ее данные, поэтому все таблицы будем присоединять именно к 'eu_testing_participants' по общему столбцу 'user_id'

In [40]:
```
# присоединяем 'new_users'
eu_data = eu_testing_participants.merge(new_users, on='user_id', how='left')
```

In [41]:
```
# присоединяем 'events'
eu_data = eu_data.merge(events, on='user_id', how='left')
```

In [42]:
```
display(eu_data.head(3))
eu_data.info()
```

	user_id	group	ab_test	first_date	region	device	event_dt	event_name	deta
0	D1ABA3E2887B6A73	A	recommender_system_test	2020-12-07	EU	PC	2020-12-07 14:43:27	purchase	99.
1	D1ABA3E2887B6A73	A	recommender_system_test	2020-12-07	EU	PC	2020-12-25 00:04:56	purchase	4.
2	D1ABA3E2887B6A73	A	recommender_system_test	2020-12-07	EU	PC	2020-12-07 14:43:29	product_cart	Na

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26290 entries, 0 to 26289
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   user_id     26290 non-null  object
 1   group       26290 non-null  object
 2   ab_test     26290 non-null  object
 3   first_date  26290 non-null  datetime64[ns]
 4   region      26290 non-null  object
 5   device      26290 non-null  object
 6   event_dt    23420 non-null  datetime64[ns]
 7   event_name  23420 non-null  object
 8   details     3196 non-null   float64
 9   date        23420 non-null  datetime64[ns]
dtypes: datetime64[ns](3), float64(1), object(6)
```

Собрали всю необходмую информацию по участникам теста в 'eu_data', видим одинаковое количество пропусков в 'event_dt', 'date' и 'event_name' - **не все пользователи совершали действия.** Более детально рассмотрим этот момент в следующем блоке, **сейчас проанализируем активность каждой из групп в период проведения промо, чтобы принять решение о допустимости пересечения теста с промоакцией**

In [43]:
```
eu_data.groupby('date')['event_name'].count().plot(figsize=(10, 5))
plt.grid(True)
plt.xticks(rotation=60)
plt.ylabel('Количество событий')
plt.xlabel('Дата')
plt.title('Количество событий в день по обеим группам');
```

В принципе, **нельзя сказать что промоакция как-то сильно повлияла на количество событий** - покупательский интерес во время проведения акции спадает. Посмотрим, как распределены доли таких событий среди всех остальных событий по группам

In [44]:
```
# разбиваем пользователей на две группы
eu_a = eu_data.query('group == "A"')
eu_b = eu_data.query('group == "B"')

# оставляем только действия совершенные в период акции по обеим группам
```

```
promo_a = eu_a.query('"2021-01-03" >= date >="2020-12-25"')
promo_b = eu_b.query('"2021-01-03" >= date >="2020-12-25"')

promo_a_event = promo_a['event_name'].count()
promo_b_event = promo_b['event_name'].count()
total_a_event = eu_a['event_name'].count()
total_b_event = eu_b['event_name'].count()

print('Группа A')
print('Всего событий:', total_a_event)
print('Событий в период проведения акции:', promo_a_event, ',', "{0:.2f}%".format(promo_a_eve
      'от общего числа')
print('Группа B')
print('Всего событий:', total_b_event)
print('Событий в период проведения акции:', promo_b_event, ',', "{0:.2f}%".format(promo_b_eve
      'от общего числа')
```

```
Группа A
Всего событий: 18309
Событий в период проведения акции: 2658 , 14.52% от общего числа
Группа B
Всего событий: 5111
Событий в период проведения акции: 506 , 9.90% от общего числа
```

А вот пропорциаонально такие события распределены неравномерно. Однако если акция транслируется обеим группам одновременно, то не стоит исключать период проведения промоакции из анализа. **Фиксируем:**

- **Во время проведения теста осуществлялась маркетинговая активность**
  - Глобально проведение акции не повлияло на покупательскую активность
  - Количество действий, осещественных в период промо, неравномерно распределено по группам

Все ли пользователи осуществляли действия?

В предыдущем блоке мы обратили внимание, что есть пропуски в столбцах, связанных с действиями, **значит, не все участники теста совершали действия.** Изучим этим пропуски.

In [45]:
```
without_event = eu_data.query('event_name.isna()')
print('Пользователей, не совершавших действий:', len(without_event), ',',
      "{0:.2f}%".format(len(without_event)/eu_data['user_id'].nunique()*100))
```

```
Пользователей, не совершавших действий: 2870 , 45.19%
```

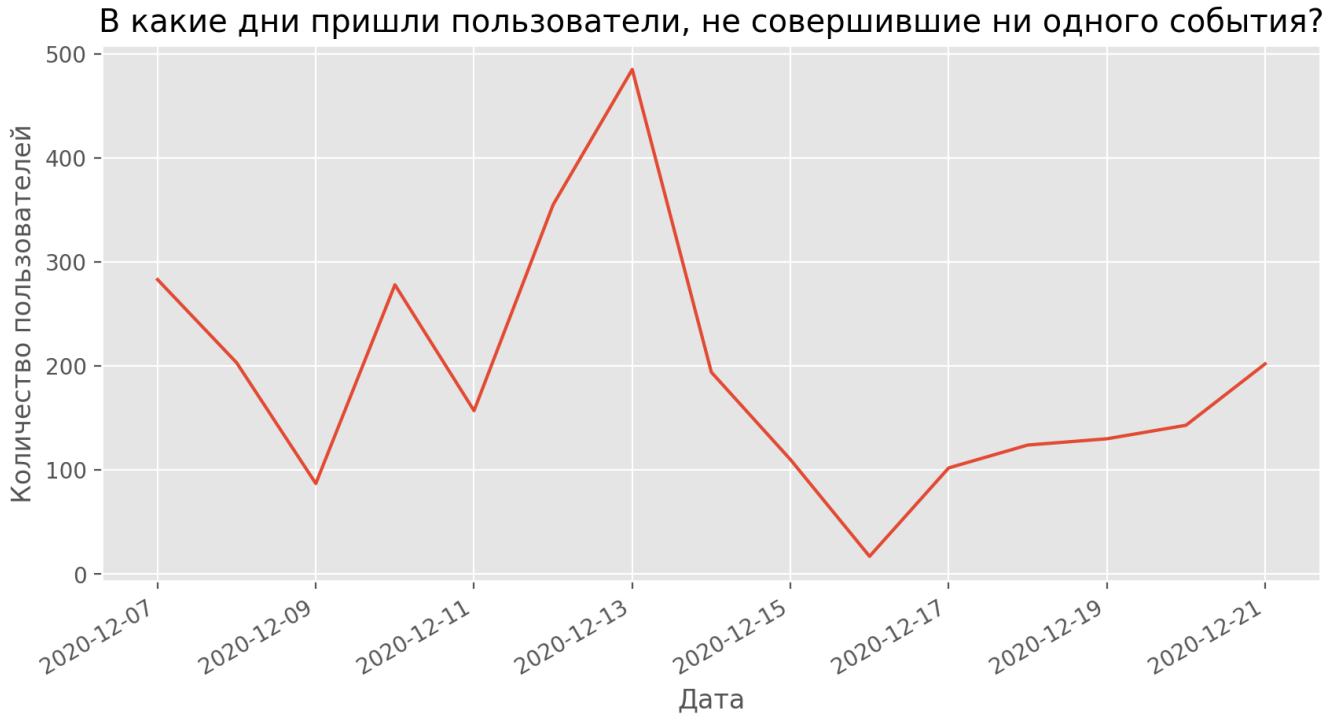
Чуть менее половины пользователей, отобранных для теста не совершали никаких действий. Посмотрим, на них внимательнее

In [46]:
```
print('Пользователи, не совершавшие действий, по группам:')
without_event['group'].value_counts()
```

Out[46]:
```
Пользователи, не совершавшие действий, по группам:
B    1840
A    1030
Name: group, dtype: int64
```

Большинство из таких пользователей относятся к группе B, что очень плохо - группа и так меньше контрольной.

In [47]:
```
without_event['first_date'].value_counts().plot(figsize=(10, 5))
plt.ylabel('Количество пользователей')
plt.xlabel('Дата')
plt.title('В какие дни пришли пользователи, не совершившие ни одного события?');
```



Большинство из таких пользователей пришли к нам 13.12. Встречаются и более ранние даты, т.е. н**ельзя сказать, что пользователи пришли к концу теста и просто не успели совершить никаких действий.**

Пользователям из группы B транслировались некие изменения - могли ли внедренные изменения спровоцировать какие-то технические ошибки для определенного типа устройства?

In [48]:
```
without_event_B = without_event.query('group == "B"')
print('Пользователи, не совершавшие действий, распределение по устройствам:')
without_event_B['device'].value_counts()
```

Out[48]:
```
Пользователи, не совершавшие действий, распределение по устройствам:
Android    823
PC         445
iPhone     396
Mac        176
Name: device, dtype: int64
```

Большинство таких пользователей использует Android, как и большинство всех остальных пользователей, так что проблему здесь не выявить. Никаких иных данных по пользователям у нас нет.

Посмотрим, в какие дни пришли пользователи, не совершившие никаких действий

In [49]:
```
without_event_A = without_event.query('group == "A"')
without_event_A_day_cnt = without_event_A.groupby('first_date')['user_id'].nunique().reset_in
without_event_B_day_cnt = without_event_B.groupby('first_date')['user_id'].nunique().reset_in
```

In [50]:
```
fig = go.Figure()
fig.add_trace(go.Scatter(x=without_event_A_day_cnt['first_date'], y=without_event_A_day_cnt['
                    mode='lines+markers',
                    name='A'))
fig.add_trace(go.Scatter(x=without_event_B_day_cnt['first_date'], y=without_event_B_day_cnt['
                    mode='lines+markers',
                    name='B'))
fig.update_layout(
    title="Количество действий в день по группам",
    xaxis_title="Дата",
    yaxis_title="Количество действий",
    xaxis = dict(tickvals =without_event_B_day_cnt['first_date']),
    legend=dict(yanchor="top", y=0.9, xanchor="left", x=0.9),
```

```
    margin=dict(l=0, r=0, t=30, b=0))
fig.show()
```

Видим неравномерно распределение таких пользоватлей по группам - у группы А график обрывается на 13.12, у группы В продолжают набираться пользователи, не совершавшие действий. Вероятно, есть ошибки в логировании или иные проблемы с данными или с использованием приложения, в любом случае, распределение по группам неравномерно, **к сожалению, мы вынуждены исключить таких пользователей, т.к. они никак не участвовали в тесте.**

In [51]:
```
eu_data = eu_data.query('event_name.notna()')
print('Всего участников теста:', eu_data['user_id'].nunique())
print('Из группы A:', eu_data.query('group == "A"')['user_id'].nunique())
print('Из группы B:', eu_data.query('group == "B"')['user_id'].nunique())
```

```
Всего участников теста: 3481
Из группы A: 2604
Из группы B: 877
```

Сильно отошли от ожиданий ТЗ. **Фиксируем:**

- **45% участников не совершили ни одного действий.** Особенностей среди участников не обнаружено
- Количество участников теста сократилось до **3481** человека, группа А почти в три раза численно превосходит группу В

Ограничение действий по лайфтайму

Обратимся к разъяснению ТЗ по поводу лайфтайма:

- **Пользователи должны иметь возможность совершать события в течении двух недель.**

Т.е. нас интересуют только пользователи успевшие "прожить" 2 недели до даты окончания теста

- **Мы должны отфильтровать события, которые пользователи совершили после 14 дней от регистрации.**

Так как нас интересует улучшение метрики только в течении 14 дней от регистрации, мы должны отбросить события, которые произошли после 14 дней от того момента, как был зарегистрирован пользователь, который их совершил. Нам необходимо, чтобы лайфтайм событий, которые будут влиять на наш тест, укладывался в установленный горизонт событий.

In [52]:
```
# рассчитываем лайфтайм для каждого события
eu_data['lifetime'] = (eu_data['date'] - eu_data['first_date']).dt.days
```

In [53]:
```
# срезаем события, совершенные после 14-го лайфтайма
eu_data = eu_data.query('lifetime <= 14')
```

In [54]:
```
print('Максимальнаая дата регистрации:', eu_data['first_date'].max())
```

```
Максимальнаая дата регистрации: 2020-12-21 00:00:00
```

Минимальная дата регистрации для нас подходит - **все пользователи пришли за две недели до окончания теста**

Оценка оставшихся пользователей

In [55]:
```
print('Всего участников теста:', eu_data['user_id'].nunique())
print('Количество совершенных действий', len(eu_data))
print('Из группы A:', eu_data.query('group == "A"')['user_id'].nunique())
print('Из группы B:', eu_data.query('group == "B"')['user_id'].nunique())
print('Отобрано новых клиентов из EU:',
      "{0:.2f}%".format(eu_data['user_id'].nunique()/len(total_ue_new_user)*100))
```

```
Всего участников теста: 3481
Количество совершенных действий 22828
Из группы A: 2604
Из группы B: 877
Отобрано новых клиентов из EU: 8.22%
```

Набор данных включал в себя большое количество пользователей, никак не участвующих в тесте (пользователи не совершали даже 'login', т.е. вообще не открывали приложение). **Исключив их, сильно отошли от ожидаемого количества участников.**

Выводы

Соберем все, что мы зафискировали в течении оценки корректности проведения теста

Негативные детали тестирования

1. Встречаются пользователи, участвующие в двух тестах одновременно.

Проанализировали распределение по группам конкурирующего и исследуемого теста, оставили таких пользователей:

- Пользователи из группы A конкурирующего теста **не должны повиять на результаты исслеуемого теста** (для них не транслировались изменения в тесте "interface_eu_test")
- Пользователям из группы B конкурирующего теста изменения в "interface_eu_test" транслировались, **но в исследуемом тесте такие пользователи распределены равномерно.**

2. Во время теста осуществлялась маркетинговая активность

Глобально проведение акции не повлияло на покупательскую активность (наблюдаем спад действий в период промо)

- допущено пересечение с маркетинговой акцией, т.к. **она транслируется обеим группам**
- количество действий, осещественных в период промо, **неравномерно распределено по группам**

3. Есть пользователи, не совершившие ни одного действия

У групп в неравных долях наблюдается набор пользователей, не совершавших действий. У групп А набор таких пользоватлей остановился 15.12, у группы В продолжался до конца периода.

- **45% участников не совершали действий**
- такие участники исключены

4. Количество участников не соответствует ожидаемым

- количество участников сократилось до **3481**, что на 2519 человек меньше ожидаемого
- всего отобрано **8.22% новых клиентов из EU**

5. Участники между группами распределены неравномерно

- 2604 входят в группу А
- 877 входят в группу В

На основании вышесказанного, можем утверждать, что **тест провден некорректно.**

Резльтаты A/B-тестирования могут быть искажены под влиянием всех 5 пунктов

Из положительных деталей:

- каждый участник входит только в одну группу, **нет пересечений по группам**
- Все пользователи собраны в нужный период
- Выходящих за период проведения теста событий нет
- огранили число событий на основании лайфтайма (оставили только те, что были совершены до 14-го дня с момента регистрации) без потери уникальных участников

Исследовательский анализ

Здесь рассмотрим следующие вопросы:

- Количество событий на пользователя одинаково распределены в выборках?
- Как число событий в выборках распределено по дням?
- Как меняется конверсия в воронке в выборках на разных этапах?
- Какие особенности данных нужно учесть, прежде чем приступать к А/Втестированию?

In [56]:
```
# сохраним данные по группам в отдельные df
a_group = eu_data.query('group == "A"')
b_group = eu_data.query('group == "B"')
```

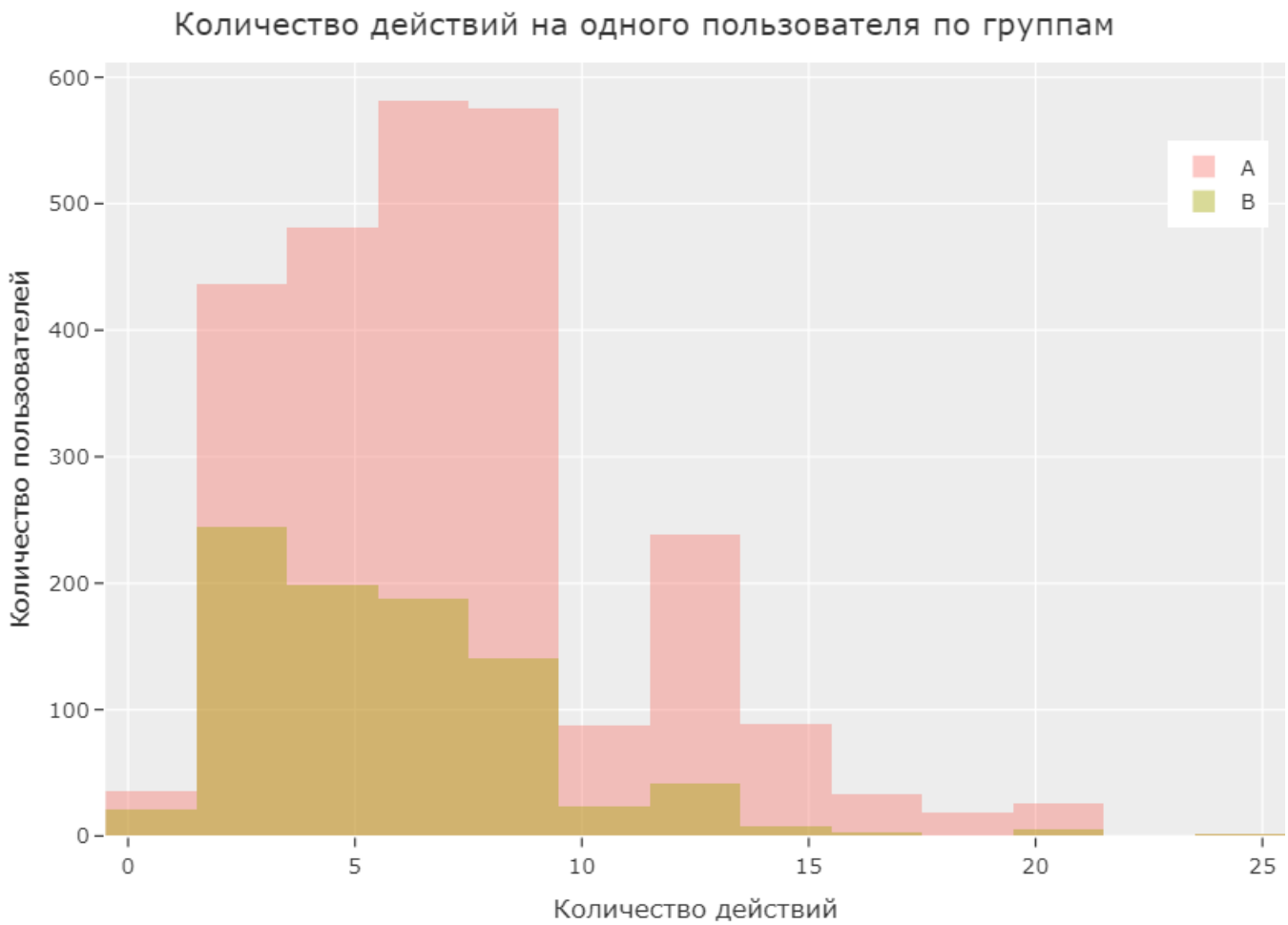
Распределение количества событий на пользователя

In [57]:
```
# считаем количество событий по пользователям в каждой из групп
a_per_user = a_group.groupby('user_id')['event_name'].count().reset_index()
b_per_user = b_group.groupby('user_id')['event_name'].count().reset_index()
```

In [58]:
```
fig = go.Figure()
fig.add_trace(go.Histogram(x=a_per_user['event_name'],

                           opacity=0.4,
                           name='A',nbinsx=20))
fig.add_trace(go.Histogram(x=b_per_user['event_name'],

                           opacity=0.4, name='B',nbinsx=20))
fig.update_layout(
    title="Количество действий на одного пользователя по группам",
    xaxis_title="Количество действий",
    yaxis_title="Количество пользователей",
    legend=dict(yanchor="top", y=0.9, xanchor="left", x=0.9),
    barmode='overlay',
    margin=dict(l=0, r=0, t=30, b=0))
fig.show()
```



Помня о том, что у группы А численное преимущество, **количество действий на пользователя распределено достаточно равномерно.**

У группы В крайне редко встречаются пользователи, совершившие более 15 действий. Посмотрим на средние значения по группам:

In [59]:
```
print('Среднее количество действий на пользователя:')
print('Группа A:', round(a_per_user['event_name'].mean(),2))
print('Группа B:',round(b_per_user['event_name'].mean(),2))
```

```
Среднее количество действий на пользователя:
Группа A: 6.9
Группа B: 5.53
```

Усредненные значения отличаются. **Фиксируем:**

- **Между группами есть различия в среднем количестве действий на пользователя, что может повлиять на результаты теста теста**

Число событий по дням

In [60]:
```
# считаем количество событий по дням в каждой из групп
a_per_day = a_group.groupby('date')['event_name'].count().reset_index()
b_per_day = b_group.groupby('date')['event_name'].count().reset_index()
```

In [61]:
```
fig = go.Figure()
fig.add_trace(go.Scatter(x=a_per_day['date'], y=a_per_day['event_name'],
                    mode='lines+markers',
                    name='A'))
fig.add_trace(go.Scatter(x=b_per_day['date'], y=b_per_day['event_name'],
                    mode='lines+markers',
                    name='B'))
fig.update_layout(
    title="Количество действий в день по группам",
    xaxis_title="Дата",
    yaxis_title="Количество действий",
    xaxis = dict(tickvals =a_per_day['date']),
    legend=dict(yanchor="top", y=0.9, xanchor="left", x=0.9),
    margin=dict(l=0, r=0, t=30, b=0))
fig.show()
```

Количество действий в день по группам сильно различается, что ожиданио: группа А имеет численное превосходство в пользователях.

- динамика действий у обеих групп схожая: рост и спад числа действий по группам приходится на одни и те же дни (за редкими исключениями)

В целом, такой график малоинформативен, **посмотрим на среднее количество событий в день на одного пользователя по группам**

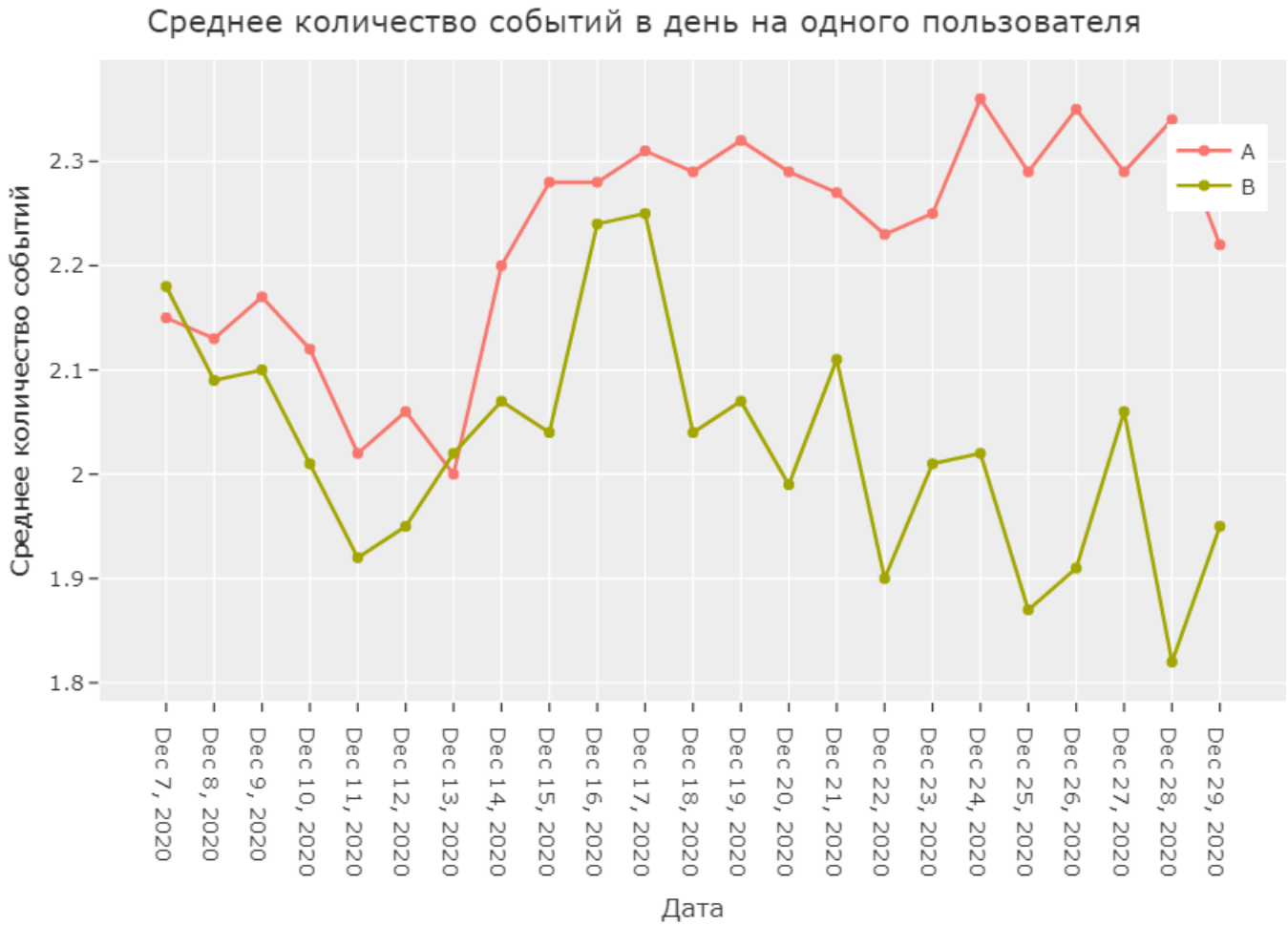
In [62]:
```
# для каждой группы считаем кол-во уникальных пользователей, кол-во событий, рассчитываем сре
a_mean_per_day = a_group.groupby('date').agg({'user_id':'nunique', 'event_name':'count'}).res
a_mean_per_day['ratio'] = round(a_mean_per_day['event_name'] / a_mean_per_day['user_id'], 2)
```

```
b_mean_per_day = b_group.groupby('date').agg({'user_id':'nunique', 'event_name':'count'}).res
b_mean_per_day['ratio'] = round(b_mean_per_day['event_name'] / b_mean_per_day['user_id'], 2)
```

In [63]:
```
fig = go.Figure()
fig.add_trace(go.Scatter(x=a_mean_per_day['date'], y=a_mean_per_day['ratio'],
                         mode='lines+markers',
                         name='A'))

fig.add_trace(go.Scatter(x=b_mean_per_day['date'], y=b_mean_per_day['ratio'],
                         mode='lines+markers',
                         name='B'))

fig.update_layout(
    title="Среднее количество событий в день на одного пользователя",
    xaxis_title="Дата",
    yaxis_title="Среднее количество событий",
    xaxis = dict(tickvals =a_per_day['date']),
    legend=dict(yanchor="top", y=0.9, xanchor="left", x=0.9),
    margin=dict(l=0, r=0, t=30, b=0))
fig.show()
```



За исключением нескольких пересечний, **у группы А среднее количество событий на пользователя стабильно выше**

Как меняется конверсия в воронке в выборках на разных этапах

Сначала построим таблицу для каждой группы и оценим сколько раз произошло каждое из действий.

In [64]:
```
a_funnel = a_group.groupby('event_name')['user_id'].nunique().sort_values(ascending=False).re
a_funnel
```

Out[64]:

	event_name	user_id
0	login	2604
1	product_page	1685
2	purchase	833
3	product_cart	782

Видим, что ожидаемая воронка не соблюдается: событие **'purchase' происходит чаще, чем событие 'product_cart'**. Мы еже обращали внимание на такое распределение числа действий на этапе предобработки: **видимо, есть возможность совершить покупку, не заходя в корзину.**

Для графика воронки расположим события в ожидаемом порядке: "зашел - открыл карточку - добавил в корзину - купил"

In [65]:
```
a_funnel = a_funnel.reindex([0,1,3,2])
```

In [66]:
```
b_funnel = b_group.groupby('event_name')['user_id'].nunique().sort_values(ascending=False).re
b_funnel
```

Out[66]:

	event_name	user_id
0	login	876
1	product_page	493
2	purchase	249
3	product_cart	244

У группы B ситуация аналогичная. `Построим графики для обеих групп.`

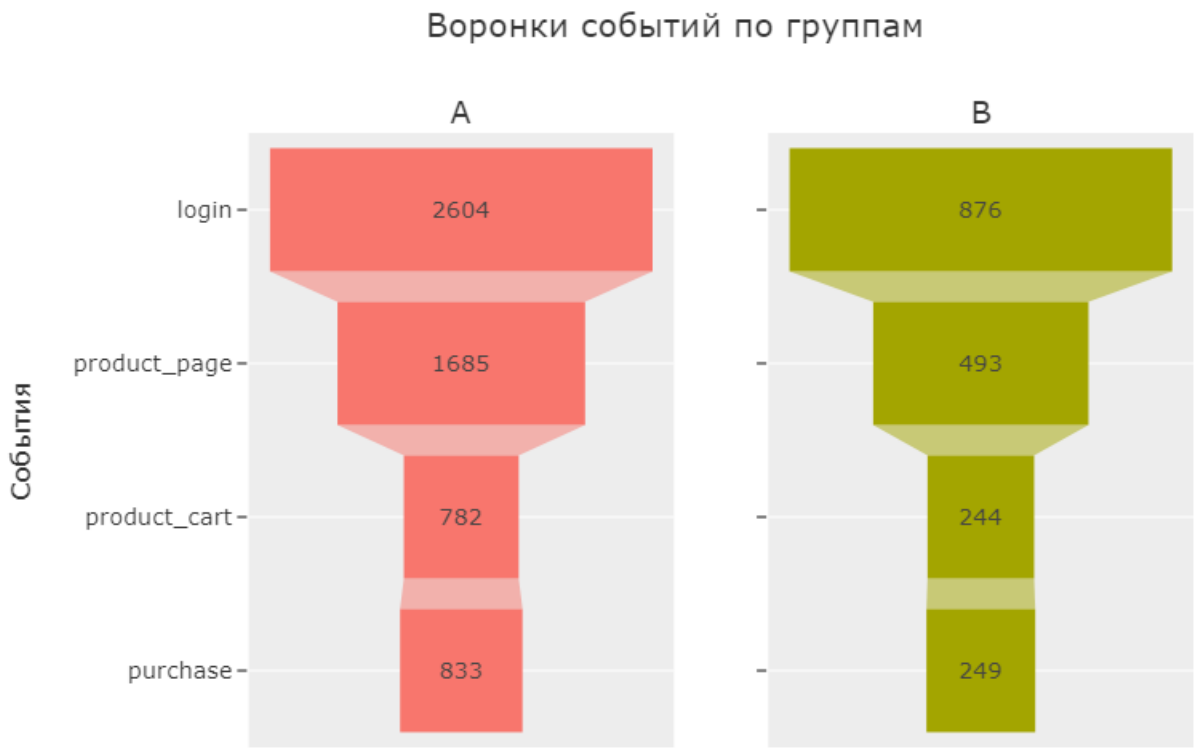
In [67]:
```
b_funnel = b_funnel.reindex([0,1,3,2])
```

In [68]:
```
fig = make_subplots(rows=1, cols=2, subplot_titles=['A', 'B'], shared_yaxes=True)

trace0 = go.Funnel(y=a_funnel['event_name'], x=a_funnel['user_id'])
trace1 = go.Funnel(y=b_funnel['event_name'], x=b_funnel['user_id'])
fig.update_layout(title='Воронки событий по группам', showlegend=False)

fig.append_trace(trace0, 1, 1)
fig.append_trace(trace1, 1, 2)
fig.update_yaxes(title='События', col=1, row=1)
fig.update_xaxes(title='Средняя стоимость', col=1, row=1)

fig.update_xaxes(title='Средняя стоимость', col=2, row=1)
fig.update_layout(showlegend=False)
fig.show()
```



Рассмотрим события 'product_cart' и 'purchase' вне зависимость друг от друга, т.к. мы уже определили. что покупка может совершаться без участия корзины.

Группа A

- карточку продукта открывают 64.7%
- в корзину из карточки добавляют 46.4% или 13.2% от всех пользователей
- покупку из просмотра карточки (с возможным заходом в корзину) осуществляют 49.4% или 14.1% от всех пользователей

Группа B

- карточку продукта открывают 56.3%
- в корзину из карточки добавляют 49.5% или 13.1% от всех пользователей
- покупку из просмотра карточки (с возможным заходом в корзину) осуществляют 50.5% или 13.4% от всех пользователей

Группы различаются по конверсиям

- Участники группы A чаще открывают карточки
- В корзину из карточки чаще добавляют участники группы B
- После просмотра карточки покупку чаще осуществляют участники группы B (но нельзя оценить, заходили ли они в корзину перед покупкой)
- Конверсия в покупку выше у группы A

Кумулятивные конверсии

Оценим кумулятивные конверсии в каждое из целевых действий у обеих групп.

- получим кумулятивные значения уникальных посетителей и пользователей, совершивших действия
- посмотрим как они отличаются в динамике
- посмотрим, наблюдается ли относительный прирост у группы B над A

Сможем оценить период с начала теста до 21.12, т.к. после этой даты новых пользователей не набиралось

In [69]:
```
'''Функция, принимает таблицу, названия событий
фильтрует таблицу по событиям, возвращает три отфильтрованные таблицы'''
def make_query(df, event_1, event_2, event_3):
    filtred_1 = df.query(f'event_name == "{event_1}"')
    filtred_2 = df.query(f'event_name == "{event_2}"')
    filtred_3 = df.query(f'event_name == "{event_3}"')
    return filtred_1, filtred_2, filtred_3
```

In [70]:
```
product_page, product_cart, purchase = make_query(eu_data, 'product_page', 'product_cart', 'p
```

In [71]:
```
# создаем массив уникальных пар значений дат и групп теста
datesGroups = eu_data[['date','group']].drop_duplicates()
```

In [72]:
```
# получаем агрегированные кумулятивные по дням данные о посетителях
visitors_aggregated = datesGroups.apply(lambda x: eu_data[np.logical_and(eu_data['date'] <= x
                                                                         eu_data['group'] ==
                                        .agg({'date' : 'max',
                                              'group' : 'max',
                                              'user_id' : 'nunique'}), axis=1).sort_values(by=
```

In [73]:
```
'''Функция, принимает таблицу,
собирает аггрегигованные данные о количестве совершивгих событие пользоваелях,
возвращает аггрегированную таблицу'''
def get_aggregated_df(df):
    event_aggrerated = datesGroups.apply(lambda x: df[np.logical_and(df['date'] <= x['date'],
                                                                     df['group'] == x['group'
                                       .agg({'date' : 'max',
                                             'group' : 'max',
                                             'user_id' : 'nunique'}), axis=1).sort_values(by=['
    return event_aggrerated
```

In [74]:
```
# применяем get_aggregated_df к отфильтрованным ранее таблицам:
product_page_aggregated = get_aggregated_df(product_page)
product_cart_aggregated = get_aggregated_df(product_cart)
purchase_aggregated = get_aggregated_df(purchase)
```

In [75]:
```
#переименовываем столбцы в полученных таблицах:
product_page_aggregated = product_page_aggregated.rename(columns={'user_id':'product_page'})
product_cart_aggregated = product_cart_aggregated.rename(columns={'user_id':'product_cart'})
purchase_aggregated = purchase_aggregated.rename(columns={'user_id':'purchase'})
```

Наконец, создаем таблицу с кумулятивными значениями в 'eu_cumulative' объединим таблицу с визитами и таблицы с аггрегированными данными по каждому ключевому событию.

In [76]:
```
# присоединем 'product_page_aggregated'
eu_cumulative = visitors_aggregated.merge(product_page_aggregated, how='left', on=['date', 'g
```

In [77]:
```
# присоединем 'product_cart_aggregated'
eu_cumulative = eu_cumulative.merge(product_cart_aggregated, how='left', on=['date', 'group']
```

In [78]:
```
# присоединям 'purchase_aggregated':
eu_cumulative = eu_cumulative.merge(purchase_aggregated, how='left', on=['date', 'group'])
```

In [79]:
```
eu_cumulative = eu_cumulative.rename(columns={'user_id':'total'})
#срезаем даты до горизонта анализа:
eu_cumulative = eu_cumulative.query('date < "2020-12-22"')
# рассчитываем кумулятивную конверсию по каждому событию:
eu_cumulative['product_page_cr'] = round(eu_cumulative['product_page'] / eu_cumulative['total
eu_cumulative['product_cart_cr'] = round(eu_cumulative['product_cart'] / eu_cumulative['total
eu_cumulative['purchase_cr'] = round(eu_cumulative['purchase'] / eu_cumulative['total'], 2)
```

In [80]:
```
# создаем отдельные таблицы для каждой из групп:
a_cumulative = eu_cumulative.query('group == "A"')
b_cumulative = eu_cumulative.query('group == "B"')
```

In [81]:
```
# создаем общую таблицу для возможности рассчета относительного прироста:
merge_cumulative = a_cumulative[['date', 'product_page_cr', 'product_cart_cr', 'purchase_cr']
    .merge(b_cumulative[['date', 'product_page_cr', 'product_cart_cr', 'purchase_cr']], left_
           right_on='date', how='left', suffixes=['_A', '_B'])
```

In [82]:
```
# рассчитываем относительный прирост группы B над A по всем конверсиям
merge_cumulative['relative_product_page'] = merge_cumulative['product_page_cr_B'] / merge_cum
merge_cumulative['relative_product_cart'] = merge_cumulative['product_cart_cr_B'] / merge_cum
merge_cumulative['relative_purchase'] = merge_cumulative['purchase_cr_B'] / merge_cumulative[
```

Собрали в 'eu_cumulative' кумулятивные конверсий по всем типам событий, а в 'merge_cumulative' данные об относительном приросте каждой из конверсий. Посмотрим на отличия групп.

In [83]:
```
'''Функция, принимает названия столбцов и название конверсии для заголовка,
отрисовывает графики кумулятивной конверсии по группам и график относительного прироста'''

def make_scatter(col_1, col_2, cr_name):
    fig = go.Figure()
    fig.add_trace(go.Scatter(x=a_cumulative['date'], y=a_cumulative[col_1],
                    mode='lines+markers',
                    name='A'))
    fig.add_trace(go.Scatter(x=b_cumulative['date'], y=b_cumulative[col_1],
                    mode='lines+markers',
                    name='B'))
    fig.update_layout(
        height=300, width=900,
        title=f"Кумулятивная конверсия в {cr_name}",
        xaxis_title="Дата",
        yaxis_title="Конверсия",
        xaxis = dict(tickvals =a_cumulative['date']),
        legend=dict(yanchor="top", y=0.9, xanchor="left", x=0.9),
        margin=dict(l=0, r=0, t=30, b=0))
    fig.show()

    fig = px.line(x=merge_cumulative['date'], y=merge_cumulative[col_2])
    fig.add_hline(y=0, line_width=3, line_dash="dash", line_color="green")
    fig.update_layout(
        height=300, width=900,
        title= f"Относительное изменение кумулятивной конверсии в {cr_name} группы B к группе
        xaxis_title="Дата",
        yaxis_title="Относительный прирост",
        xaxis = dict(tickvals = a_cumulative['date']),
        legend=dict(yanchor="top", y=0.9, xanchor="left", x=0.9),
        margin=dict(l=0, r=0, t=30, b=0))

    fig.show()
```

In [84]:
```
make_scatter("product_page_cr", "relative_product_page", '"product_page"')
```

Конверсия в просмотр карточки:

- Группа B весь стабильно и значительно показывает результат хуже, чем группа A
- **Улучшение метрики у группы B относительно группы A не наблюдается.** В период с 18.12 по 21.12 относительное падение конверсии у группы по отношению к A составляет более 12%

In [85]:
```
make_scatter("product_cart_cr", "relative_product_cart", '"product_cart"')
```

Конверсия в добавление в корзину:

- Начиная с 9.12 до конца периода конверсия группы B стабильно ниже, чем конверсия группы A
- **Улучшения метрики у группы B по отношению к группе A начиная с 9.12 не наблюдается.** Под конец периода группа B отстает от группы A на ~6.5%

In [86]:
```
make_scatter("purchase_cr", "relative_purchase", '"purchase"')
```

Конверсия в покупку:

- До 15.12 группа B демонстрировала конверсию выше, чем группа A
- В первой половине наблюдаем относительный прирост конверсии группы B более, чем на 10%. **К концу периода конверсия группы B относительно ниже.**

Итого: в период 7.12-21-12 конверсии в просмотр карточки и добавление в корзину у группы стабильно ниже. Относительный прирост в конверсии в покупку наблюдается только в первой половине периода, далее конверсия группы B проигрывает группе A.

Доли целевых действий в каждый из лайфтаймов

Проанаоизируем лайфтаймы. Посмотрим, **сколько пользователей (в долях) из каждой группы совершает целевые действия в каждый из лайфтаймов** - так мы сможем оценить, какая группа обычно быстрее совершает одно из целевых действий.

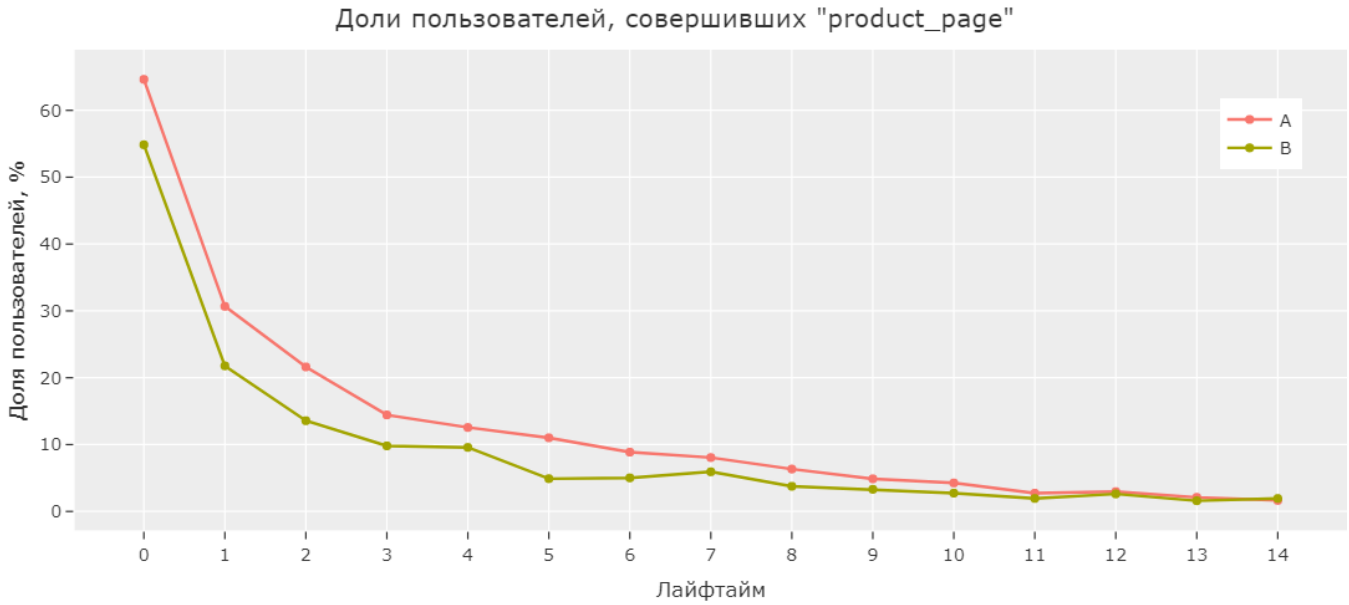
In [87]:
```
'''Функция, принимает таблицу и названия события
фильтрует таблицу по событию, группирует по лайфтайму, подсчитывает количество уникальных пол
рассчитывает долю, возвращает отфильтрованную таблицу'''
def make_lifetime_event(df, event):
    group_event = df.query(f'event_name == "{event}"')
    lifetime_event = group_event.groupby('lifetime')['user_id'].nunique().reset_index()
    lifetime_event['ratio'] = round(lifetime_event['user_id'] / df['user_id'].nunique()*100,2
    return lifetime_event
```

In [88]:
```
a_lifetime_pp = make_lifetime_event(a_group, 'product_page')
a_lifetime_pc = make_lifetime_event(a_group, 'product_cart')
a_lifetime_pur = make_lifetime_event(a_group, 'purchase')

b_lifetime_pp = make_lifetime_event(b_group, 'product_page')
b_lifetime_pc = make_lifetime_event(b_group, 'product_cart')
b_lifetime_pur = make_lifetime_event(b_group, 'purchase')
```

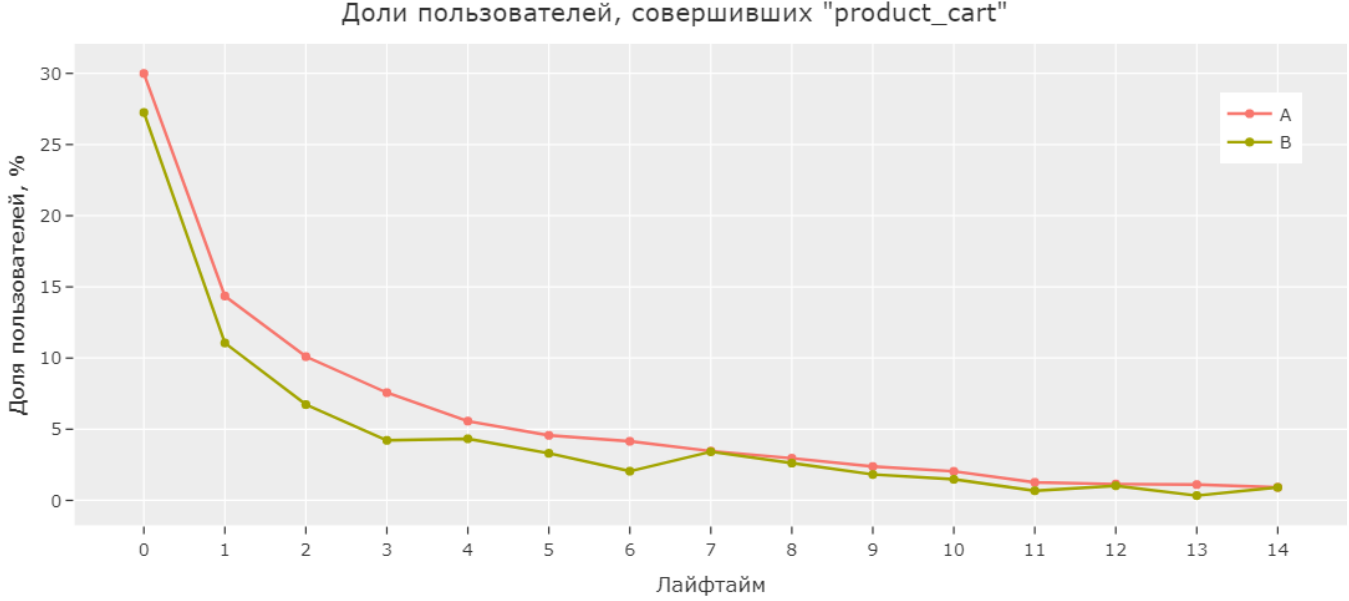
In [89]:
```
'''Функция, принимает две таблицы и название события,
отрисовывает графики'''
def make_lifetime_scatter (df_1,df_2, event):
    fig = go.Figure()
    fig.add_trace(go.Scatter(x = df_1['lifetime'], y = df_1['ratio'], name='A'))
    fig.add_trace(go.Scatter(x = df_2['lifetime'], y = df_2['ratio'], name='B'))
    fig.update_layout(
        height=400, width=900,
        title= f'Доли пользователей, совершивших "{event}"',
        xaxis_title="Лайфтайм",
        yaxis_title="Доля пользователей, %",
        xaxis = dict(tickvals = a_lifetime_pp['lifetime']),
        legend=dict(yanchor="top", y=0.9, xanchor="left", x=0.9),
        margin=dict(l=0, r=0, t=30, b=0))
    fig.show()
```

In [90]:
```
make_lifetime_scatter(a_lifetime_pp, b_lifetime_pp, 'product_page')
```



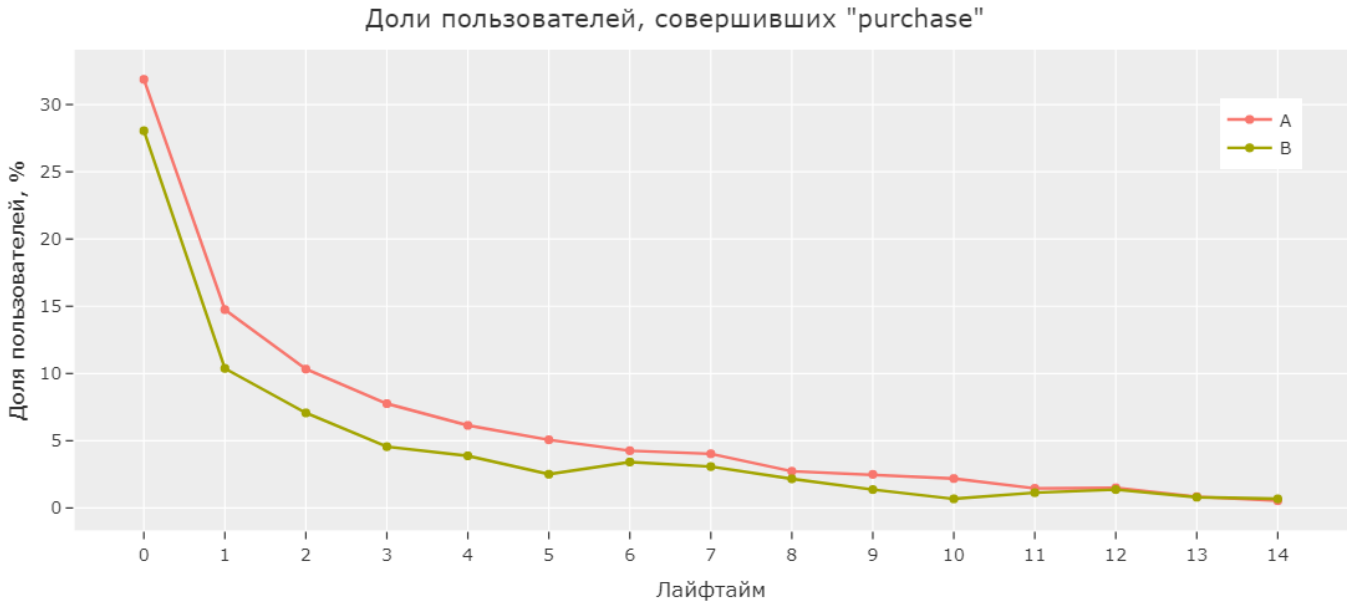
- **Пользователи обоих групп чаще всего совершают просмотр карточек в первый же день** (в нулевой лайфтайм)
- **Пользователи из группы А в первый день совершают события чаще,** чем пользователи из группы В
- Группа В в целом менее активна, уже на первый лайфтайм 78% пользователей перестают открывать карточки

In [91]:
```
make_lifetime_scatter(a_lifetime_pc, b_lifetime_pc, 'product_cart')
```



- Доли пользователей группы B стабильно ниже (за исключением нескольких пересечений)
- Пользователи обеих групп чаще всего совершают добавление в корзину в первый же день, **однако у пользоваталей группы A доля таких пользователей выше**
- Доли обеих групп практически сравниваются на 7-й лайфтайм, однако группа **A демонстрирует заметно лучший результат на 1-3 лайфтаймы**

In [92]:
```
make_lifetime_scatter(a_lifetime_pur, b_lifetime_pur, 'purchase')
```



- снова **преимущество за группой A** - почти 32% совершают первую покупку в первый день, у группы B конверсия 28%.
- группа A показывает результат выше в 0-11 лайфтаймы, **в начале "жизни" (0-6 лайфтаймы) положительный отрыв от группы B очевиден**

Выводы

Наблюдается преимущества группы A над группой B:

- выше среднее количество действий на одного пользователя (6.9 против 5.53)
- пользователи совершили ощутимо больше событий за период (хотя динамика по дням у групп схожая)
- пользователи чаще совершают конверсию в просмотр карточки и в покупку

- в период с 7.12-21.12 кумулятивная конверсия лучше:
  - результаты группы B в конверсии в просмотр карточки и добавление в корзину стабильно ниже, чем у группы A
  - конверсия в покупку во второй половине периода так же ниже у группы B
- пользователи чаще совершают целевые действия в первые дни жизни и в целом ведут себя активнее на протяжении всех 14 дней

Резюмируем: группа A имеет очевидные преимущества перед группой B, что наверняка отразится на результатах теста.

Статистический анализ

Снова обратимся к ТЗ, чтобы корректно провести тест:

- ожидаемый эффект: за 14 дней с момента регистрации пользователи покажут улучшение каждой метрики не менее, чем на 10%,
- проверяется разница в конверсии от начального этапа воронки.

Отберем для каждой группы количество участников, совершивших 'login' и среди них посчитаем долю совершивших каждое из целевых событий.

Для проверки каждой из конверсий сформулируем одни и те же гипотезы:

- **Нулевая гипотеза (H0):** Между группами **нет** статистически значимых различий **(группы равны)**
- **Альтернативная гипотеза (H1):** Между группами **есть** статистически значимые различия **(группы не равны)**

Установим уровень статистической значимости в **0.05**

При каждой проверке какой-либо выборки из данных, мы увеличиваем вероятность получить ошибку. **Применим поправку Бонферрони:** разделим уровень стат.значимости на количество сравнений внутри групп (3)

In [93]:
```
'''Функция, принимает таблицу, 4 вида событий,
подсчитывает количество уникальных пользователей для каждого из событий,
возвращает численные результаты'''
def count_users(df, event_1, event_2, event_3, event_4):
    count_1 = df.query(f'event_name == "{event_1}"')['user_id'].nunique()
    count_2 = df.query(f'event_name == "{event_2}"')['user_id'].nunique()
    count_3 = df.query(f'event_name == "{event_3}"')['user_id'].nunique()
    count_4 = df.query(f'event_name == "{event_4}"')['user_id'].nunique()
    return count_1, count_2, count_3, count_4
```

In [94]:
```
a_login, a_product_page, a_product_cart, a_purchase = count_users(a_group, 'login', 'product_
b_login, b_product_page, b_product_cart, b_purchase = count_users(b_group, 'login', 'product_
```

In [95]:
```
alpha = 0.05
print('Событие: просмотр карточки')
print()
get_z_test(a_product_page, b_product_page, a_login, b_login, alpha/3)
```

```
Событие: просмотр карточки

1685 493 2604 876
Конверсия первой группы: 64.71%
Конверсия второй группы: 56.28%
p-value: 8.195976000324734e-06
Отвергаем нулевую гипотезу, между группами есть статистически значимые различия
```

Результат группы A - выше, разница статистически значима

In [96]:
```
print('Событие: добавление в корзину')
print()
get_z_test(a_product_cart, b_product_cart, a_login, b_login, alpha/3)
```

```
Событие: добавление в корзину

782 244 2604 876
Конверсия первой группы: 30.03%
Конверсия второй группы: 27.85%
p-value: 0.2215941567364419
Не отвергаем нулевую гипотезу, между группами нет статистически значимых различий
```

Разница между группами статистически не значима, но средняя конверсия группы B ниже, что не соответствует ожиданиям тестирования

In [97]:
```
print('Событие: покупка')
print()
get_z_test(a_purchase, b_purchase, a_login, b_login, alpha/3)
```

```
Событие: покупка

833 249 2604 876
Конверсия первой группы: 31.99%
Конверсия второй группы: 28.42%
p-value: 0.04864766695042433
Не отвергаем нулевую гипотезу, между группами нет статистически значимых различий
```

Разница между группами статистически не значима, но средняя конверсия группы B ниже, что не соответствует ожиданиям тестирования

Выводы, решение по тесту

Обработаны 4 полученных датасета.

Замечания по проведению A/B-тестирования:

1. Есть пересечения с конкурирующим тестом

Некорректно проводит несколько тестов на одних и тех же пользователях. Во избежание больших потерь в количестве уникальных пользователей принято решение оставить участников конкурирующего теста т.к.:

- участники из группы A конкурирующего теста не должны быть подвержену его влиянию, т.к. для них не транслировались изменения
- доли участников из группы B конкурирующего теста распределены в раном соотношении в исследуемом тесте

2. Группы изначально неравны количественно

До всех срезов в группе A насчитывали 3824 участников, в B 2877

3. Нет событий проиошедних после 30.12

При этом тест продолжался до 04.01

4. Во время проведения теста проходила маркетинговая активность

Глобально акция не повлияла на поведение пользователей, но доли событий этот период распределны по группам неравномерно (в группе A событий больше, чем в группе B

5. 45.19% пользователей не совершили ни одного действия

Не наблюдается каких-то закономерностей в их характеристиках, таких пользователей пришлось удалить из анализа, т.к. фактически они не принимали участие в тестировании

6. Количество участников не соответсвует ожиданиям

После сокращения числа неактивных пользователей:

- Всего участников теста: 3481
- Отобрано новых клиентов из EU: 8.22%
- Количество совершенных действий 22828 (количество событий сокращено согласно ТЗ: остались только пользователи, успевшие "прожить" 14 дней и события, не произошедшие до 14-го лайфтайма)

7. ИТОГОВЫЕ ГРУППЫ РАСПРЕДЕЛЕНЫ НЕРАВНОМЕРНО

- участников из группы A: 2604
- участников из группы B: 877

Принимая во внимание все 6 пунктов:

Оценка проведения теста: тест нельзя назвать корретным

По имеющимся данным проведен исследовательский анализ.

Результаты исследовательского анализа

Однозначное преимущество группы A над B.

- Больше среднее количество действий на пользователя: 6.9 против 5.53
- ощутимо больше действий в день
- выше среднее количество событий в день на пользователя
- выше конверсия в открытие карточки и в покупку
- кумулятивная конверсия в каждое из целевых действий: конверсия A превышает конверсию B
- доли целевых действий в каждый из лайфтаймов: пользователи группы A чаще совершают целевые действия в первые "дни жизни"

Единственное замеченное преимущество группы B: выше конверсия в добавление корзину и ее прирост по отношению к группе A в начале проведения теста (после середины декабря конверсия A превышает конверсию B)

Результаты тестирования

Конверсии рассчитываются от начала воронки ('login') согласно ТЗ

1. Конверсия в просмотр карточки

- A: 64.71%
- B: 56.28%
- p-value: 8.196

Победа за группой A

2. Конверсия в добавление в корзину

- A: 30.03%
- B: 27.85%
- p-value: 0.222

Между группами нет статистически значимой разницы

3. Конверсия в покупку

- A: 31.99%
- B: 28.42%
- p-value: 0.049

Между группами нет статистически значимой разницы

Учитывая результаты тестирования необходимо зафиксировать победу группы A и остановить тест, НО:

- к результатам теста стоит относиться скептически, т.к. группа B в связи с некорректной разбивкой не показала должной активности
- группа B не показала ожидаемого прироста метрик, но можем ли мы доверять результатам теста с такой разбивкой по группам?

Решение по тесту: рекомендуется провести тест заново, учитывая все описанные замечания по оценке корректности.