Оглавление

Анализ поведения пользователей в мобильном приложении

1 Цель исследования:

1. Изучить воронку продаж внутри мобильного приложения
2. Узнать, как пользователи доходят до покупки. Сколько пользователей доходит до покупки и сколько «застревает» на предыдущих шагах. Определить на каких именно шагах у пользователей возникают трудности
3. Исследовать результаты A/A/B-эксперимента и определить наиболее эффективную группу тестирования

2 Информация о входных данных

Таблица **logs_data** - лог действий и событий, которые совершают пользователи в приложении:

- **`EventName`** — *название события*
- **`DeviceIDHash`** — *уникальный идентификатор пользователя*
- **`EventTimestamp`** — *время события*
- **`ExpId`** — *номер эксперимента:* ***246*** *и* ***247*** *— контрольные группы,* ***248*** *— экспериментальная группа*

3 Оглавление

Шаг 1. Изучение входных данных

Импортирую все необходимые библиотеки и с помощью метода **`.read_csv()`** создаю *DataFrame* с названием **logs_data**.

Ввод [1]:

```
# Импортирую необходимые библиотеки
import pandas as pd               # Библиотека Pandas
import matplotlib.pyplot as plt   # Библиотека для визуализации
import numpy as np                # Библиотека для математических вычислений
import seaborn as sns             # Библиотека для визуализации данных
import datetime as dt             # Библиотека для преобразования к типу данных "дата"
import math as mth                # Библиотека для математических вычислений
from scipy import stats as st     # Библиотека для высокоуровневых математических вычислений
from plotly import graph_objects as go # Библиотека для интерактивной визуализации

ClrG = '\033[32m'                 # Инициализирую переменную для вывода текста зелёным цветом
ClrDef = '\033[0m'                # Инициализирую переменную для вывода текста с дефолтными настройками

# Устанавливаю единый стиль палитры для всех графиков
sns.set_palette('pastel')         # Цветовая палитра пастельных тонов
sns.set_style('dark')             # Тёмный стиль графика

# Создаю DataFrame
logs_data = pd.read_csv('logs_exp.csv',sep='\t' )   # Информация о заведениях общественного питания

# Создаю функцию для получения основной информации о датафрейме
def datainfo(dataframe):
    print(ClrG, 'Таблица:', ClrDef)
    display(dataframe.head(10))
    print(ClrG, 'Числовое описание таблицы:', ClrDef)
    display(dataframe.describe())
    print(ClrG, 'Структура таблицы:', ClrDef)
    dataframe.info()
    data_full_duplicated = dataframe.duplicated().sum()
    print()
    print('Количество пропущенных значений:', ClrG, dataframe.isnull().sum().sum(), ClrDef)   # Вывожу количество пропущенных зна
    print('Количество полных дубликатов:', ClrG, data_full_duplicated, ClrDef)     # Вывожу количество полных дубликатов
    print('Итого процент дубликатов составляет', ClrG, '{:.1%}'.format((data_full_duplicated / len(dataframe))), ClrDef) # Вывожу
```

Изучение таблицы "logs_data"

Ввод [2]:

```
1 datainfo(logs_data)
```

Таблица:

	EventName	DeviceIDHash	EventTimestamp	ExpId
0	MainScreenAppear	4575588528974610257	1564029816	246
1	MainScreenAppear	7416695313311560658	1564053102	246
2	PaymentScreenSuccessful	3518123091307005509	1564054127	248
3	CartScreenAppear	3518123091307005509	1564054127	248
4	PaymentScreenSuccessful	6217807653094995999	1564055322	248
5	CartScreenAppear	6217807653094995999	1564055323	248
6	OffersScreenAppear	8351860793733343758	1564066242	246
7	MainScreenAppear	5682100281902512875	1564085677	246
8	MainScreenAppear	1850981295691852772	1564086702	247
9	MainScreenAppear	5407636962369102641	1564112112	246

Числовое описание таблицы:

	DeviceIDHash	EventTimestamp	ExpId
count	2.441260e+05	2.441260e+05	244126.000000
mean	4.627568e+18	1.564914e+09	247.022296
std	2.642425e+18	1.771343e+05	0.824434
min	6.888747e+15	1.564030e+09	246.000000
25%	2.372212e+18	1.564757e+09	246.000000
50%	4.623192e+18	1.564919e+09	247.000000
75%	6.932517e+18	1.565075e+09	248.000000
max	9.222603e+18	1.565213e+09	248.000000

Структура таблицы:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244126 entries, 0 to 244125
Data columns (total 4 columns):
 #   Column          Non-Null Count   Dtype 
---  ------          --------------   ----- 
 0   EventName       244126 non-null  object
 1   DeviceIDHash    244126 non-null  int64 
 2   EventTimestamp  244126 non-null  int64 
 3   ExpId           244126 non-null  int64 
dtypes: int64(3), object(1)
memory usage: 7.5+ MB

Количество пропущенных значений:  0
Количество полных дубликатов:  413
Итого процент дубликатов составляет  0.2%
```

3.0.1 Получаю следующую структуру таблицы:

- Количество столбцов: **4**
- Количество строк: **244126**
- Тип данных в столбцах: **int64(3), object(1)**
  - Пропущенных значений **нет**
  - Полных дубликатов **413**
  - Все столбцы имеют **корректный** тип данных
  - Аномальных значений **не наблюдается**

Так как суммарный объём дубликатов составляет **0.2%**, я принимаю решение удалить строки с дубликатами.

4 Шаг 2. Предобработка данных

Для проведения дальнейшего исследовательского анализа данных, необходимо привести таблицу `logs_data` к корректному виду:

- *Удалить дубликаты*
- *Изменить заголовки столбцов*
- *Добавить новые столбцы с датой и временем совершения события*

Ввод [3]:

```
# Удаление дубликатов
print('Строк до обработки:', ClrG, len(logs_data), ClrDef)
logs_data = logs_data.drop_duplicates()
print('Строк после обработки:', ClrG, len(logs_data), ClrDef)

# Переименование столбцов
logs_data.columns = ['event', 'device', 'timestamp', 'group']

# Создаю столбцы с корректной датой события
logs_data['datetime'] = pd.to_datetime(logs_data['timestamp'], errors='ignore', unit='s') # Дата и время события
logs_data['date'] = logs_data['datetime'].dt.floor('d') # Дата события
display(logs_data.head()) # Вывод таблицы на экран
```

```
Строк до обработки:  244126
Строк после обработки:  243713
```

	event	device	timestamp	group	datetime	date
0	MainScreenAppear	4575588528974610257	1564029816	246	2019-07-25 04:43:36	2019-07-25
1	MainScreenAppear	7416695313311560658	1564053102	246	2019-07-25 11:11:42	2019-07-25
2	PaymentScreenSuccessful	3518123091307005509	1564054127	248	2019-07-25 11:28:47	2019-07-25
3	CartScreenAppear	3518123091307005509	1564054127	248	2019-07-25 11:28:47	2019-07-25
4	PaymentScreenSuccessful	6217807653094995999	1564055322	248	2019-07-25 11:48:42	2019-07-25

4.1 Вывод

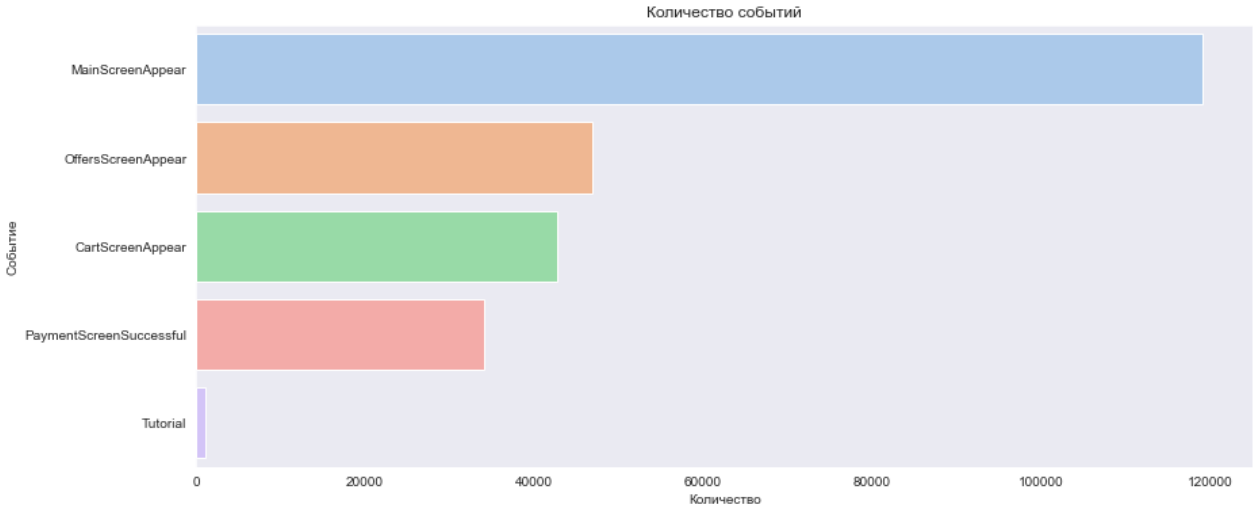
В ходе предобработки данных было выявлено и удалено **413** дубликатов, а так же изменены названия столбцов таблицы **`logs_data`** . Также были добавлены столбцы **datetime** и **date** с информацией о дате и времени совершения каждого события. Теперь данные пригодны для проведения исследовательского анализа.

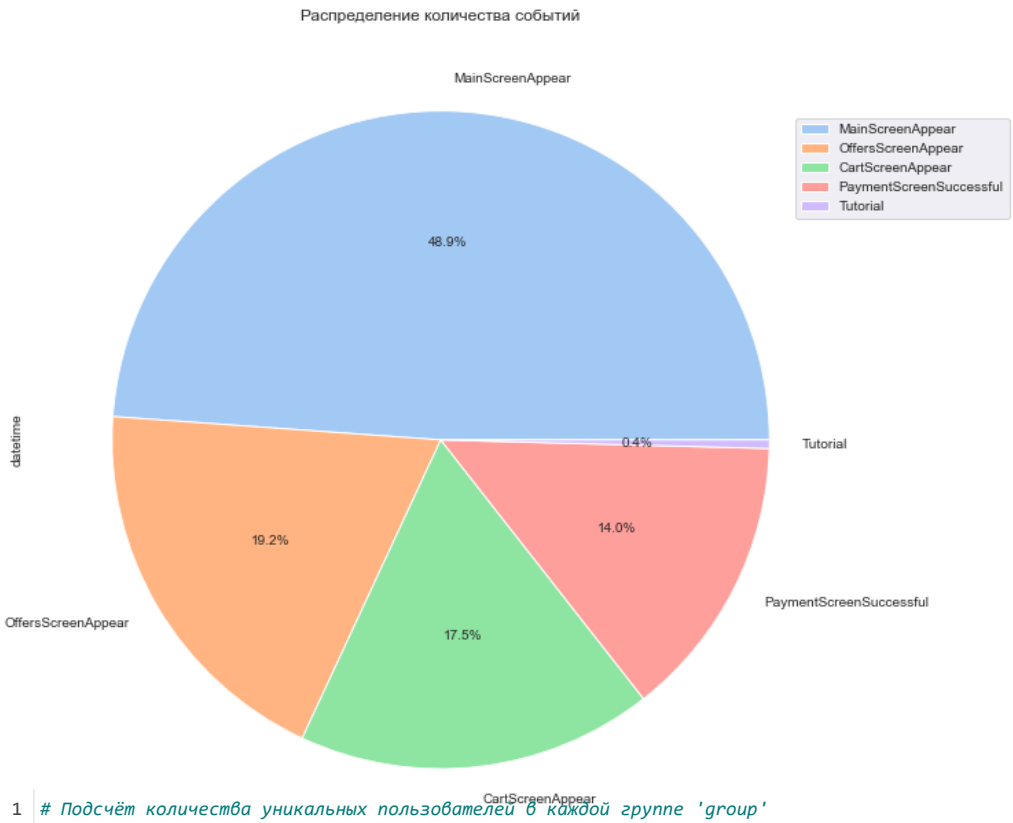
5 Шаг 3. Исследовательский анализ данных

5.1 3.1. Количество событий в логе

Ввод [4]:

```
# Подсчёт количества записей для каждого события
event_count = logs_data.groupby('event').agg({'datetime':'count'}).reset_index().sort_values(by='datetime', ascending=False)
event_count.columns = ['Событие', 'Количество']

# Рисую график количества записей для каждого события
plt.figure(figsize=(14, 6)) # Размер графика
plt.title('Количество событий') # Название графика
sns.barplot(x='Количество', y='Событие', data=event_count)
plt.show()

# Рисую круговую диаграмму
logs_data.groupby('event').agg({'datetime':'count'}).sort_values(by='datetime', ascending=False).plot(y="datetime", kind="pie", f
plt.legend(bbox_to_anchor=(0.6, 0, 0.6, 0.9)) # Расположение легенды на графике
plt.title('Распределение количества событий') # Название графика
plt.show()

# Вывожу таблицу на экран
display(event_count)
print('Количество отслеживаемых событий:', ClrG, len(event_count), ClrDef)
print('Количество записей для данных событий:', ClrG, len(logs_data))
```





ное событие составляет **49%** от всего

е было вызвано всего **1018 раз**, что

118 раз, что составляет **14%** от общей

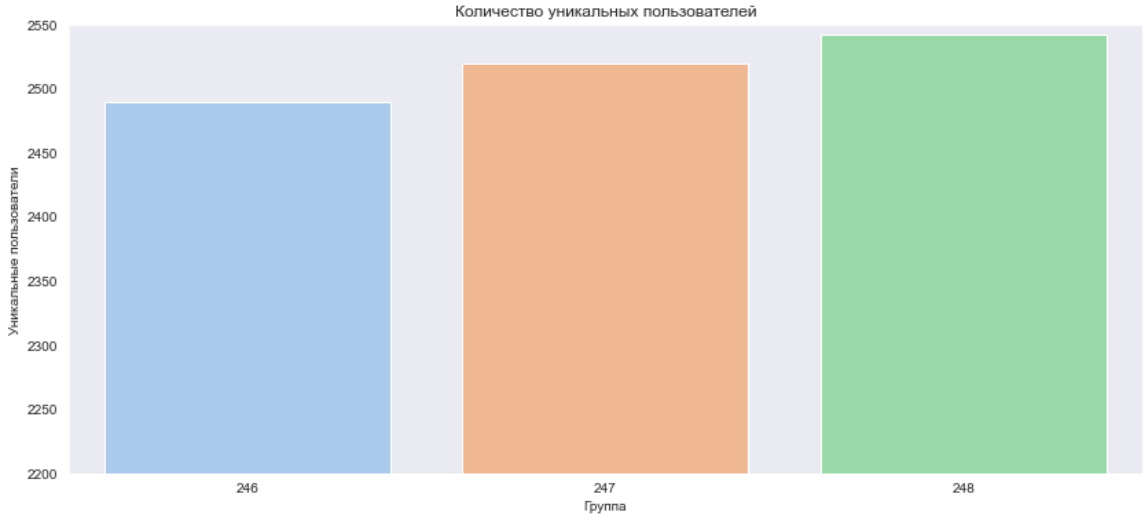
ется **`PaymentScreenSuccessful`** .

```
# Подсчёт количества уникальных пользователей в каждой группе 'group'
user_count = logs_data.groupby('group').agg({'device':'nunique'}).reset_index()
user_count.columns = ['Группа', 'Уникальные пользователи']

# Рисую график количества уникальных пользователей в каждой группе
plt.figure(figsize=(14, 6)) # Размер графика
plt.title('Количество уникальных пользователей') # Название графика
sns.barplot(y='Уникальные пользователи', x='Группа', data=user_count).set_ylim(2200, 2550)
plt.show()

# Вывод информации на экран
print(ClrG, 'Количество уникальных пользователей:', ClrDef)
print('   В группе 246 составляет', ClrG, user_count.iloc[0]['Уникальные пользователи'], ClrDef)
print('   В группе 247 составляет', ClrG, user_count.iloc[1]['Уникальные пользователи'], ClrDef)
print('   В группе 248 составляет', ClrG, user_count.iloc[2]['Уникальные пользователи'], ClrDef)
print('Всего', ClrG, user_count['Уникальные пользователи'].sum(), ClrDef)
```

	Событие	Количество
1	MainScreenAppear	119101
2	OffersScreenAppear	46808
3	CartScreenAppear	42668
4	PaymentScreenSuccessful	34118
5	Tutorial	1018

Количество отслеживаемых событий: 5

Количество записей для анализа: 243713




```
Количество уникальных пользователей:
   В группе 246 составляет  2489
   В группе 247 составляет  2520
   В группе 248 составляет  2542
Всего  7551
```

5.4 Вывод

Всего уникальных пользователей в логе **7551 человек** из них:

- В 246 группе: **2489 человек**
- В 247 группе: **2520 человек**
- В 248 группе: **2542 человек**

5.5 3.3. Среднее количество событий на одного пользователя

Ввод [6]:

```
# Вычисляю количество событий для каждого уникального пользователя
event_data = logs_data.groupby('device').agg({'event':'count'}).reset_index()

# Рисую график "Диаграмма размаха" для показателя "количество посадочных мест" по видам объекта питания
plt.figure(figsize=(14, 6)) # Размер графика
plt.title('Диаграмма размаха количества событий')
sns.boxplot(y='event', data=event_data).set(xlabel='', ylabel='Количество событий')
plt.xticks(rotation= 45)
plt.show()

print('99% пользователей совершают менее', ClrG, event_data['event'].quantile(0.99), 'событий') # Вычисляю значение 99 квантиля
```


```
99% пользователей совершают менее  200.5 событий
```

Таким образом, все значения, которые превышают **200 событий** можно считать аномальными выбросами от которых необходимо избавиться.

Ввод [7]:

```
# Получаю список идентификаторов устройств с аномальным количеством событий
anomaly_device_id = list(event_data.query('event > 200')['device'])
print('Количество строк:')
print(' до обработки', ClrG, len(logs_data), 'строк', ClrDef)
# Удаляю все записи с идентификаторами из списка 'anomaly_device_id'
clear_logs_data = logs_data.query('device not in @anomaly_device_id')
print(' после обработки', ClrG, len(clear_logs_data), 'строк', ClrDef)
print('Удалено', ClrG, '{:.2%}'.format((len(logs_data) - len(clear_logs_data)) / len(logs_data)))
```

```
Количество строк:
 до обработки  243713 строк
 после обработки  209333 строк
Удалено  14.11%
```

От записей с идентификаторами устройств, для которых характерны аномальные значения количества совершенных событий я успешно избавился, теперь можно посчитать сколько в среднем совершает событий один уникальный пользователь в приложении.

Ввод [8]:

```
# Подсчёт среднего количества совершаемых событий для одного пользователя
print('В среднем один пользователь совершает', ClrG, int(clear_logs_data.groupby('device').agg({'event':'count'})['event'].mean())
```

```
В среднем один пользователь совершает  28 событий
```

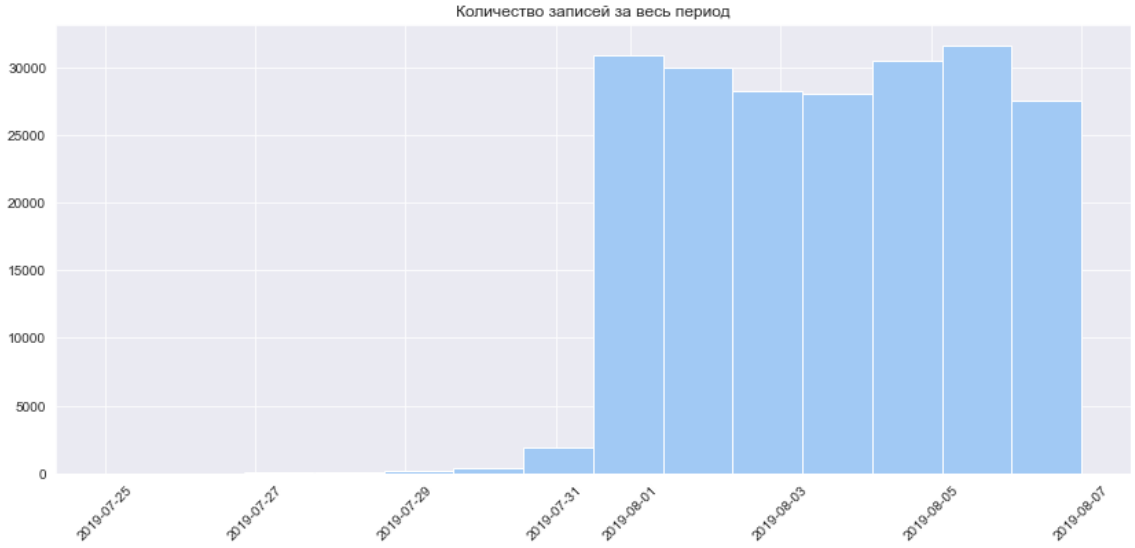
5.6 Вывод

- В среднем каждый пользователь совершает **28 событий**

5.7 3.4. Определение временного промежутка анализа

Ввод [9]:

```
# Гистограмма количества записей за весь период
plt.figure(figsize=(14, 6)) # Размер графика
plt.title('Количество записей за весь период')
clear_logs_data['date'].hist(bins=14)
plt.xticks(rotation= 45)
plt.show()

# Расчёт периода ведения лога
print('Дата первой записи в логе', ClrG, clear_logs_data['datetime'].min(), ClrDef)
print('Дата последней записи в логе', ClrG, clear_logs_data['datetime'].max(), ClrDef)
print('Период ведения лога', ClrG, clear_logs_data['datetime'].max() - clear_logs_data['datetime'].min())
```




```
Дата первой записи в логе  2019-07-25 04:43:36
Дата последней записи в логе  2019-08-07 21:15:17
Период ведения лога  13 days 16:31:41
```

На гистограмме видно, что в **первые 6 дней** количество записей крайне низкое и начиная с **2019-08-01** количество записей резко возрастает. Таким образом я считаю, что только начиная **с 1 августа 2019 года** начинает накапливаться достаточное количество записей в логах для проведения событийного анализа, следовательно, все записи **до 2019-08-01** можно не учитывать в дальнейшем анализе, так как они исказят общую картину при проведении дальнейшего анализа.

Ввод [10]:

```
# Фильтрую датасет и оставляю записи которые были сделаны позже 2019-07-31
clear_logs_data = clear_logs_data.query('date > "2019-07-31"')
# Вывожу временной период на экран
print('Временной период составляет с', ClrG, clear_logs_data['datetime'].min(), ClrDef, 'по', ClrG, clear_logs_data['datetime'].m
```

```
Временной период составляет с  2019-08-01 00:07:28  по  2019-08-07 21:15:17
```

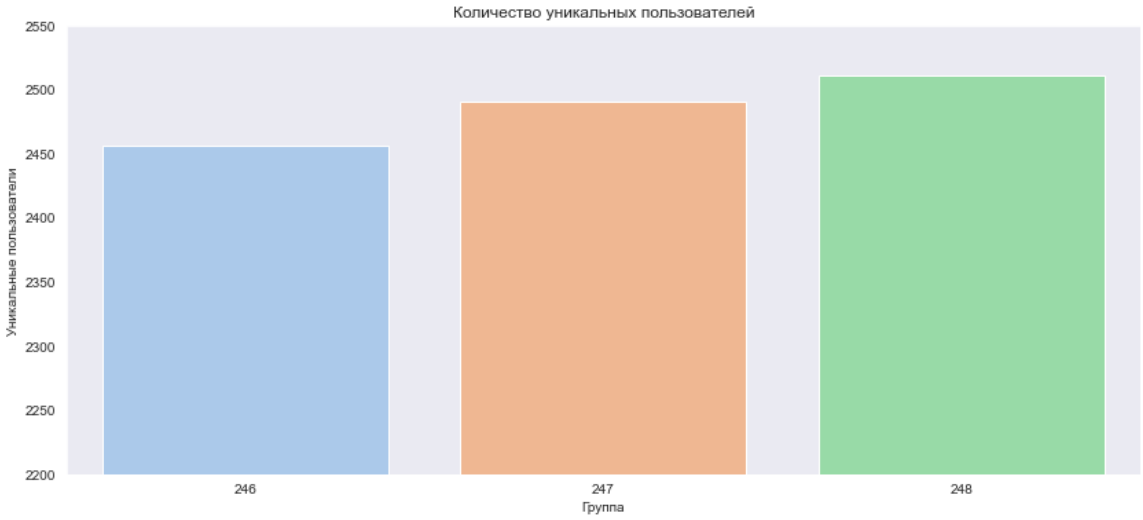
5.8 Вывод

Таким образом после удаления временного периода для которого характерно крайне низкое количество записей, мне удалось установить корректный временной период для дальнейшего анализа и он составляет: **с 2019-08-01 00:07:28 по 2019-08-07 21:15:17**

5.9 3.5. Количество пользователей в каждой группе тестирования

Ввод [11]:

```
# Подсчёт количества уникальных пользователей в каждой группе 'group'
user_count = clear_logs_data.groupby('group').agg({'device':'nunique'}).reset_index()
user_count.columns = ['Группа', 'Уникальные пользователи']

# Рисую график количества уникальных пользователей в каждой группе
plt.figure(figsize=(14, 6)) # Размер графика
plt.title('Количество уникальных пользователей') # Название графика
sns.barplot(y='Уникальные пользователи', x='Группа', data=user_count).set_ylim(2200, 2550)
plt.show()

# Вывод информации на экран
print(ClrG, 'Количество уникальных пользователей:', ClrDef)
print('   В группе 246 составляет', ClrG, user_count.iloc[0]['Уникальные пользователи'], ClrDef)
print('   В группе 247 составляет', ClrG, user_count.iloc[1]['Уникальные пользователи'], ClrDef)
print('   В группе 248 составляет', ClrG, user_count.iloc[2]['Уникальные пользователи'], ClrDef)
print(' Всего', ClrG, user_count['Уникальные пользователи'].sum(), ClrDef)
```




```
Количество уникальных пользователей:
   В группе 246 составляет  2456
   В группе 247 составляет  2491
   В группе 248 составляет  2511
Всего  7458
```

5.10 Вывод

Выполнив исследовательский анализ данных, были выявлены следующие ключевые факторы:

1. Определен актуальный временной период для дальнейшего анализа: **с 2019-08-01 00:07:28 по 2019-08-07 21:15:17**

2. Количество уникальных пользователей в каждой группе:
   - В *246 группе*: **2456** пользователей
   - В *247 группе*: **2491** пользователей
   - В *248 группе*: **2511** пользователей

3. Среднее количество совершаемых событий одним пользователем в приложении: **28 событий**

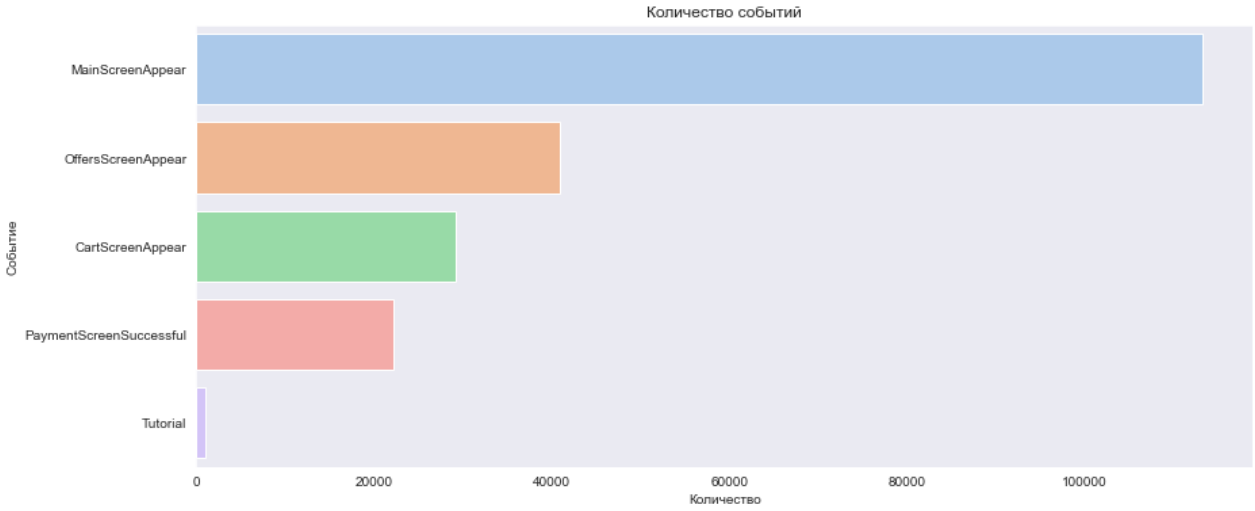
4. Особенности событий в приложении:
   - Наиболее часто вызываемое событие в приложении - `MainScreenAppear` - основной экран приложения, данное событие составляет **49%** от всего объёма событий совершаемых в приложении, данное событие было вызвано **119101 раз**
   - Наименее часто вызываемое событие в приложении - `Tutorial` - руководство приложения - данное событие было вызвано всего **1018 раз**, что составляет **0,4%** от общего объёма событий
   - Ключевое событие для приложения - `PaymentScreenSuccessful` - экран успешной оплаты, было вызвано **34118 раз**, что составляет **14%** от общей доли событий, следовательно можно считать, что каждое седьмое событие совершаемое в приложении является `PaymentScreenSuccessful`.

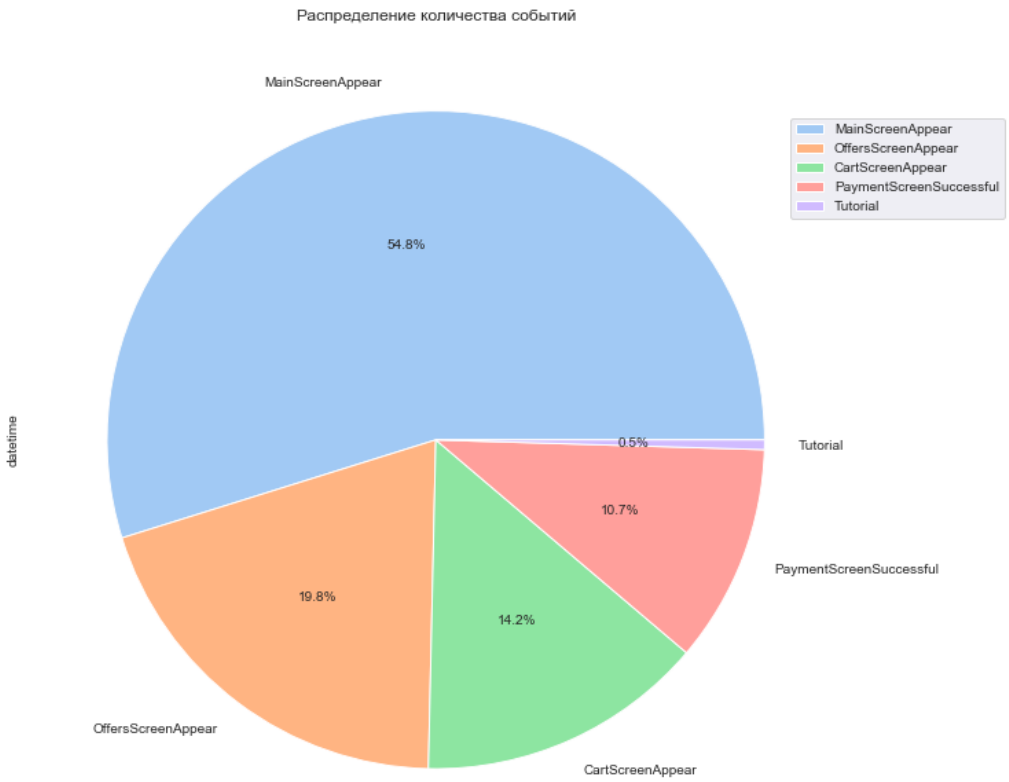
6 Шаг 4. Изучение воронки событий

6.1 4.1. Анализ всех событий

Ввод [12]:

```
# Подсчёт количества записей для каждого события
event_count = clear_logs_data.groupby('event').agg({'datetime':'count'}).reset_index().sort_values(by='datetime', ascending=False
event_count.columns = ['Событие', 'Количество']

# Рисую график количества записей для каждого события
plt.figure(figsize=(14, 6)) # Размер графика
plt.title('Количество событий') # Название графика
sns.barplot(x='Количество', y='Событие', data=event_count)
plt.show()

# Рисую круговую диаграмму
clear_logs_data.groupby('event').agg({'datetime':'count'}).sort_values(by='datetime', ascending=False).plot(y="datetime", kind="p
plt.legend(bbox_to_anchor=(0.6, 0, 0.6, 0.9)) # Расположение легенды на графике
plt.title('Распределение количества событий') # Название графика
plt.show()

# Вывожу таблицу на экран
display(event_count)
print('Количество отслеживаемых событий:', ClrG, len(event_count), ClrDef)
print('Количество записей для данных событий:', ClrG, len(logs_data))
```





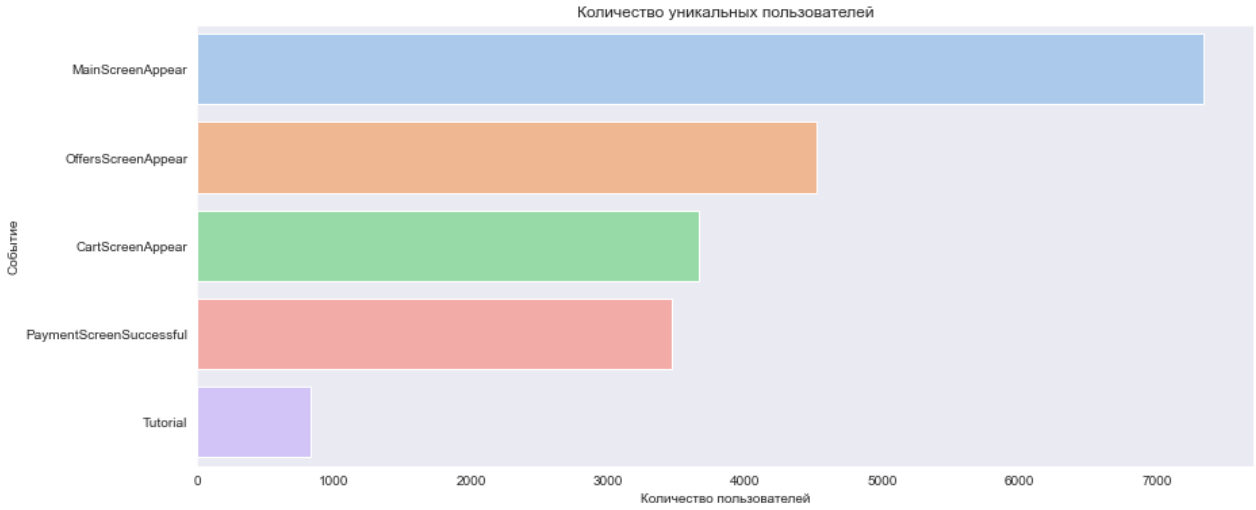
	Событие	Количество
1	MainScreenAppear	113264
2	OffersScreenAppear	40956
0	CartScreenAppear	29250
3	PaymentScreenSuccessful	22164
4	Tutorial	981

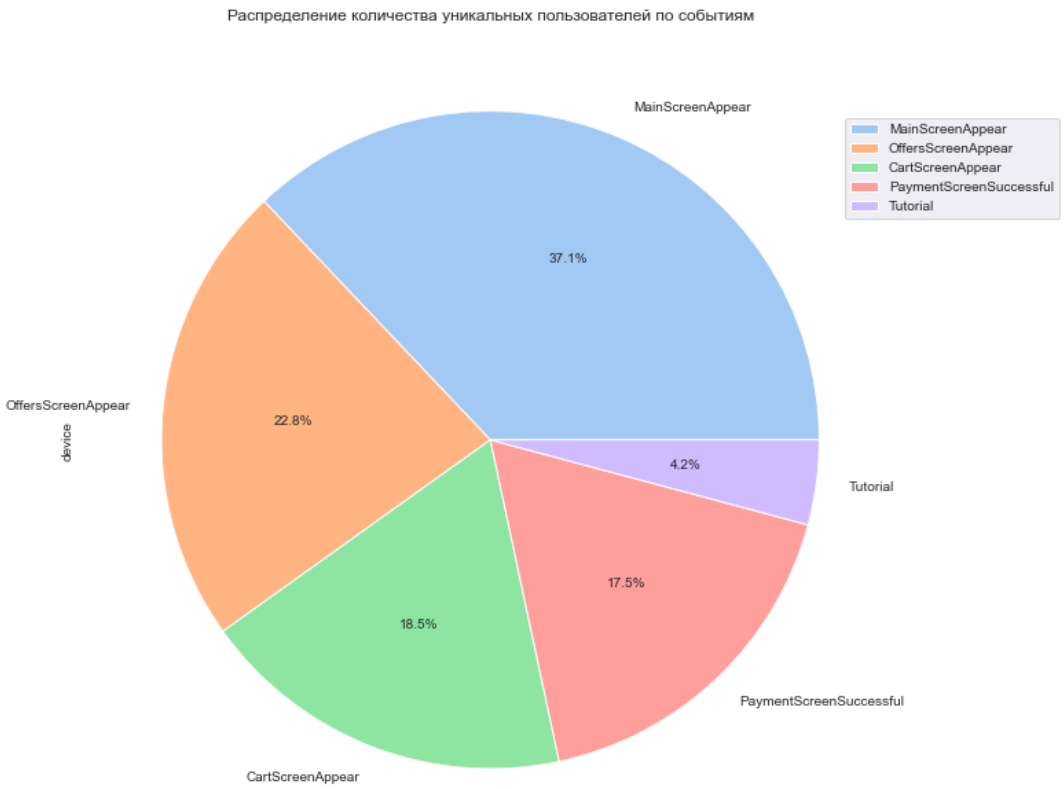
```
Количество отслеживаемых событий:  5
Количество записей для данных событий:  243713
```

6.2 4.2. Количество пользователей совершивших событие

Ввод [13]:

```
# Подсчёт количества уникальных пользователей для каждого события
event_user_count = clear_logs_data.groupby('event').agg({'device':'nunique'}).reset_index().sort_values('device', ascending=False
event_user_count.columns = ['Событие', 'Количество пользователей']

# Всего уникальных пользователей в логе
total_users = len(clear_logs_data['device'].unique())

# Рисую график количества уникальных пользователей для каждого события
plt.figure(figsize=(14, 6)) # Размер графика
plt.title('Количество уникальных пользователей') # Название графика
sns.barplot(x='Количество пользователей', y='Событие', data=event_user_count)
plt.show()

# Рисую круговую диаграмму
clear_logs_data.groupby('event').agg({'device':'nunique'}).sort_values('device', ascending=False).plot(y="device", kind="pie", fi
plt.legend(bbox_to_anchor=(0.6, 0, 0.6, 0.9)) # Расположение легенды на графике
plt.title('Распределение количества уникальных пользователей по событиям') # Название графика
plt.show()

# Доля пользователей от общего количества пользователей приложения, которые совершили событие хотябы 1 раз
event_user_count['Доля от всех пользователей'] = event_user_count['Количество пользователей'] / total_users * 100
event_user_count['Доля от всех пользователей'] = round(event_user_count['Доля от всех пользователей'], 1)

# Вывожу таблицу на экран
display(event_user_count)
print('Количество уникальных пользователей', ClrG, total_users, ClrDef)
print('Количество отслеживаемых событий', ClrG, len(event_user_count), ClrDef)
```





:ет свидетельствовать о том, что в
ией, но только **7344 пользователей**
жить что у них не запустилось
й факт необходимо исследовать.

	Событие	Количество пользователей	Доля от всех пользователей
1	MainScreenAppear	7344	98.5
2	OffersScreenAppear	4517	60.6
0	CartScreenAppear	3658	49.0
3	PaymentScreenSuccessful	3463	46.4
4	Tutorial	824	11.0

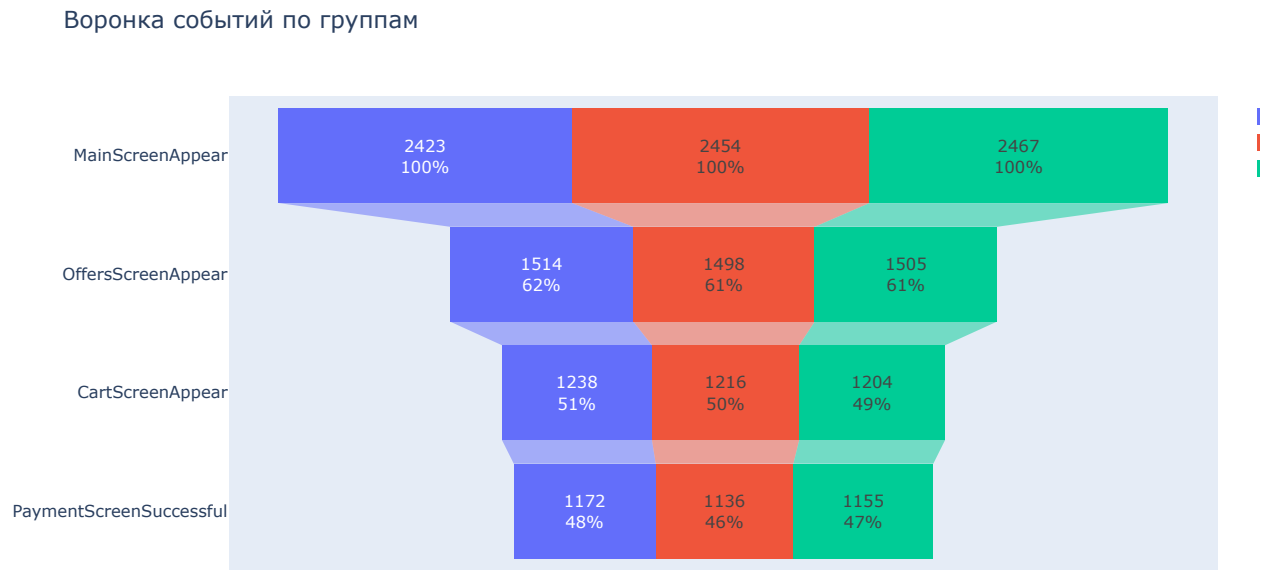
```
Количество уникальных пользователей  7458
Количество отслеживаемых событий  5
```

7.1 4.3. Расчёт воронки событий

Ввод [14]:

```
# Создаю таблицу с количеством уникальных пользователей для каждого события в каждой группе теста [246, 247, 248]
group_event_count = clear_logs_data.query('event != "Tutorial"').groupby(['group', 'event']).agg({'device':'nunique'}).reset_inde

display(group_event_count)

# График - Воронка событий по группам теста
fig = go.Figure()
for group in [246,247,248]:
    funnel = group_event_count.query('group ==@group')
    fig.add_trace(go.Funnel(name = group, y = funnel['event'], x = funnel['device'], orientation = "h", textinfo = "value+percent
    fig.update_layout(title='Воронка событий по группам')
fig.show()
```

	group	event	device
9	248	MainScreenAppear	2467
5	247	MainScreenAppear	2454
1	246	MainScreenAppear	2423
2	246	OffersScreenAppear	1514
10	248	OffersScreenAppear	1505
6	247	OffersScreenAppear	1498
0	246	CartScreenAppear	1238
4	247	CartScreenAppear	1216
8	248	CartScreenAppear	1204
3	246	PaymentScreenSuccessful	1172
11	248	PaymentScreenSuccessful	1155
7	247	PaymentScreenSuccessful	1136



7.2 Вывод

Таким образом, получаются следующие результаты:

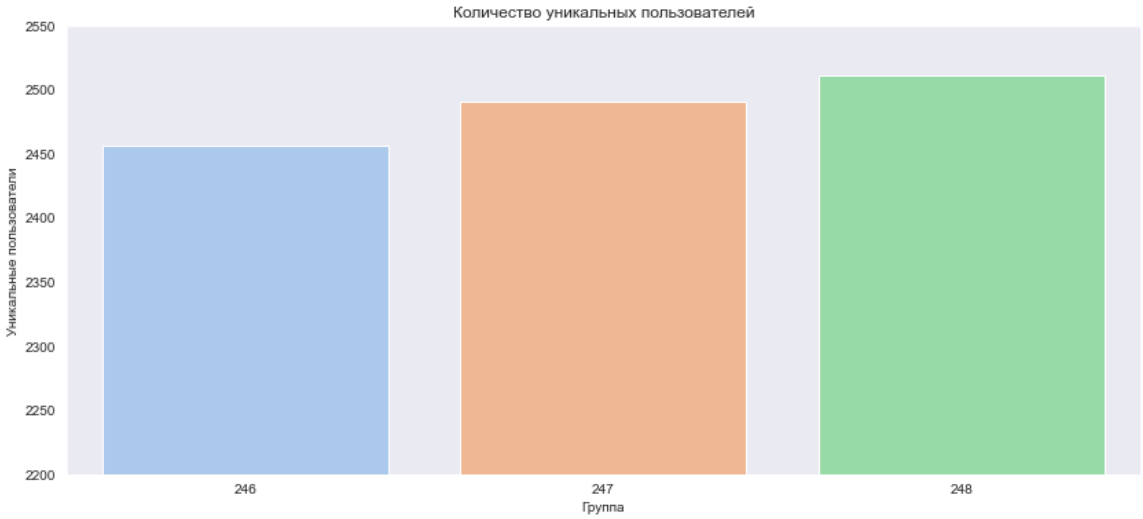
- В среднем **47%** пользователей от общего количества пользователей совершают покупку и производят успешную оплату в приложении.
- Большее всего пользователей теряется на *2ом шаге воронки* - **OffersScreenAppear**, то есть данное событие совершает только **61%** пользователей от общего количества пользователей совершивших предыдущее событие.
- На остальных шагах воронки потери пользователей не такие серьёзные.
- В целом ситуация по каждой тестовой группе идентична, разница между группами составляет примерно **1-2%**.

8 Шаг 5. Изучение результатов эксперимента

8.1 5.1. Количество пользователей в каждой экспериментальной группе

Ввод [15]:

```
# Подсчёт количества уникальных пользователей в каждой экспериментальной группе
user_count = clear_logs_data.groupby('group').agg({'device':'nunique'}).reset_index()
user_count.columns = ['Группа', 'Уникальные пользователи']

# Рисую график количества уникальных пользователей в каждой группе
plt.figure(figsize=(14, 6)) # Размер графика
plt.title('Количество уникальных пользователей') # Название графика
sns.barplot(y='Уникальные пользователи', x='Группа', data=user_count).set_ylim(2200, 2550)
plt.show()

# Вывод информации на экран
print(ClrG, 'Количество уникальных пользователей:', ClrDef)
print('   В группе 246 составляет', ClrG, user_count.iloc[0]['Уникальные пользователи'], ClrDef)
print('   В группе 247 составляет', ClrG, user_count.iloc[1]['Уникальные пользователи'], ClrDef)
print('   В группе 248 составляет', ClrG, user_count.iloc[2]['Уникальные пользователи'], ClrDef)
```




```
Количество уникальных пользователей:
   В группе 246 составляет  2456
   В группе 247 составляет  2491
   В группе 248 составляет  2511
```

8.2 Вывод

После всех манипуляций с данными получаются следующие результаты:

- Количество уникальных пользователей:
  - *В 246 группе составляет*: **2456**
  - *В 247 группе составляет*: **2491**
  - *В 248 группе составляет*: **2511**

8.3 5.2. Проведение A/A - тестирования

- Нулевая гипотеза сформулирована следующим образом: **Доли двух контрольных групп не отличаются друг от друга**
- Альтернативная гипотеза сформулирована следующим образом: **Доли двух контрольных групп отличаются друг от друга**

Ввод [16]:

```
# Создаю сводную таблицу с количеством пользователей в каждом событии для каждой группы
user_counts = (
    clear_logs_data
    .groupby(['group', 'event']).agg({'device':'nunique'})
    .pivot_table(index='event', columns='group', values='device')
    .reset_index()
    .sort_values(246, ascending=False)
)

display(user_counts)
# Расчитываю общее количество пользователей в каждой группе
total_user = clear_logs_data.groupby('group').agg({'device':'nunique'}).reset_index()
display(total_user)

# Создаю списки с количеством пользователей для каждого события в каждой группе
# 246 экспериментальная группа:
A1_list = [total_user.iloc[0]['device'], user_counts.iloc[0][246], user_counts.iloc[1][246], user_counts.iloc[2][246], user_count
# 247 экспериментальная группа:
A2_list = [total_user.iloc[1]['device'], user_counts.iloc[0][247], user_counts.iloc[1][247], user_counts.iloc[2][247], user_count
# 248 экспериментальная группа:
B_list = [total_user.iloc[2]['device'], user_counts.iloc[0][248], user_counts.iloc[1][248], user_counts.iloc[2][248], user_counts
```

group	event	246	247	248
1	MainScreenAppear	2423	2454	2467
2	OffersScreenAppear	1514	1498	1505
0	CartScreenAppear	1238	1216	1204
3	PaymentScreenSuccessful	1172	1136	1155
4	Tutorial	269	279	276

	group	device
0	246	2456
1	247	2491
2	248	2511

Ввод [17]:

```
A1_list
```

Out[17]:

```
[2456, 2423, 1514, 1238, 1172, 269]
```

Для автоматизации процесса расчёта статистически значимой разницы между долями двух совокупностей необходимо создать функцию **`find_stat_value`** , которая принимает два списка с количеством пользователей по каждому событию для двух экспериментальных групп и расчитывает **p-value** для каждого события. В качестве границы критического уровня статистической значимости **alpha** я считаю значение **5%**.

Ввод [18]:

```
# Функция для расчёта статистически значимой разницы между долями двух генеральных совокупностей
def find_stat_value(first_list, second_list):

    # Список с названием события в приложении
    event_name_list = ['MainScreenAppear', 'OffersScreenAppear', 'CartScreenAppear', 'PaymentScreenSuccessful', 'Tutorial']
    # Основной цикл функции, который вычисляет p-value и проверяет нулевую гипотезу для каждого события
    for x in range(0, len(first_list)-1):
        x1 = first_list[x+1]  # Второе значение в первой группе
        x2 = second_list[x+1] # Второе значение во второй группе
        y1 = first_list[x]    # Первое значение в первой группе
        y2 = second_list[x]   # Первое значение во второй группе

        alpha = 0.05 # Указание критического уровня статистической значимости
        successes = np.array([x1, x2])
        trials = np.array([y1, y2])
        p1 = successes[0]/trials[0] # Расчёт значения пропорции в первой группе
        p2 = successes[1]/trials[1] # Расчёт значения пропорции во второй группе
        # Расчёт совокупной пропорции в комбинированном датасете
        p_combined = (successes[0] + successes[1]) / (trials[0] + trials[1])
        difference = p1 - p2  # Расчёт азницы пропорций между группами
        # Расчёт статистики в ст.отклонениях стандартного нормального распределения
        z_value = difference / mth.sqrt(p_combined * (1 - p_combined) * (1/trials[0] + 1/trials[1]))
        distr = st.norm(0, 1) # Указание стандартного отклонения в нормальном распределении
        p_value = 1 - distr.cdf(z_value) # Расчёт значения статистической разницы между группами
        # Вывод полученных результатов на экран
        print('* Событие:', ClrG, event_name_list[x], ClrDef)
        print('* p-значение: ', ClrG, p_value, ClrDef)
        # Сравнение полученного p-value с установленным уровнем статистической значимости
        if (p_value < alpha):
            print("* Результат:", ClrG, "Нулевая гипотеза отвергнута: между долями есть значимая разница", ClrDef)
        else:
            print("* Результат:", ClrG, "Нулевая гипотеза не отвергнута: нет оснований считать доли разными", ClrDef)
        print('    ')
```

Функция **`find_stat_value`** успешно создана, теперь можно передать ей значения экспериментальных групп 246 и 247 и выяснить, имеют ли данные группы статистически значимые различия.

Ввод [19]:

```
# Вызываю функцию 'find_stat_value' для групп A1 и A2 [246 и 247]
find_stat_value(A1_list, A2_list)
```

```
* Событие:  MainScreenAppear
* p-значение:   0.33654759036042425
* Результат:  Нулевая гипотеза не отвергнута: нет оснований считать доли разными

* Событие:  OffersScreenAppear
* p-значение:   0.1501975094129797
* Результат:  Нулевая гипотеза не отвергнута: нет оснований считать доли разными

* Событие:  CartScreenAppear
* p-значение:   0.33708768208369055
* Результат:  Нулевая гипотеза не отвергнута: нет оснований считать доли разными

* Событие:  PaymentScreenSuccessful
* p-значение:   0.09569409860801947
* Результат:  Нулевая гипотеза не отвергнута: нет оснований считать доли разными

* Событие:  Tutorial
* p-значение:   0.8179103445228788
* Результат:  Нулевая гипотеза не отвергнута: нет оснований считать доли разными
```

8.4 Вывод

Выполнив A/A - тестирование двух экспериментальных групп, можно сделать следуюище выводы:

- Статистически значимой разницы между долями двух групп **нет**. Данное явление характерно для **всех событий** в обоих группах.
- Следовательно, можно считать, что разбиение на группы работает **корректно**.

8.5 5.3. Проведение А/Б - тестирования

В ходе проведения А/Б-теста необходимо сравнить каждую контрольную группу с группой **Б** по отдельности, а также сравнить объединенную группу **А** с группой **Б**. Таким образом результаты тестирования получатся наиболее достоверными.

- Нулевая гипотеза сформулирована следующим образом: **Доли двух контрольных групп не отличаются друг от друга**
- Альтернативная гипотеза сформулирована следующим образом: **Доли двух контрольных групп отличаются друг от друга**

8.5.1 5.1.1. Тестирование контрольных групп: А1 / Б

Ввод [20]:

```
# Вызываю функцию 'find_stat_value' для групп A1 и B [246 и 248]
find_stat_value(A1_list, B_list)
```

```
* Событие:  MainScreenAppear
* p-значение:   0.12190005370239487
* Результат:  Нулевая гипотеза не отвергнута: нет оснований считать доли разными

* Событие:  OffersScreenAppear
* p-значение:   0.14363753866283124
* Результат:  Нулевая гипотеза не отвергнута: нет оснований считать доли разными

* Событие:  CartScreenAppear
* p-значение:   0.10807461677365149
* Результат:  Нулевая гипотеза не отвергнута: нет оснований считать доли разными

* Событие:  PaymentScreenSuccessful
* p-значение:   0.9293743761725728
* Результат:  Нулевая гипотеза не отвергнута: нет оснований считать доли разными

* Событие:  Tutorial
* p-значение:   0.7045566861176883
* Результат:  Нулевая гипотеза не отвергнута: нет оснований считать доли разными
```

8.6 Вывод

Между количеством пользователей совершивших каждое событие в группе **A1** и **Б** нету статистически значимой разницы, можно считать, что доля пользователей совершивших одно и тоже событие одинаковая для обеих групп.

8.6.1 5.1.2. Тестирование контрольных групп: A2 / Б

Ввод [21]:

```
# Вызываю функцию 'find_stat_value' для групп A2 и B [247 и 248]
find_stat_value(A2_list, B_list)
```

```
* Событие:  MainScreenAppear
* p-значение:   0.22726681153781514
* Результат:  Нулевая гипотеза не отвергнута: нет оснований считать доли разными

* Событие:  OffersScreenAppear
* p-значение:   0.4891199481843963
* Результат:  Нулевая гипотеза не отвергнута: нет оснований считать доли разными

* Событие:  CartScreenAppear
* p-значение:   0.20785666685644766
* Результат:  Нулевая гипотеза не отвергнута: нет оснований считать доли разными

* Событие:  PaymentScreenSuccessful
* p-значение:   0.9969959423892376
* Результат:  Нулевая гипотеза не отвергнута: нет оснований считать доли разными

* Событие:  Tutorial
* p-значение:   0.3554124559773988
* Результат:  Нулевая гипотеза не отвергнута: нет оснований считать доли разными
```

8.7 Вывод

Между количеством пользователей совершивших каждое событие в группе **A2** и **Б** нету статистически значимой разницы, можно считать, что доля пользователей совершивших одно и тоже событие одинаковая для обеих групп.

8.7.1 5.1.3. Тестирование контрольных групп: A / Б

Ввод [22]:

```
# Создаю объединенную экспериментальную группу A
A_list = []
for x in range(0, len(A1_list)):
    A_list.append(A1_list[x] + A2_list[x])

# Вызываю функцию 'find_stat_value' для групп A и B
find_stat_value(A_list, B_list)
```

```
* Событие:  MainScreenAppear
* p-значение:   0.13093201429777923
* Результат:  Нулевая гипотеза не отвергнута: нет оснований считать доли разными

* Событие:  OffersScreenAppear
* p-значение:   0.26525951081798116
* Результат:  Нулевая гипотеза не отвергнута: нет оснований считать доли разными

* Событие:  CartScreenAppear
* p-значение:   0.11703214149530194
* Результат:  Нулевая гипотеза не отвергнута: нет оснований считать доли разными

* Событие:  PaymentScreenSuccessful
* p-значение:   0.9912973834300848
* Результат:  Нулевая гипотеза не отвергнута: нет оснований считать доли разными

* Событие:  Tutorial
* p-значение:   0.5396013457001718
* Результат:  Нулевая гипотеза не отвергнута: нет оснований считать доли разными
```

8.8 Вывод

В ходе проведения А/Б теста, удалось выяснить, что между количеством пользователей совершивших каждое событие в группе **A2** и **Б** нету статистически значимой разницы, можно считать, что доля пользователей совершивших одно и тоже событие одинаковая для обеих групп. Следовательно, можно сделать вывод, что ***новые шрифты, которые показывали пользователям из группы `Б` никак не повлияли на поведение пользователей внутри приложения***.

9 Общий вывод

- Событийный анализ позволил определить следующие особенности в поведении пользователей:
  - Наиболее часто вызываемым событием является - **`MainScreenAppear`** - основной экран приложения, данное событие было вызвано **119101 раз**, что составляет **49%** от всего объёма событий совершаемых в приложении
  - Наименее часто вызываемым событием является - **`Tutorial`** - руководство приложения, данное событие было вызвано всего **1018 раз**, что составляет **0,4%** от общего объёма событий
  - Ключевое событие для приложения - **`PaymentScreenSuccessful`** - экран успешной оплаты, было вызвано **34118 раз**, что составляет **14%** от общей доли событий, следовательно можно считать, что каждое седьмое событие совершаемое в приложении является **PaymentScreenSuccessful**.
  - В среднем каждый пользователь совершает **28 событий** в приложении, а 99% пользователей совершают **менее 200** событий
- Анализ воронки событий внутри приложения выявил следующие особенности:
  - В среднем **47% пользователей** от общего количества пользователей совершают покупку и производят успешную оплату в приложении.
  - Большее всего пользователей теряется на 2ом шаге воронки - **`OffersScreenAppear`** , данное событие совершает только **61% пользователей** от общего количества пользователей совершивших предыдущее событие.
  - На остальных шагах воронки потери пользователей не такие серьёзные.
- Результаты А/Б - тестирования показали, что *новые шрифты, которые показывали пользователям из группы `Б` никак не повлияли на поведение пользователей внутри приложения и доля пользователей, совершивших покупки осталась практически без изменений*.