Проект: "A/A/B-тест изменения шрифта в мобильном приложении"

Вводные данные:
Мы работаем в стартапе, который продаёт продукты питания. Нужно разобраться, как ведут себя пользователи нашего мобильного приложения.
Дизайнеры захотели поменять шрифты во всём приложении, а менеджеры выразили опасения, что пользователям будет непривычно. Договорились принять решение по результатам A/A/B-теста.
Пользователей разбили на 3 группы:

- две контрольные со старыми шрифтами
- одну экспериментальную — с новыми

Решение о создании двух групп A вместо одной принято из-за следующих преимуществ. Если две контрольные группы окажутся равны, мы сможем быть уверены в точности проведенного тестирования. Если же между значениями A и A будут существенные различия, это поможет обнаружить факторы, которые привели к искажению результатов. Сравнение контрольных групп также поможет понять, сколько времени и данных потребуется для дальнейших тестов.
Цель проекта Дать ответ - какой шрифт в приложении лучше: старый или новый.
Задачи проекта:

- Изучить воронку продаж.
- Узнать, как пользователи доходят до покупки.
- Ответить на вопросы:
    - Сколько пользователей доходит до покупки, а сколько — «застревает» на предыдущих шагах?
    - На каких именно шагах застревают пользователи?
- После этого исследовать результаты A/A/B-эксперимента.
- Выяснить, какой шрифт лучше старый (две контрольные группы) или новый (экспериментальная группа).

План работы:

- **Шаг № 1** Откроем файл с данными и изучим общую информацию

1. Загрузка библиотек для работы
2. Описание данных
3. Загрузка файла с данными и изучение общей информации
    - **Шаг № 2.** Подготовим данные
4. Замена названия столбцов на удобные для понимания
5. Проверка данных на пропуски и типы данных
6. Работа с пропусками(при необходимости)
7. Корректировка типов данных, если нужно
8. Добавление столбца даты и времени и добавление отдельного столбца дат
    - **Шаг №3.** Изучим и проверим данные
9. Сколько всего событий в логе?
10. Сколько всего пользователей в логе?
11. Сколько в среднем событий приходится на пользователя?
12. Данными за какой период мы располагаем?
13. Найдем максимальную и минимальную дату.
14. Изучим как меняется количество данных:
15. Построим столбчатую диаграмму с количеством событий в зависимости от времени в разрезе групп.
16. Дать ответ - есть уверенность что мы имеем одинаково полные данные за весь период?
17. Определить, с какого момента данные полные и отбросить более старые.
18. Установить за какой период времени мы располагаем данными на самом деле?
19. Много ли событий и пользователей мы потеряли, отбросив старые данные?
20. Проверить, что у нас есть пользователи из всех трёх экспериментальных групп.
    - **Шаг 4** - Изучим воронку событий
21. Посмотрим, какие события есть в логах, как часто они встречаются. Отсортируем события по частоте.
22. Посчитаем, сколько пользователей совершали каждое из этих событий. Отсортируем события по числу пользователей. Посчитаем долю пользователей, которые хоть раз совершали событие.

23. Предположим, в каком порядке происходят события. Все ли они выстраиваются в последовательную цепочку? Их не нужно учитывать при расчёте воронки.
24. По воронке событий посчитаем, какая доля пользователей проходит на следующий шаг воронки (от числа пользователей на предыдущем). То есть для последовательности событий A → B → C посчитаем отношение числа пользователей с событием B к количеству пользователей с событием A, а также отношение числа пользователей с событием C к количеству пользователей с событием B.
25. Узнаем на каком шаге теряете больше всего пользователей?
26. Какая доля пользователей доходит от первого события до оплаты?
    - **Шаг 5.** Изучим результаты эксперимента
27. Сколько пользователей в каждой экспериментальной группе?
28. Есть 2 контрольные группы для A/A-эксперимента, чтобы проверить корректность всех механизмов и расчётов. Проверим, находят ли статистические критерии разницу между выборками 246 и 247.
29. Напишем функцию для проверки: Выберем событие. Посчитаем число пользователей, совершивших это событие в каждой из контрольных групп. Посчитаем долю пользователей, совершивших это событие. Проверм, будет ли отличие между группами статистически достоверным.
30. Применим функцию для проверки для самого популярного события
31. применим функцию для проверки для всех других событий
32. Ответим на вопрос - можно ли сказать, что разбиение на группы работает корректно?
33. Аналогично поступим с группой с изменённым шрифтом. Сравним результаты с каждой из контрольных групп в отдельности по каждому событию. Сравним результаты с объединённой контрольной группой. Какие выводы из эксперимента можно сделать?
34. Написать какой уровень значимости мы выбрали при проверке статистических гипотез выше? Посчитать, сколько проверок статистических гипотез мы сделали. Иметь в виду, что при уровне значимости 0.1 в 10% случаев можно ошибочно отклонить нулевую гипотезу при условии, что она верна. Ответить на вопрос -какой уровень значимости стоит применить? Правильно ли мы выбрали уровень значимости. При необходимости изменения повторить предыдущие пункты и проверьте свои выводы.

1 Загрузка и изучение общей информации о данных

1.1 Описание данных

Путь к файлу:

- /datasets/logs_exp.csv
- **Описание**

Каждая запись в логе — это действие пользователя, или событие.

- EventName — название события;
- DeviceIDHash — уникальный идентификатор пользователя;
- EventTimestamp — время события;
- ExpId — номер эксперимента: 246 и 247 — контрольные группы, а 248 — экспериментальная.

1.2 Библиотеки для работы

In [1]:

```
import pandas as pd #
# pd.set_option('max_colwidth', 150) # увеличение ширины столбца
import numpy as np #
import datetime as dt
from datetime import datetime, timedelta

import scipy.stats as stats
from scipy import stats
from scipy import spatial
from scipy.stats import norm

import matplotlib
from matplotlib import pyplot as plt
%matplotlib inline

from plotly import graph_objects as go
```

```
#import plotly.express as px
#from plotly.offline import iplot

import seaborn as sns
```

1.3 Загрузка данных

In [2]:

```
try:
    df = pd.read_csv('https://вырезано', sep='\t')

except:
    df = pd.read_csv('/datasets/logs_exp.csv', sep='\t')
df.head(1)
```

Out[2]:

	EventName	DeviceIDHash	EventTimestamp	ExpId
0	MainScreenAppear	4575588528974610257	1564029816	246

1.4 Изучим общую информацию

In [3]:

```
print(f'Сводная информация о таблице :\n{df.info()}')
print('Выведем несколько случайных строк из таблицы для изучения:')
display(df.sample(n=5, random_state=1))
```

```
<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
Сводная информация о таблице :
None
Выведем несколько случайных строк из таблицы для изучения:
```

	EventName	DeviceIDHash	EventTimestamp	ExpId
120873	MainScreenAppear	9096061512161785646	1564917596	248
4103	OffersScreenAppear	1074933668878523996	1564631437	247
92724	OffersScreenAppear	8115340604306569112	1564838447	246
128891	CartScreenAppear	3163109716415429772	1564930563	246
186942	OffersScreenAppear	2405626918155135838	1565082344	247

1.4.1 Резюме по итогам ознакомления с данными

Датасет хранит 244126 строк с данными о логах событий приложения в 4-х столбцах:

- наименование события,
- ID пользователя,
- информацию о времени события
- номером эксперимента (группы).

Названия столбцов не соответствуют стилю, требуется замена на нижний регистр и применение змеиного стиля написания. Также необходима замена типа данных в графе с информацией о дате и времени события.

2 Подготовка данных

Подготовим данные к анализу, для этого произведем замена названия столбцов в соответствии со змеиным стилем написания и к нижнему регистру. Произведем проверку данных на пропуски, заменим тип данных в колонке, содержащей информацию о времени события.
Добавим столбец даты и времени с названием datetime и добавим столбец хранящий дату с названием date.

2.1 Проверим на дубликаты и пропуски

Произведем проверку. При наличии дубликатов, удалим... При наличии пропусков...

In [4]:

```
print(f'Датафрейм содержит строк: {len(df)}.\n\
В таблице количество явных дубликатов: {df.duplicated().sum()} шт.\n\
В таблице количество пропусков: {df.isnull().values.sum()} шт.')
```

Датафрейм содержит строк: 244126.
В таблице количество явных дубликатов: 413 шт.
В таблице количество пропусков: 0 шт.

2.1.1 Удалим дубликаты

In [5]:

```
df = df.drop_duplicates().reset_index(drop=True)
print(f'После удаления в таблице количество явных дубликатов: {df.duplicated().sum()} шт.')
```

После удаления в таблице количество явных дубликатов: 0 шт.

2.1.2 Резюме по дубликатам и пропускам в таблице

Пропусков в таблице не обнаружено. Явные дубликаты обнаружены и удалены в количестве 413 штук.

2.2 Добавим столбцы в df

Преобразуем столбец EventTimestamp в формат даты и времени.
С помощью метода strftime() отдельно добавим столбец хранящий только дату с названием date.

In [6]:

```
df['EventTimestamp'] = pd.to_datetime(df['EventTimestamp'], unit='s')
df['date'] = df['EventTimestamp'].dt.strftime('%Y-%m-%d')
display(df.sample(n=3, random_state=10))
```

	EventName	DeviceIDHash	EventTimestamp	ExpId	date
98344	MainScreenAppear	3416181679258622505	2019-08-03 15:46:24	247	2019-08-03
128293	MainScreenAppear	7241586465157505325	2019-08-04 14:45:09	246	2019-08-04
118170	OffersScreenAppear	4758223822167675176	2019-08-04 10:20:44	248	2019-08-04

2.3 Переименуем названия

2.3.1 Переименуем название столбцов датафреймов.

Для приведения написания заголовков колонок в змеином регистре и строчными буквами, а также для удобства визуального восприятия названий столбцов с его содержимым, явно переименуем столбцы.
После выведем список с названием столбцов в датафрейме и убедимся, что они теперь соответствует стилю написания.

In [7]:

```
print('Наименования столбцов до переименования в таблице orders:\n', list(df))
df.columns = ['event_name', 'user_id', 'datetime', 'group_id', 'date']
print('Названия столбцов датафрейма orders после изменения:\n', list(df.columns))
```

Наименования столбцов до переименования в таблице orders:
['EventName', 'DeviceIDHash', 'EventTimestamp', 'ExpId', 'date']
Названия столбцов датафрейма orders после изменения:
['event_name', 'user_id', 'datetime', 'group_id', 'date']

2.4 Переименуем группы в A1 A2 и B

Для дальнейшего визуального удобства переименуем экспериментальные группы следующим образом:

- A1 - контрольная группа номер эксперимента 246
- A2 - контрольная группа номер эксперимента 247
- B - экспериментальная гуппаномер эксперимента 248

Напишем функцию для замены наименований. Если в таблицу попали результаты других экспериментов (значение id не соответствует id наших экспериментов 246,247 или 248), то функция выдаст предупреждение.

In [8]:

```
# напишем функцию для переименования id эксперимента в явное название групп A1, A2 и B
def a_b_name(id_group):
    if id_group == 246:
        return 'A1'
    elif id_group == 247:
```

```
        return 'A2'
    elif id_group == 248:
        return 'B'
    else:
        print('Ahtung! В таблицу попали результаты других экспериментов.')

#применим функцию def a_b_name(id_group) к датафрейма df и запишем результат в столбец

df['group_id'] = df['group_id'].apply(a_b_name)
df.sample(n=3, random_state=1)
```

Out[8]:

	event_name	user_id	datetime	group_id	date
242800	MainScreenAppear	7714956624707119480	2019-08-07 20:06:20	A1	2019-08-07
57104	MainScreenAppear	3642642126195324639	2019-08-02 12:58:34	B	2019-08-02
91886	MainScreenAppear	7838054161069641474	2019-08-03 13:00:23	B	2019-08-03

2.4.1 Резюме по итогам переименования

Группы экспериментов теперь визуально идентифицируются лучше, не нужно сверяться со списком id эксперимента, чтобы понять о какой группе идет речь - контрольной или экспериментальной.
Заголовки переименовали в соответствие со стилем в змеином регистре и строчными буквами, а также для визуального удобства восприятия.
Список с наименованиями столбцов после переименования и добавления новых столбцов. С описанием данных, которые они содержат:

- 'event_name' - наименование события,
- 'user_id' - ID пользователя,
- 'datetime' - дата и время события
- 'group_id' - группа эксперимента после переименования в A1, A2 или B, где им соответствуют номера экспериментов:
  - A1 - контрольная группа номер эксперимента 246
  - A2 - контрольная группа номер эксперимента 247
  - B - экспериментальная группа номер эксперимента 248(группы).
- 'date' - дата события

2.5 Проверка на пересечения пользователей в группе

Проверим отсутствие пересечения пользователей в группах A1, A2 и B.
Попавший в одну группу пользователь остается в ней до конца теста.
Если таких пересечений не обнаружится, значит инструмент деления пользователей на группы тестирования сработал без ошибок.

In [9]:

```
df.groupby('user_id', as_index=False).agg({'group_id':'nunique'}).query('group_id > 1')
```

Out[9]:

	user_id	group_id

In [10]:

```
print('Всего уникальных пользователей:', df['user_id'].nunique())
count = 0
for i in df['group_id'].unique():
    data_users = df[df['group_id'] == i]['user_id'].nunique()
    count = count + data_users
    print(f'В группе {i}  уникальных пользователей: {data_users}.')
print(f"Сложим количество пользователей групп A1, A2, B получим. {count}\n\
\nРазница между количеством уникальных пользователей и \n\
результатом сложения пользователей из групп A1, A2, B равна:{(count -
df['user_id'].nunique())}.")
```

```
Всего уникальных пользователей: 7551
В группе A1  уникальных пользователей: 2489.
В группе B  уникальных пользователей: 2542.
В группе A2  уникальных пользователей: 2520.
```

Сложим количество пользователей групп A1, A2, B получим. 7551

Разница между количеством уникальных пользователей и
результатом сложения пользователей из групп A1, A2, B равна:0.

2.5.1 Резюме проверки на пересечения пользователей в группе

Проверка показала отсутствие пересечения пользователей в группах A1, A2 и B. Приступим к изучению и проверки данных.

3 Изучим и проверим данные о логах по сырым данным

3.1 Изучим информацию о логах по сырым данным

Узнаем следующую информацию:

- Сколько всего событий в логе
- Сколько всего пользователей в логе?
- Сколько в среднем событий приходится на пользователя
- Данными за какой период мы располагаем
- Найдем максимальную и минимальную дату.

3.1.1 Запишем в переменные:

- users - количество уникальных пользователей по сырым данным
- events_sum- Сколько всего событий в логе по сырым данным
- events_median - медианное количество событий на одного пользователя по сырым данным
- events_mean - среднее количество событий на одного пользователя по сырым данным

In [11]:

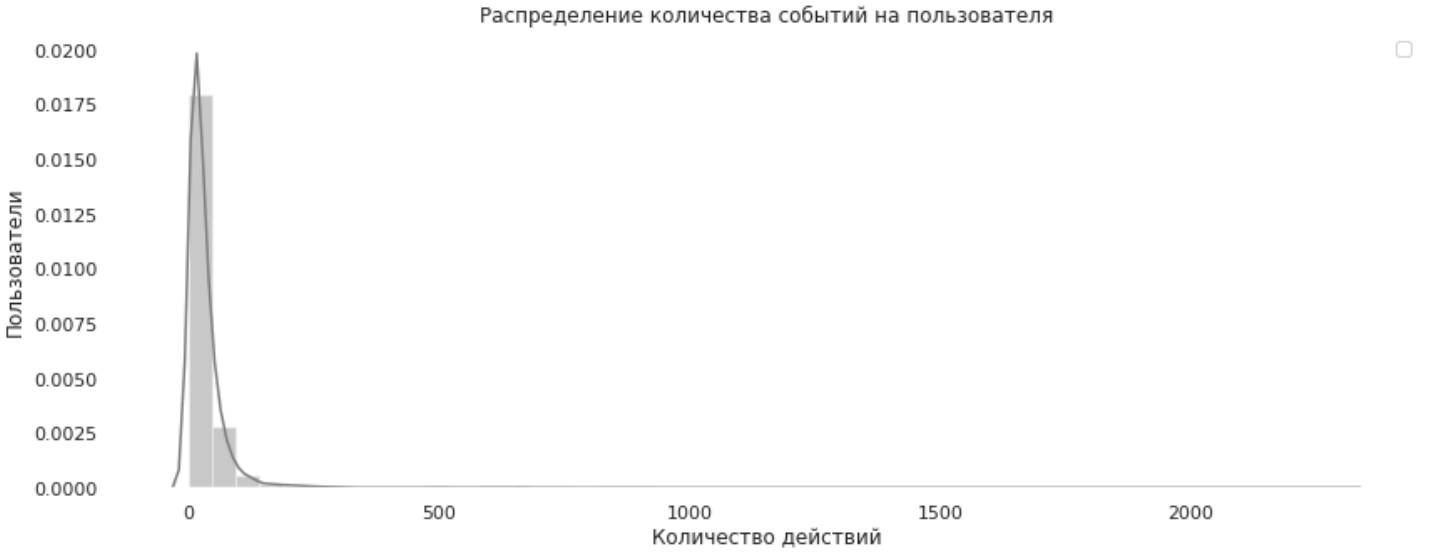
```
users = df['user_id'].nunique() #
events_sum = df.event_name.count()
events_median = int(df.groupby('user_id')['event_name'].agg('count').median())
events_mean = int(df.groupby('user_id')['event_name'].agg('count').mean())
```

3.1.2 Распределение событий на одного пользователя

Посмотрим как распределены события на одного пользователя на графике.

In [12]:

```
sns.set (rc={'axes.facecolor':'white', 'figure.facecolor':'white'})
plt.figure(figsize=(14, 5))
plt.title('Распределение количества событий на пользователя')
sns.distplot(df.groupby('user_id').agg(event_count =('event_name', 'count')), color='grey'
plt.legend()
plt.grid()
plt.xlabel('Количество действий')
plt.ylabel('Пользователи')
plt.show();
```



3.1.3 Получим информацию о логах

- Сколько всего событий в логе.
- Сколько всего пользователей в логе?
- Сколько в среднем событий приходится на пользователя
- Данными за какой период мы располагаем
- Найдем максимальную и минимальную дату.

In [13]:

```
print(f"Всего событий в логе {events_sum}.\
\nВсего уникальных пользователей в логе {users}.\
\nНа одного пользователя медианное количество событие равно {events_median}.\
\nНа одного пользователя среднее количество событие равно {events_mean}.\
\nРазница в количестве событий по среднему и медиане \
составила {round((events_median / events_mean), 2) * 100} %.\
\nМы располагаем данными за период с {df['datetime'].min()} по {df['datetime'].max()}\
\nМаксимальная дата {df['datetime'].min()}.\nМинимальная дата{df['datetime'].max()}")
```

Всего событий в логе 243713.
Всего уникальных пользователей в логе 7551.
На одного пользователя медианное количество событие равно 20.
На одного пользователя среднее количество событие равно 32.
Разница в количестве событий по среднему и медиане составила 62.0 %.
Мы располагаем данными за период с 2019-07-25 04:43:36 по 2019-08-07 21:15:17
Максимальная дата 2019-07-25 04:43:36.
Минимальная дата2019-08-07 21:15:17

3.1.4 Итоги изучения и проверки данных о логах по сырым данным.

Распределение количества событий на одного пользователя не нормально. Для оценки среднего количества действий на пользователя возьмем медианное значение.
Всего по сырым данным:

- всего совершено событий 243713,
- всего уникальных пользователей 7551,
- в среднем приходитмя 20 событий на пользователя (по медиане)
- мы располагаем данными за период с 2019-07-25 04:43:36 по 2019-08-07 21:15:17
- Максимальная дата 2019-07-25 04:43:36
- Минимальная дата2019-08-07 21:15:17.

Теперь изучим как меняется количество данных.

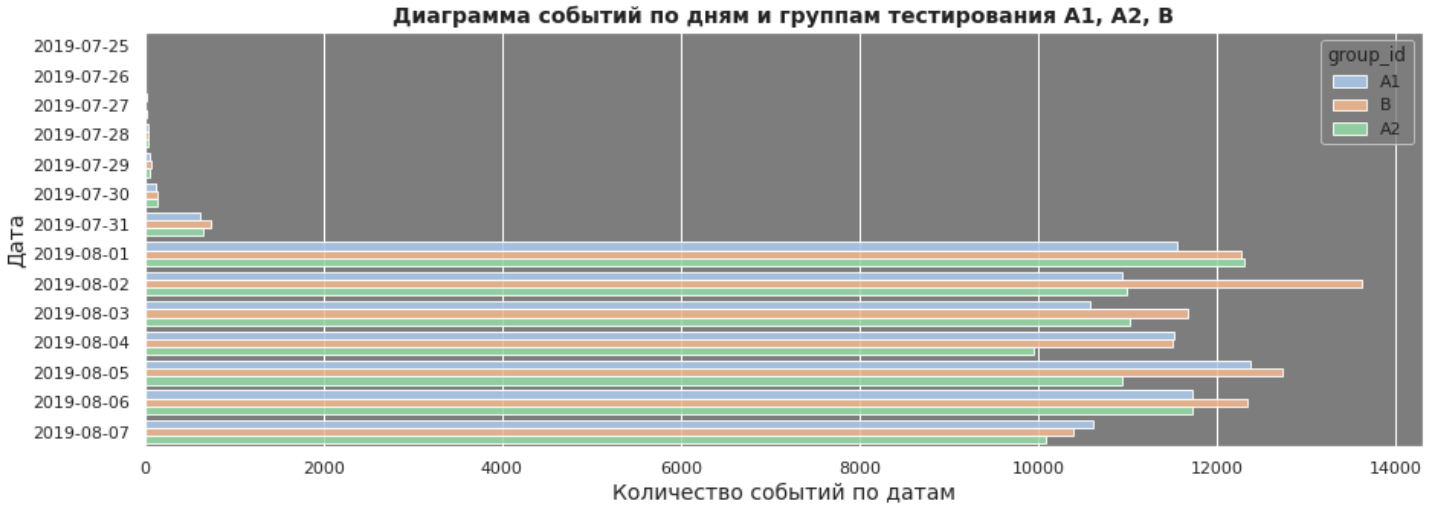
3.2 Диаграмма количества событий в зависимости от времени в групп тестирования: A1, A2, B

Изучим, как меняется количество данных. Для этого построим столбчатую диаграмму, которая отобразит количество событий в зависимости от даты в разрезе групп. Для отбора возьмем столбец с датой, без времени, так как некомфортно изучать график с большим количеством столбцов.

In [14]:

```
plt.figure(figsize=(15,5))
sns.set (rc={'axes.facecolor':'grey', 'figure.facecolor':'white'})
ax = sns.countplot(y=df['date'], data=df, hue='group_id', alpha=0.9, palette="pastel") #,
dodge=False)
ax.set_title('Диаграмма событий по дням и группам тестирования A1, A2, B', fontsize = 14,
          fontweight ='bold')

plt.ylabel('Дата', fontsize = 14)
plt.xlabel('Количество событий по датам', fontsize = 14)
plt.show();
```



3.2.1 Резюме по диаграмме количества событий по датам в разрезе групп тестирования: A1, A2, B

Видим как меняется количество событий по дням. Высокая активность с 1 по 7 августа. Очень слабая активность с 25по 30 июля, сильнее чем в начале тестно, но все равное еще слабая активность 31 июля. Заметим, что активностей хватает у всех трех групп тестирования. Можно заметить, что событий у группы B немного больше в какие-то дни, а 2 августа почти на 3 тыс событий больше, чем у групп A1 и A2.
Технически в логи новых дней по некоторым пользователям могут «доезжать» события из прошлого — это может «перекашивать данные». Поэтому построим столбчатую диаграмму, которая отобразит количество событий в зависимости от времени в разрезе групп. Посмотрим на Так как данные содержат большое количество дат, выведем для начала столбчатую диаграмму укрупненно, с количеством событий по дням, а также посмотрим активности по дням в разрезе групп.

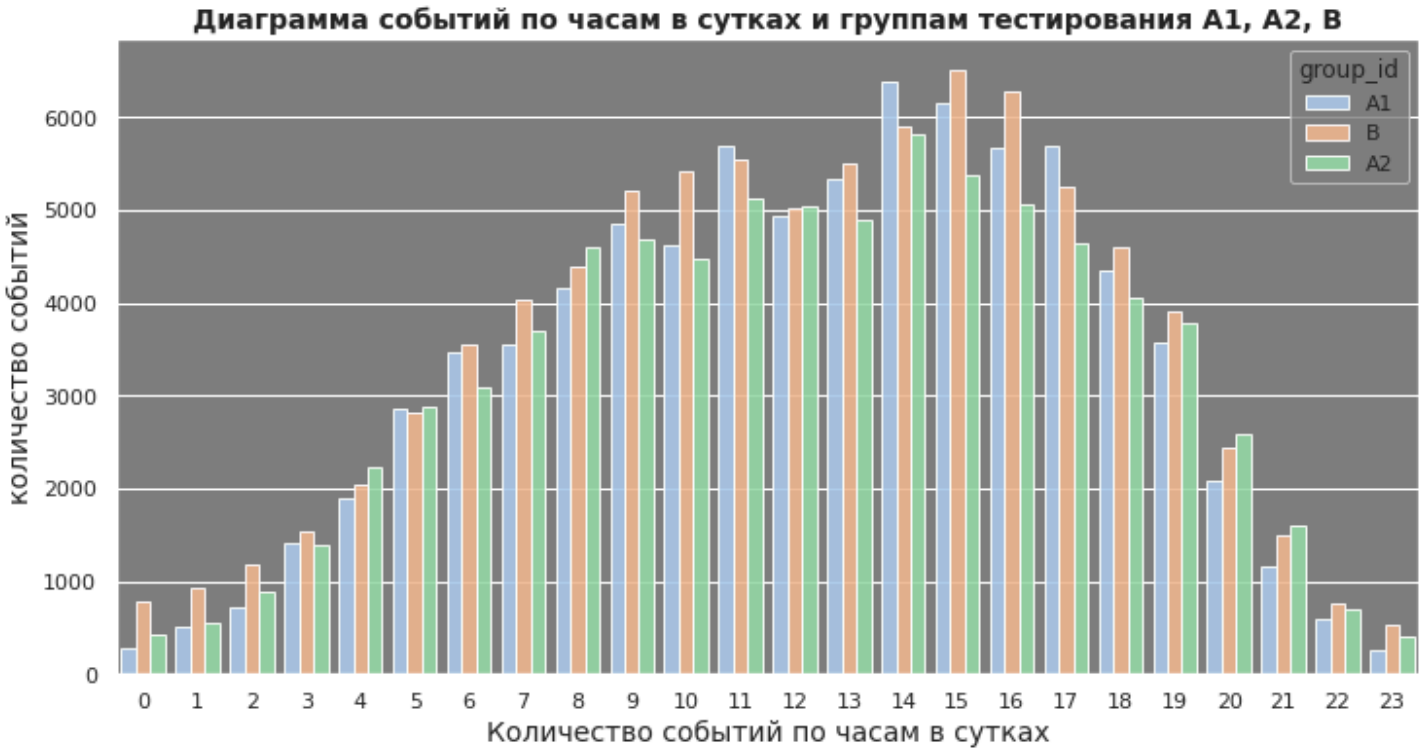
3.2.2 Распределение событий по часам

Посмотрим, как распределены события по времени суток. Визуализируем диаграмму событий по времени суток в разрезе групп тестирования, а также без сортировки по группам тестирования.

In [15]:

```
plt.figure(figsize=(12, 6))
sns.set (rc={'axes.facecolor':'grey', 'figure.facecolor':'white'})
ax = sns.countplot(x=df['datetime'].dt.hour, data=df, hue='group_id', alpha=0.9,
palette="pastel")
ax.set_title('Диаграмма событий по часам в сутках и группам тестирования A1, A2, B',
             fontsize = 14,
             fontweight ='bold')

plt.ylabel('количество событий', fontsize = 14)
plt.xlabel('Количество событий по часам в сутках', fontsize = 14)
plt.show();
```



In [16]:

```
plt.title('Распределение событий по часам')
df['datetime'].dt.hour.hist(bins=24, figsize=(15, 5), alpha=0.4, color='red')
plt.grid(False)
plt.xticks(range(0, 24))
plt.show()
```

3.2.3 Резюме по графику распределения событий по времени.

Заметна периодичность во времени: в ночные часы событий заметно еньше, чем днём/ Минимальная активность не снижается ниже 1000 за час.

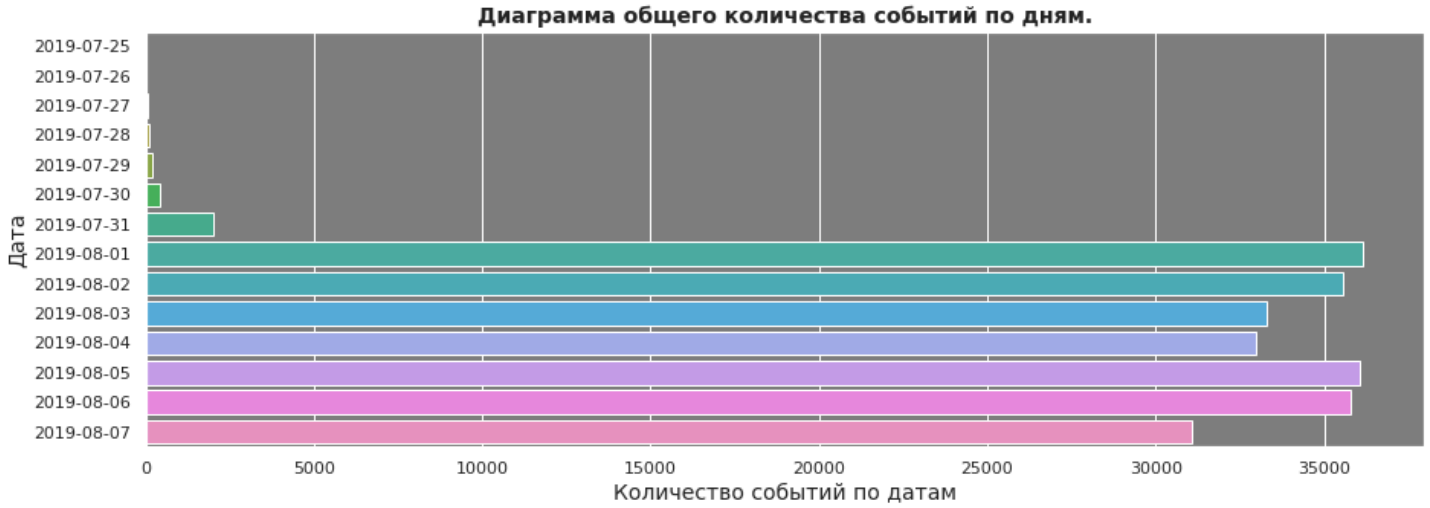
3.2.4 Диаграмма общего количества событий без группировки по виду события

Визуализируем общее количество событий по дням на столбчатой диаграмме. Так мы увидим распределение событий по дням.

In [17]:

```
plt.figure(figsize=(15,5))
ax = sns.countplot(y=df['date'], data=df, dodge=False)
ax.set_title('Диаграмма общего количества событий по дням.', fontsize = 14,
         fontweight ='bold')

plt.ylabel('Дата', fontsize = 14)
plt.xlabel('Количество событий по датам', fontsize = 14)
plt.show();
```



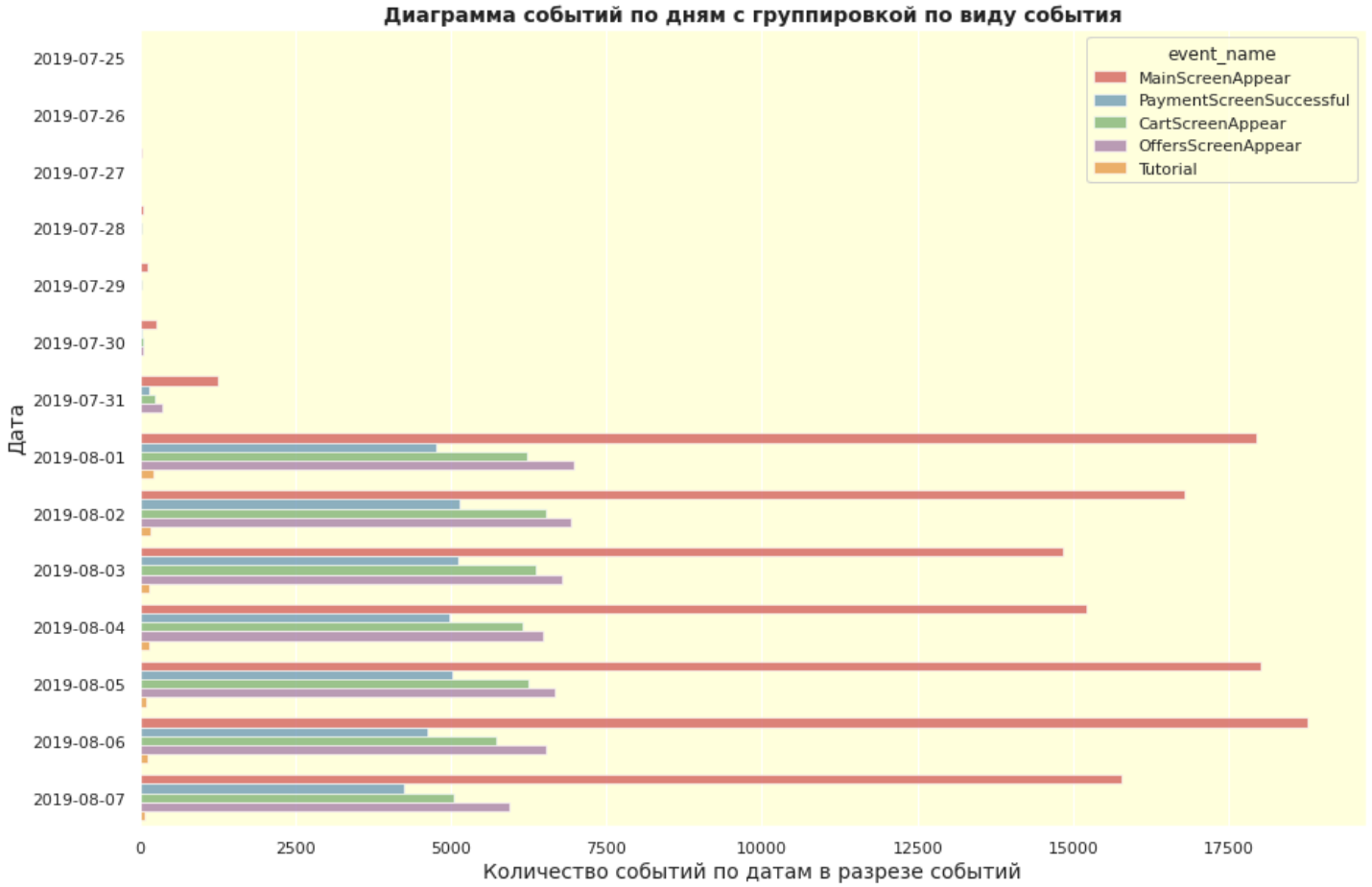
3.2.5 Диаграмма событий по дням с группировкой по виду события

Разобъем все события не только по дням, но и по виду совершенного события.

In [ ]:

In [18]:

```
plt.figure(figsize=(15, 10))
sns.set (rc={'axes.facecolor':'lightyellow', 'figure.facecolor':'white'})
ax = sns.countplot(y=df['date'],  data=df, hue='event_name', alpha=0.6, palette="Set1")
ax.set_title('Диаграмма событий по дням с группировкой по виду события',
             fontsize = 14,
             fontweight ='bold')
plt.ylabel('Дата', fontsize = 14)
plt.xlabel('Количество событий по датам в разрезе событий', fontsize = 14)
plt.show()
```



3.2.6 Резюме по итогам вывода диаграмм количества событий

Всего в день при высокой активности у нас видно от 31 до 36 тысяч разных активностей в сутки.

Следует иметь в виду, что на диаграмме в разрезе вида события столбика диаграммы показывает не общее количество событий за день, а именно персонального события.
На диаграмме с группировкой по активностям начиная с 1 августа можно увидеть виды активностей и их количество во сутки:

- MainScreenAppear - самая высокая активность у этого события, его количество колеблется от 15тыс до 17.5тыс в день.
- OffersScreenAppear - колеблется в районе 6-7тыс активностей в сутки
- CartScreenAppear - эта активность в пределах 5-6 тыс в сутки
- PaymentScreenSuccessful - от 4,5 до 5 тыс в сутки
- Tutorial - можно предположить не более сотни активностей в сутки, масштаб графика не позволяет увидеть детальнее. Это самая малочисленная активность.

Так как стабильная активность пользователей началась с 1 августа можно предположить, что у нас не все данные за период, предшествующий 1 августа. Заметно, что активность начала заметно возрастать 31 июля.
Так как данные в датафрейме нам даны не только по датам, но и по времени события, то теперь проверим в какое время произошло возрастание активности 31 июля 2019 г. и найдем дату и время отсчета активностей для анализа

In [19]:

```
print('Список уникальных событий', (list(df['event_name'].unique())))
```

Список уникальных событий ['MainScreenAppear', 'PaymentScreenSuccessful', 'CartScreenAppear', 'OffersScreenAppear', 'Tutorial']

3.3 Найдем дату и время отсчета активностей для анализа

Укрупним вывод на диаграмму событий по часам и посмотрим в какое время происходили события 31 июля. Нет смысла смотреть более ранние даты, так как в этот период была очень низкая активность у пользователей. Поищем время начала большого числа активностей, чтобы использовать его в анализе.
Получим срез по дате 31 июля и построим гистограмму событий по времени, укрупнив время до часов - найдем время, которое послужит нам точкой отсчета, с которой мы начнем анализ.

In [20]:

```
df_31 = df.loc[((df['date'] == '2019-07-31'))]
df_31.head(3)
```

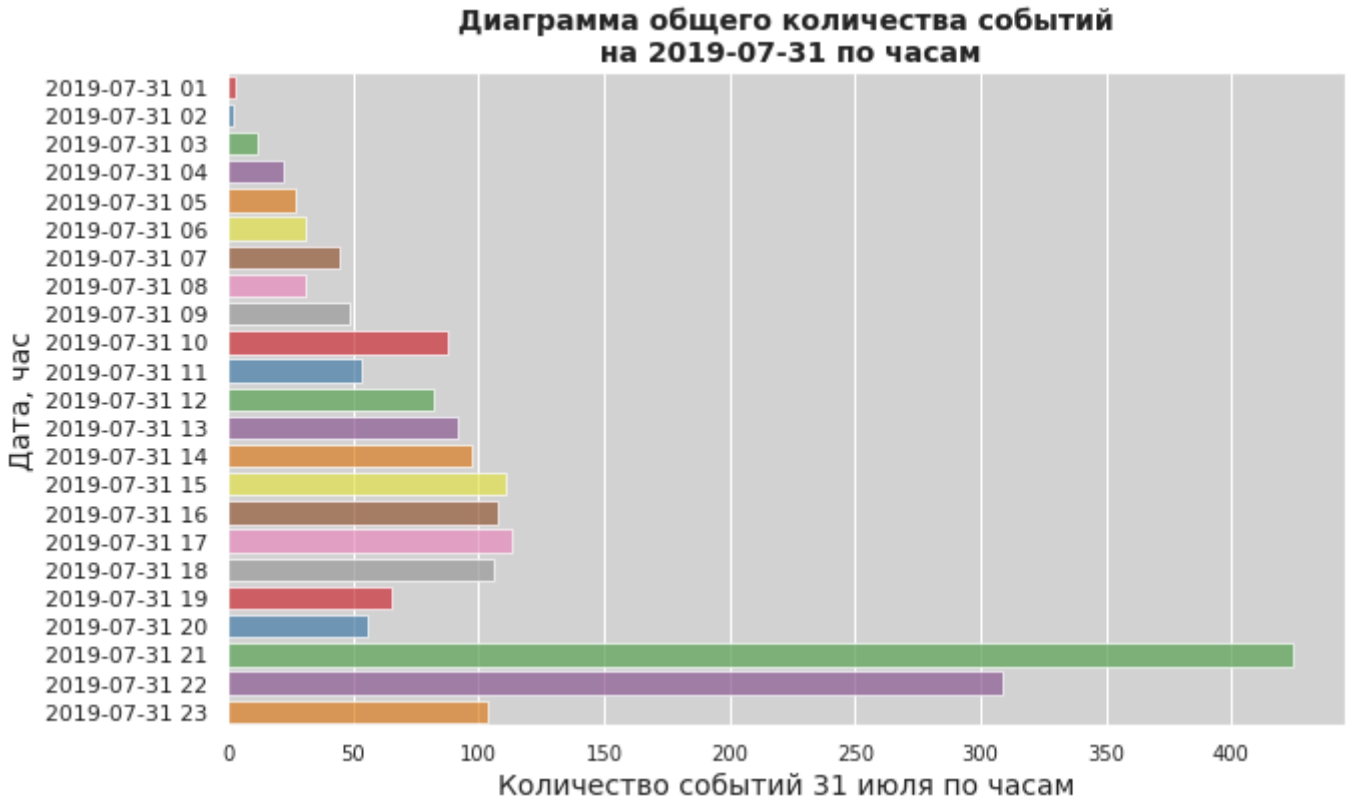
Out[20]:

	event_name	user_id	datetime	group_id	date
796	MainScreenAppear	3670880358399219515	2019-07-31 01:11:46	A2	2019-07-31
797	OffersScreenAppear	3799109751993694887	2019-07-31 01:21:04	B	2019-07-31
798	Tutorial	1126021718529336913	2019-07-31 01:32:11	A2	2019-07-31

In [21]:

```
plt.figure(figsize=(10, 6))
sns.set (rc={'axes.facecolor':'lightgrey', 'figure.facecolor':'white'})
ax = sns.countplot(y=df_31['datetime'].dt.strftime('%Y-%m-%d %H'), data=df_31, dodge=False
alpha=0.7, palette="Set1")
ax.set_title('Диаграмма общего количества событий\n на 2019-07-31 по часам', fontsize = 14
          fontweight ='bold')

plt.ylabel('Дата, час', fontsize = 14)
plt.xlabel('Количество событий 31 июля по часам', fontsize = 14)
plt.xticks(fontsize = 10);
plt.show();
```



3.3.1 Определим момент полных данных

На графике видно, что активность 31 июля резко возросла в 21 час до 450 действий. Ранее мы строили график, который показал, что активность в самые спокойные часы была не ниже 1000 действий в час. Всеже активности 31 июля не хватае, чтобы считать ее полной.
Поэтому моментом полных данных определим как 2019-08-01 00:00:00. Все события и данные, которые произошли до 2019-08-01 00:00:00. удалим, чтобы не искажать метрики.

3.4 Изучим и проверим данные о логах по обработанным данным

3.4.1 Удалим неполные данные

Отбросим все данные, которые предшествовали 2019-08-01 00:00:00. и запишем данные в таблицу df_new. Этот датафрейм будем использовать для анализа очищенных данных.

In [22]:

```
df_new = df[df['datetime'] >= '2019-08-01 00:00:00']
```

3.4.2 Изучим информацию о логах по обработанным данным

Узнаем следующую информацию:

- Сколько всего событий в логе по обработанным данным.
- Сколько всего пользователей в логе по обработанным данным.
- Сколько в среднем событий приходится на пользователя по обработанным данным.
- Данными за какой период мы располагаем после обработки данных.
- Какое количество пользователей потеряно при обработке.
- Какое количество событий потеряно при обработке.

In [23]:

```
users_new = df_new['user_id'].nunique() #
events_sum_new = df_new.event_name.count()
events_median_new = int(df_new.groupby('user_id')['event_name'].agg('count').median())
events_mean_new = int(df_new.groupby('user_id')['event_name'].agg('count').mean())
```

In [24]:

```
print(f"Всего событий в логе после обработки {events_sum_new} ({events_sum} до обработки)\
\nПосле обработки потеряно {events_sum - events_sum_new} событий или \
{round((events_sum / events_sum_new - 1), 5) * 100} %.\n\
\nВсего уникальных пользователей в логе после обработки {users_new} ({users} до
обработки).\
\nПосле обработки потеряно {users- users_new} пользователей или \
{round((users/users_new -1), 3) * 100} %.\n\
```

```
\nНа одного пользователя медианное количество событий равно {events_median_new}.\
\nНа одного пользователя среднее количество событий равно {events_mean_new}.\n\
\nМы располагаем полными данными за период с {df_new['datetime'].min()} по
{df_new['datetime'].max()}\
\nМаксимальная дата {df_new['datetime'].min()}.\nМинимальная
дата{df_new['datetime'].max()}.")
```

Всего событий в логе после обработки 240887 (243713 до обработки)
После обработки потеряно 2826 событий или 1.173 %.

Всего уникальных пользователей в логе после обработки 7534 (7551 до обработки).
После обработки потеряно 17 пользователей или 0.2 %.

На одного пользователя медианное количество событий равно 19.
На одного пользователя среднее количество событий равно 31.

Мы располагаем полными данными за период с 2019-08-01 00:07:28 по 2019-08-07 21:15:17
Максимальная дата 2019-08-01 00:07:28.
Минимальная дата 2019-08-07 21:15:17.

3.4.3 Итоги изучения и проверки данных о логах по обработанным данным.

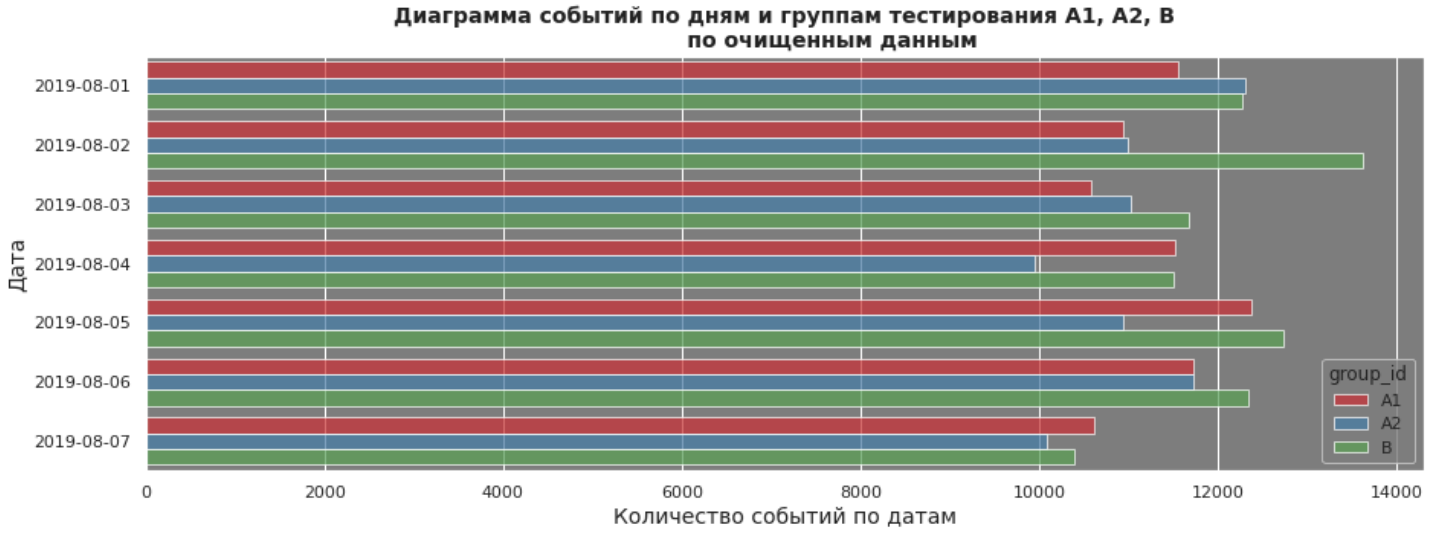
Мы удалили целую неделю исследований. При этом потери оказались минимальными - событий потеряно чуть более 1%, количество пользователей сократилось на 0,2%.
У нас есть данные о 7,5 тыс пользователях и почти о 241тыс событий за неделю, а точнее за период с 2019-08-01 00:07:28 по 2019-08-07 21:15:17.
На одного пользователя медианное количество событий 19. Потеряли 1 пользователя после удаления неполных данных.
Посмотрим еще раз на диаграмму распределения событий по группам исследования.

3.5 Диаграмма событий по дням с группировкой по виду события по очищенным данным

In [25]:

```
plt.figure(figsize=(15,5))
sns.set (rc={'axes.facecolor':'grey', 'figure.facecolor':'white'})
ax = sns.countplot(y=df_new['date'], data=df_new, hue='group_id', alpha=0.7,
palette="Set1")
ax.set_title('Диаграмма событий по дням и группам тестирования A1, A2, B\n\
            по очищенным данным', fontsize = 14,
            fontweight ='bold')

plt.ylabel('Дата', fontsize = 14)
plt.xlabel('Количество событий по датам', fontsize = 14)
plt.show();
```



3.5.1 Резюме по диаграмме количества событий по группам тестирования. Очищенные данные.

Нет оснований полагать, что данных по какой-то из групп не хватает, так как активностей хватает у всех трех групп тестирования. Можно заметить, что событий у группы B немного больше 2, 5 и 6 августа, чем у групп A1 и A2.

3.6 Выводы по разделу

В ходе изучения данных, было обнаружено, что в первую неделю количество событий составилоо чуть более 1% от количества событий за все время наблюдений.
Мы рассмотрели дату начала бурного роста активности 31 июля и нашли время, с которого пошел этот рост.
Приняли за истину, что 2019-08-01 00:00:00 - дата и время с которого будут взяты данные для анализа.
Таким образом

- максимальная дата анализа 2019-08-01 00:07:28.
- минимальная дата анализа 2019-08-07 21:15:17.

Период и все данные, которые мы посчитали неполными - в анализ не попали.
После обработки потеряно 2826 событий или 1.173 %., а также 17 никальных пользователей или 0.2 %.
На одного пользователя приходится медианное количество событий - 19.
Диаграмма показала наличие событий у всех трех групп пользователей.
Предварительно можно утверждать, что у нас имеются полные данные для анализа. Изучим теперь детальнее поведение пользователей в нашем мобильном приложении.

4 Изучим воронку событий

Посмотрим, какие события есть в логах, как часто они встречаются. Отсортируем события по частоте.
Посчитаем, сколько пользователей совершали каждое из этих событий. Отсортирем события по числу пользователей. Посчитаем долю пользователей, которые хоть раз совершали событие. Предположим, в каком порядке происходят события. Все ли они выстраиваются в последовательную цепочку? Их не нужно учитывать при расчёте воронки.
По воронке событий посчитаем, какая доля пользователей проходит на следующий шаг воронки (от числа пользователей на предыдущем). То есть для последовательности событий A → B → C посчитаем отношение числа пользователей с событием B к количеству пользователей с событием A, а также отношение числа пользователей с событием C к количеству пользователей с событием B. Посморим на каком шаге теряете больше всего пользователей. Какая доля пользователей доходит от первого события до оплаты

4.1 Какие события есть в логах

Посмотрим, какие события есть в логах, как часто они встречаются.

4.1.1 Сгруппируем логи

В таблица events_group - сгруппируем по виду события, посчитаем уникальных пользователей и количество действий, которые они совершили. Отсортируем события по частоте.

In [26]:

```
events_group = df_new.groupby('event_name')['user_id']\
.agg(['count', 'nunique']).reset_index()

events_group.columns =['event_name', 'events_sum', 'users_uniq']
```

```
print('Рейтинг событий по убыванию:\n')
events_group.sort_values(by='events_sum', ascending=False)
```

Рейтинг событий по убыванию:

Out[26]:

	event_name	events_sum	users_uniq
1	MainScreenAppear	117328	7419
2	OffersScreenAppear	46333	4593
0	CartScreenAppear	42303	3734
3	PaymentScreenSuccessful	33918	3539
4	Tutorial	1005	840

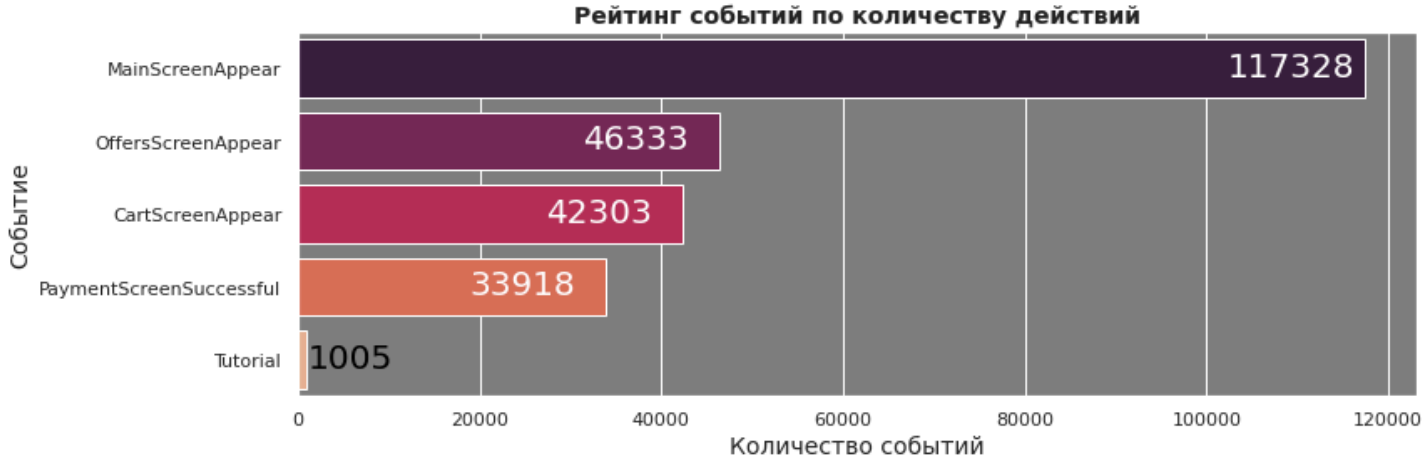
4.1.2 Диаграмма с логами

Визуализируем события в логах на столбчатой диаграмме.

In [27]:

```
plt.figure(figsize=(12, 4))
order = events_group.sort_values(by='events_sum',
ascending=False).reset_index(drop=True)['event_name']
ax = sns.barplot(y='event_name', x='events_sum', data=events_group, order=order,
palette='rocket')

ax.set_title('Рейтинг событий по количеству действий', fontsize = 14, fontweight ='bold')
#добавим количество на каждый столбик
for i in ax.patches:
   if i.get_width() > 30000:
       ax.text(i.get_width() - 15000, i.get_y() + 0.5,
               str(int(i.get_width())), fontsize=20, color='white')
   else:
       ax.text(i.get_width() +30, i.get_y() + 0.5,
               str(int(i.get_width())), fontsize=20, color='black')
plt.xlabel('Количество событий', fontsize = 14)
plt.ylabel('Событие', fontsize = 14)
plt.show()
```



4.1.3 События в логах. Описание.

Дадим определения логам и прокомментируем результаты диаграммы:

- MainScreenAppear - Отображение главного экрана.

Количество действий -117328 раз. Самое популярное действие у пользователей приложения.

- OffersScreenAppear - Отображение экрана предложений.

Количество действий - 46333 раз. Этот экран второй по популярности. При этом видно, что это действие совершается почти в 3 раза меньше, чем отображение главного экрана.

- CartScreenAppear - Отображение экрана корзины.

Количество действий - 42303 раз. Третье по популярности действие. Смотрят на корзину с товарами почти также часто, как и на окно с предложениями товара.

- PaymentScreenSuccessful - Отображение экрана об успешной оплате.

Количество действий - 33918 раз. Экран с успешной оплатой отображается почти на 10тысяч раз меньше, чем экран с корзиной. Это само по себе не так уж плохо. Из корзины можно вернуться на экран с предложением и добавить товар.

- Tutorial - Инструкция к приложению.

Количество действий - 1005 раз. Действий мало. Неудивительно, к инструкции обращаются, предположительно, либо очень педантичные люди, либо когда совсем не получается совершить какое-то действие, ну или возникли конкретные затруднения. Люди не любят читать инструкции. Как один из вариантов - инструкцию не замечают. Отследить это можно при условии, что есть обязательное действие "пропустить инструкцию". Такого в данных не предоставлено.

В любом случае, нашим пользователям, судя по количеству других действий, не помешало неознакомление с инструкцией, чтобы осуществлять навигацию по приложению. Можно предположить, что это действие необязательное.

4.2 Количество пользователей на каждое событие

4.2.1 Сколько пользователей совершали каждое событие

Посчитаем, сколько пользователей совершали каждое событие. Отсортируем события по числу пользователей.

In [28]:

```
print('Рейтинг событий по убыванию:\n')
events_group.sort_values(by='users_uniq', ascending=False)
```

Рейтинг событий по убыванию:

Out[28]:

	event_name	events_sum	users_uniq
1	MainScreenAppear	117328	7419
2	OffersScreenAppear	46333	4593
0	CartScreenAppear	42303	3734
3	PaymentScreenSuccessful	33918	3539
4	Tutorial	1005	840

4.2.2 Диаграмма с пользователями на событие.

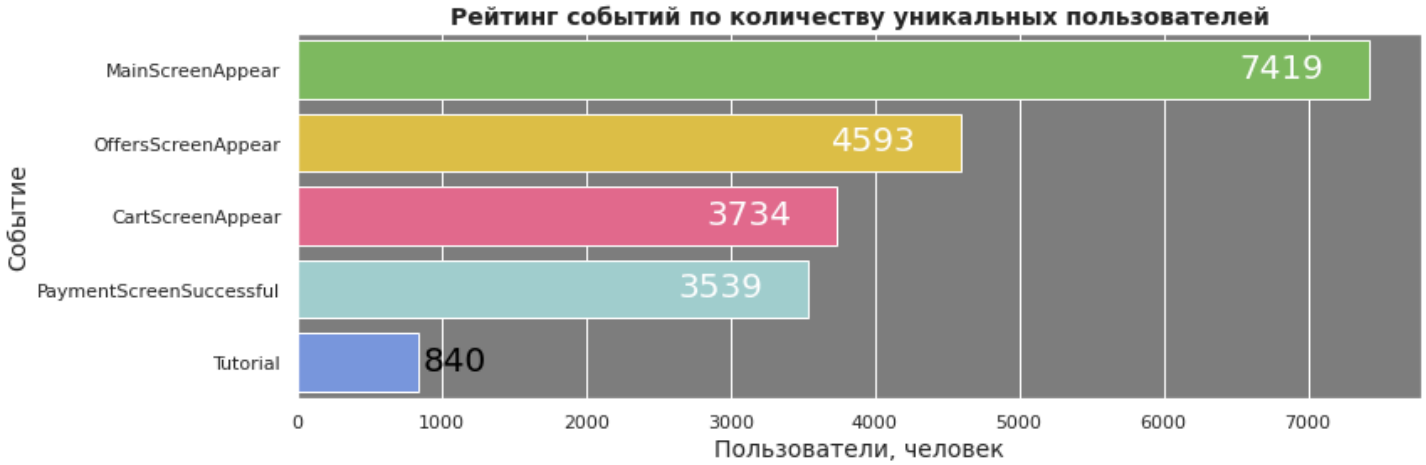
Визуализируем количество пользователей на каждое событие на столбчатой диаграмме.

In [29]:

```
plt.figure(figsize=(12, 4))
#order = events_group.sort_values(by='events_sum',
ascending=False).reset_index(drop=True)['event_name']
colors = ["#78C850", "#F8D030", "#F85888", '#98D8D8', '#6890F0']
# '#78C850', '#F08030', '#6890F0','#F8D030', '#F85888', '#705898', '#98D8D8'
ax = sns.barplot(y='event_name', x='users_uniq', data=events_group, order=order,
palette=colors) #'rocket' # 'mako'

ax.set_title('Рейтинг событий по количеству уникальных пользователей', fontsize = 14,
fontweight ='bold')
#добавим подписи с количеством
for i in ax.patches:
   if i.get_width() > 3000:
       ax.text(i.get_width() - 900, i.get_y() + 0.5,
               str(int(i.get_width())), fontsize=20, color='white')
   else:
       ax.text(i.get_width() +30, i.get_y() + 0.5,
               str(int(i.get_width())), fontsize=20, color='black')
plt.xlabel('Пользователи, человек', fontsize = 14)
```

```
plt.ylabel('Событие', fontsize = 14)
plt.show()
```



4.2.3 События по пользователям

Прокомментируем получившиеся результаты. Рейтинг событий с количеством совершенных действий и количеством пользователей получился одинаковым.

- MainScreenAppear - Отображение главного экрана -самое популярное событие. Уникальных пользователей, посмотревших на главный экран 7419 чел.
- OffersScreenAppear - Отображение экрана предложений. К экрану предложений обратились 4593 уникальных пользователей.
- CartScreenAppear - Отображение экрана корзины. Дошли до корзину уже 3734 человек.
- PaymentScreenSuccessful - Отображение экрана об успешной оплате. Успешно оплатили корзину 3539 пользователей.
- Tutorial - Инструкция к приложению. Из всех пользователей обратились к инструкции 840 человека. Как видим кто-то из пользователей прочитал инструкцию не один раз, но таких мало, ведь действий совершено всего 1005 на 840 человека.

4.3 Доля пользователей на каждое событие

4.3.1 Какая доля пользователей совершали каждое событие

Посчитаем, какая доля пользователей совершали хотя бы один раз событие.

In [30]:

```
users_per_event = (
    df_new.groupby('event_name')
    .agg(users_uniq=('user_id','nunique'))
    .sort_values('users_uniq', ascending=False)
    .reset_index()
)
users_per_event
```

Out[30]:

	event_name	users_uniq
0	MainScreenAppear	7419
1	OffersScreenAppear	4593
2	CartScreenAppear	3734
3	PaymentScreenSuccessful	3539
4	Tutorial	840

In [31]:

```
print('Всего уникальных пользователей:', df_new['user_id'].nunique())
```

Всего уникальных пользователей: 7534

4.3.2 Диаграмма долей пользователей совершивших событие

Всего 7534 уникальных пользователей. Посмотрим сколько всего пользователей совершило хоть одно действие в процентном соотношении к общему числу пользователей. Визуализируем на диаграмме

In [32]:

```
plt.figure(figsize=(12, 5))
```

```
sns.set (rc={'axes.facecolor':'white', 'figure.facecolor':'white'})
plt.title('Количество пользователей, совершивших событие и их доля от общего числа
пользователей',
          fontsize = 14, fontweight ='bold')
sns.barplot(y=users_per_event['event_name'], x=users_per_event['users_uniq'],
palette='rocket')

#добавим количество на каждый столбик и %
for i, v in enumerate(users_per_event['users_uniq'].values):
   plt.text(v + 100, i - 0.1, str(v), color='black', fontsize = 12)
   plt.text(v + 100, i + 0.2, '({:.1%})'.format(v / df_new['user_id'].nunique()),
color='black', fontsize = 12)

plt.xlabel('Число пользователей')
plt.ylabel('Событие')

plt.show();
```

In [33]:

```
list(df_new['event_name'].unique())
```

Out[33]:

['Tutorial',
 'MainScreenAppear',
 'OffersScreenAppear',
 'CartScreenAppear',
 'PaymentScreenSuccessful']

4.3.3 Доля пользователей хоть раз совершивших событие

Как видно на диаграмме

- Tutorial - экран с инструкцией - самое непопулярное событие у пользователей. Только 11,1% читают порядок пользоания приложением.
- PaymentScreenSuccessful - успешно оплатили корзину - 47% пользователей. Почти каждый второй. Неплохо.
- CartScreenAppear - перешли на страницу с корзиной заказа 50%. Почти все из них потом успешно оплатили корзину.
- OffersScreenAppear - страницу с предложением товаров посмотрели 61% от пользователей.
- MainScreenAppear - главную страницу приложения увидели почти все пользователи - 98,5%

Как ведут себя пользователи в разрезе групп. Посмотрим на диаграмму долей пользователей, совершивших событие по группам тестирования.

4.3.4 Как ведут себя пользователи в разрезе групп тестирования

Посмотрим на диаграмму долей пользователей, совершивших событие по группам тестирования. Нет ли недостатка пользователей из каждой группы тестирования для каждого вида события.

In [34]:

```
users_per_event_test = (
    df_new.groupby(['event_name','group_id'])
    .agg(users_unique=('user_id','nunique'))
    .sort_values('users_unique', ascending=False)
    .reset_index()
)
sns.set (rc={'axes.facecolor':'white', 'figure.facecolor':'white'})
plt.figure(figsize=(12, 5))

colors = ["#F8D030", '#98D8D8', '#6890F0']
# '#78C850', '#F08030', '#6890F0','#F8D030', '#F85888', '#705898', '#98D8D8'
plt.title('Распределение пользователей из разных тестовых групп по событиям',
          fontsize = 14, fontweight ='bold')
sns.barplot(
    y=users_per_event_test['event_name'],
    x=users_per_event_test['users_unique'],
    hue=users_per_event_test['group_id'],
    palette=colors)

plt.xlabel('Количество пользователей')
plt.ylabel('Событие')

plt.show()
```

4.3.5 Распределение групп пользователей по событиям

Пользователи из групп иестирования A1, A2 и B по каждому виду события распределены равномерно. Перекосов нет.

4.4 Воронка событий

Предположим, что наиболее вероятная последовательность событий пользователя в нашем приложении выглядит так:

- сначала просмотр главной страницы - MainScreenAppear
- затем переход и просмотр страницы с предложение товаров - OffersScreenAppear
- затем просмотр страницы с корзиной товаров - CartScreenAppear
- затем осуществляется оплата и пользователь попадает на страницу с успешной оплатой корзины - PaymentScreenSuccessful
- Просмотр инструкции к приложению - Tutorial в последовательную цепочку событий не входит.

Исключим события - Tutorial и посмотрим на воронку событий.

4.4.1 Воронка событий пользователей прошедших через всю воронку

Посмотрим воронку событий с числом пользователей, прошедших всю воронку, без разбивки на группы тестирования.

In [35]:

```
from plotly import graph_objects as go
users = df_new[df_new['event_name'] != 'Tutorial'].pivot_table(
    index='user_id',
    columns='event_name',
    values='datetime',
    aggfunc='min')

step_1 = ~users['MainScreenAppear'].isna()
step_2 = step_1 & (users['OffersScreenAppear'] > users['MainScreenAppear'])
step_3 = step_2 & (users['CartScreenAppear'] > users['OffersScreenAppear'])
step_4 = step_3 & (users['PaymentScreenSuccessful'] > users['CartScreenAppear'])

n_main_screen = users[step_1].shape[0]
n_offers_screen = users[step_2].shape[0]
n_cart_screen = users[step_3].shape[0]
n_payment_success = users[step_4].shape[0]

events_name = ['Посетили главную страницу',
               'Посмотрели экран с товарами',
               'Перешли на страницу корзины',
               'Успешно оплатили']
n_users = [n_main_screen, n_offers_screen, n_cart_screen, n_payment_success]

fig = go.Figure(go.Funnel(
    y = events_name,
    x = n_users
    ))
fig.update_layout(title_text='Количество пользователей прошедших через всю воронку')
fig.show()
print(
f'Потери пользователей после действия MainScreenAppear: {round((1 - n_offers_screen /
n_main_screen) * 100), 2}%.\n\
Потери пользователей после действия OffersScreenAppear: {round((1 - n_cart_screen /
n_offers_screen) * 100), 2}%.\n\
Потери пользователей после действия CartScreenAppear: {round((1 - n_payment_success /
n_cart_screen) * 100), 2}%.'
)
print(f'Всю воронку прошло {(round(n_payment_success / n_main_screen * 100, 2))}%
пользователей')
```

Потери пользователей после действия MainScreenAppear: (43, 2)%.
Потери пользователей после действия OffersScreenAppear: (58, 2)%.
Потери пользователей после действия CartScreenAppear: (74, 2)%.
Всю воронку прошло 6.12% пользователей

4.4.2 Какая доля пользователей доходит от первого события до оплаты.

Сделаем выводы по воронке событий для пользователей, прошедших все шаги.
Видно, что всю воронку прошли всего 6,12% пользователей от тех, которые прошли через все шаги воронки, если за 100% считать тех, кто посетил главную старницу приложения.
Это говорит о том, что для совершения оплаты или добавления товара в корзину, не требуется посещать все шаги, иными словами у посетителей есть вариант купить товар минуя какие-то шаги.

4.4.3 Воронка событий

Мы определились с последовательностью действий, а именно:

- MainScreenAppear событие A
- OffersScreenAppear событие B
- CartScreenAppear событие C
- PaymentScreenSuccessful событие D

По воронке событий посчитаем, какая доля пользователей проходит на следующий шаг воронки (от числа пользователей на предыдущем). То есть для последовательности событий A → B → C→ D посчитаем отношение числа пользователей с событием B к количеству пользователей с событием A, а также отношение числа пользователей с событием C к количеству пользователей с событием B, а также отношение числа пользователей с событием D к количеству пользователей с событием C.
На каком шаге теряете больше всего пользователей? Какая доля пользователей доходит от первого события до оплаты?

In [36]:

```
# посчитаем % по A → B → C→ D  (от предыдущего шага)
users = df_new.pivot_table(
    index='user_id',
    columns='event_name',
    values='datetime',
    aggfunc='min')

print('Посетителей всего:',
      '({:.0%})'.format(users['MainScreenAppear'].count() /
users['MainScreenAppear'].count()))
print('Просмотрели страницу с предложением товара в % от предыдущего шага:',
      '({:.0%})'.format(users['OffersScreenAppear'].count() /
users['MainScreenAppear'].count()))
print('Перешли на страницу корзины товаров в % от предыдущего шага:',
      '({:.0%})'.format(users['CartScreenAppear'].count() /
users['OffersScreenAppear'].count()))
print('Перешли на страницу с успешной оплатой в % от предыдущего шага:',
      '({:.0%})'.format(users['PaymentScreenSuccessful'].count() /
users['CartScreenAppear'].count()))
```

Посетителей всего: (100%)
Просмотрели страницу с предложением товара в % от предыдущего шага: (62%)
Перешли на страницу корзины товаров в % от предыдущего шага: (81%)
Перешли на страницу с успешной оплатой в % от предыдущего шага: (95%)

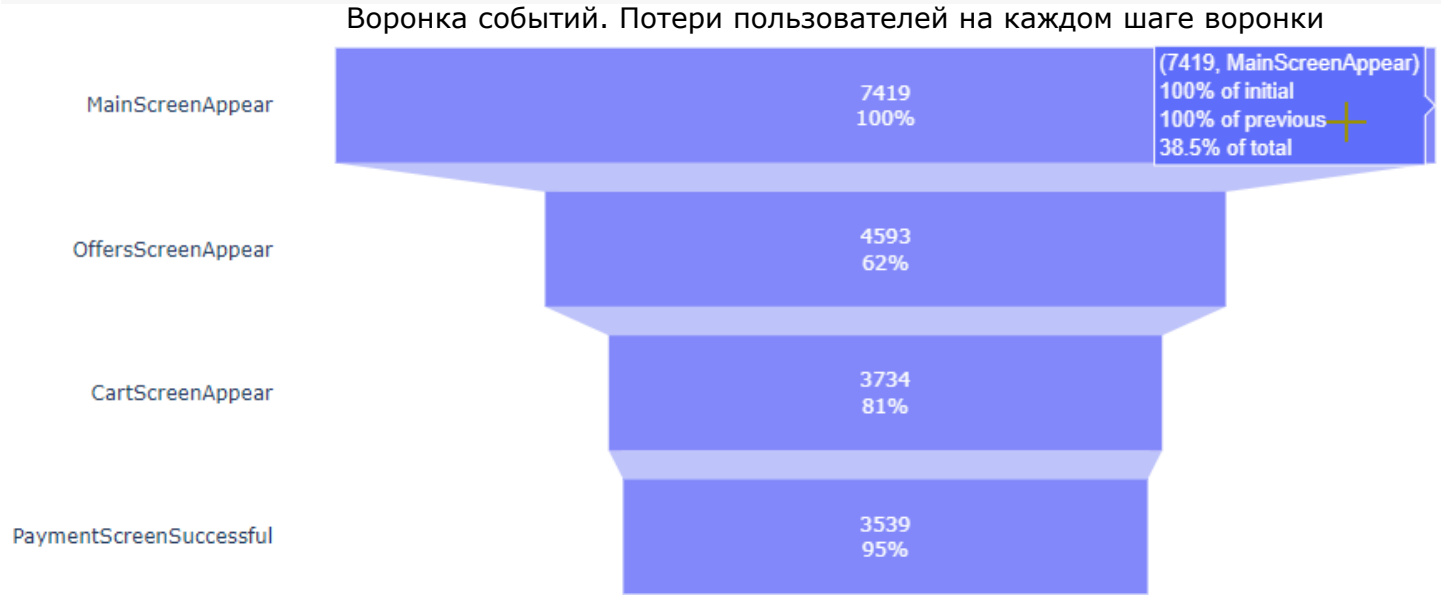
4.4.4 Визуализация потерь пользователей на шагах воронки

Построим воронку событий и посмотрим на каком шаге теряем больше всего пользователей.

In [37]:

```
funnel = (df_new.
          groupby('event_name', as_index=False).
          agg({'user_id': 'nunique'}).
          rename(columns={'user_id' : 'total_users'}).
```

```
        sort_values(by='total_users', ascending=False))
funnel['percent'] = round(funnel['total_users'] / df_new['user_id'].nunique() * 100, 2)
funnel = funnel[funnel['event_name'] != 'Tutorial']
fig = go.Figure(go.Funnel(y = funnel['event_name'],
                          x = funnel['total_users'],
                          opacity = 0.8,
                          textposition = 'inside',
                          textinfo = 'value + percent previous'))
fig.update_layout(title_text='Воронка событий. Потери пользователей на каждом шаге
воронки')
fig.layout.template = 'plotly_white'
fig.show()
```



4.4.5 Выводы по воронке. На каком шаге теряем пользователей

Видим, что самые большие потери на первом шаге, так как до следующего шага доходят только 62% пользователей, иными словам, после просмотра главной страницы теряется 38% пользователей.
Зато если пользователь ознакомился со страницей предложений товаров, то в 81% случаев доходят до корзины.
И из дошедших до корзины уе 95% пользователей успешно осуществляют оплату.

5 Изучим результаты эксперимента.

5.1 Сколько пользователей в каждой экспериментальной группе.

Мы знаем, что у нас есть 2 контрольные группы для A/A-эксперимента: эксперимент № 246, которой мы присвоили код A1 и эксперимент № 247, которой мы присвоили код A2). Контрольные группы нужны чтобы проверить корректность всех механизмов и расчётов. Есть одна экспериментальная группа B (эксперимент № 248).

5.1.1 Посмотрим на количество участников в каждой группе.

In [38]:

```
df_new.groupby('group_id', as_index=False).agg(count=('user_id', 'nunique'))\
.sort_values(by=['count'], ascending=False)
```

Out[38]:

	group_id	count
2	B	2537
1	A2	2513
0	A1	2484

У группы B больше всего пользователей, у группы A1 меньше всего. Посмотрим какое количество пользователей соврешала действия в каждой группе.

5.1.2 сводная таблица с событиями с разбивкой по группам тестирования.

In [39]:

```
funnel_group = (df_new.
        groupby(['event_name', 'group_id'], as_index=False).
        agg({'user_id': 'nunique'}).
```

```
        rename(columns={'user_id' : 'total_users'}).
        sort_values(by=['group_id', 'total_users'], ascending=False))

funnel_group = funnel_group[funnel_group['event_name'] != 'Tutorial']
funnel_group.style.background_gradient(sns.light_palette("seagreen", as_cmap=True))
```

Out[39]:

	event_name	group_id	total_users
5	MainScreenAppear	B	2493
8	OffersScreenAppear	B	1531
2	CartScreenAppear	B	1230
11	PaymentScreenSuccessful	B	1181
4	MainScreenAppear	A2	2476
7	OffersScreenAppear	A2	1520
1	CartScreenAppear	A2	1238
10	PaymentScreenSuccessful	A2	1158
3	MainScreenAppear	A1	2450
6	OffersScreenAppear	A1	1542
0	CartScreenAppear	A1	1266
9	PaymentScreenSuccessful	A1	1200

In [40]:

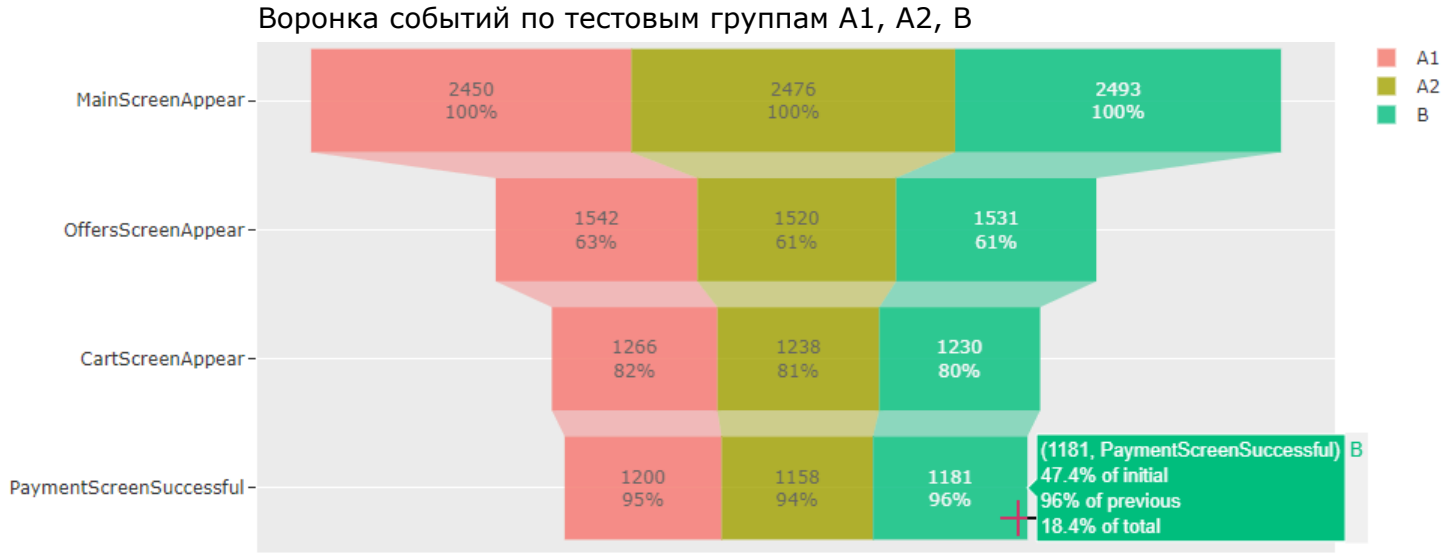
```
fig = go.Figure()

fig.add_trace(go.Funnel(name = 'A1',
                        y = funnel_group.query('group_id == "A1"')['event_name'],
                        x = funnel_group.query('group_id == "A1"')['total_users'],
                        opacity = 0.8,
                        textposition = 'inside',
                        textinfo = 'value + percent previous'))

fig.add_trace(go.Funnel(name = 'A2',
                        y = funnel_group.query('group_id == "A2"')['event_name'],
                        x = funnel_group.query('group_id == "A2"')['total_users'],
                        opacity = 0.8,
                        textposition = 'inside',
                        textinfo = 'value + percent previous'))


fig.add_trace(go.Funnel(name = 'B',
                        y = funnel_group.query('group_id == "B"')['event_name'],
                        x = funnel_group.query('group_id == "B"')['total_users'],
                        opacity = 0.8,
                        textposition = 'inside',
                        textinfo = 'value + percent previous'))

fig.update_layout(title_text='Воронка событий по тестовым группам A1, A2, B')
fig.layout.template = 'ggplot2' #'seaborn', 'simple_white', 'plotly',
       # 'plotly_white', 'plotly_dark', 'presentation', 'xgridoff',
       # 'ygridoff', 'gridon', 'none'
fig.show()
```



5.1.3 Итоги по воронке событий с разбивкой по группам тестирования.

Визуально видно, что группы очень похожи.

Для начала проверим находят ли статистические критерии разницу между выборками экспериментов 246 и 247 (A/A-тест).

Используем Z-критерий (статистический тест, позволяющий определить, различаются ли два средних значения генеральной совокупности, когда дисперсии известны и размер выборки велик).

Для удобства, напишем функцию.

5.2 Функция проверки z_test

In [41]:

```
def z_test(df1, df2, event, alpha, n):
    '''
Функция принимает на вход две таблицы с логами. По заданному событию попарно проверяет
есть ли статистически значимая разница между долями пользователей, совершивших его в перво
группе второй группе.

Входные параметры:
    - df1, df2 - датафреймы с логами
    - event - событие
    - alpfa - выбранный уровень статистической значимости
    - n - поправка Боннферони для критического уровня статистической значимости
    '''

    # критический уровень статистической значимости с поправкой Бонферрони
    bonferroni_alpha = alpha / n
    # число пользователей в группе 1 и группе 2:
    n_users = np.array([df1['user_id'].nunique(),
                        df2['user_id'].nunique()])

    # число пользователей, совершивших событие в группе 1 и группе 2
    success = np.array([df1[df1['event_name'] == event]['user_id'].nunique(),
                        df2[df2['event_name'] == event]['user_id'].nunique()])

    # пропорции успехов в группах:
    p1 = success[0] / n_users[0]
    p2 = success[1] / n_users[1]

    # пропорция успехов в комбинированном датасете:
    p_combined = (success[0] + success[1]) / (n_users[0] + n_users[1])

    # разница пропорций в датасетах
    difference = p1 - p2
```

```
    # считаем статистику в ст.отклонениях стандартного нормального распределения
    z_value = difference /  np.sqrt(p_combined * (1 - p_combined) * (1/n_users[0] +
1/n_users[1]))

    # задаем стандартное нормальное распределение (среднее 0, ст.отклонение 1)
    distr = stats.norm(0, 1)

    p_value = (1 - distr.cdf(abs(z_value))) * 2    #тест двусторонний, удваиваем результат

    print('Событие:', event)
    print('p-значение: ', (round(p_value, 4)))

    if p_value < bonferroni_alpha:
        print('Отвергаем нулевую гипотезу: между долями есть статистически значимая
разница')
    else:
        print(
        'Не получилось отвергнуть нулевую гипотезу, нет оснований считать доли разными')
```

5.3 Описание процесса проверки.

Нам нужно будет сопоставить доли по каждому событию между:

- контрольными группами A1 №эксперимента 246 и A2 №эксперимента 247;
- каждой из контрольной группы по отдельности и экспериментальной (A1 № 246 и B № 248 и A2 № 247 и B № 248);
- объединенной контрольной группой и экспериментальной (A1 № 246 + A2 № 247) и B № 248.

Получилось 4 вида событий, 4 A/A теста и 12 A/B, всего 16.
следовательно для всех тестов мы вводим поправку Бонферрони bonferroni_alpha = alpha / 4 или bonferroni_alpha = alpha / 12, чтобы застраховать себя от ложного результата.

5.4 Гипотезы для проверки

Сформулируем основную и альтернативные гипотезы для всех попарных сравнений:

- **H0: Доли уникальных посетителей, побывавших на этапе воронки, одинаковы**
- **H1: Доли уникальных посетителей, побывавших на этапе воронки, отличаются**

5.4.1 Сформируем датафремй с группами A1, A2 и B

In [42]:

```
df_a_1 = df_new[df_new['group_id'] == 'A1']
df_a_2 = df_new[df_new['group_id'] == 'A2']
df_a = df_new[df_new['group_id'] != 'B']
df_b = df_new[df_new['group_id'] == 'B']
```

5.4.2 Проверим наличие статистически значимой разницы между группами A1 и A2

Применим функцию z_test(df1, df2, event, alpha, n) к датафреймам df_a_1 и df_a_2

In [43]:

```
for event in funnel_group['event_name'].unique():
    z_test(df_a_1, df_a_2
           , event, 0.05, 4)
    print()
```

Событие: MainScreenAppear
p-значение: 0.7571
Не получилось отвергнуть нулевую гипотезу, нет оснований считать доли разными

Событие: OffersScreenAppear
p-значение: 0.2481
Не получилось отвергнуть нулевую гипотезу, нет оснований считать доли разными

Событие: CartScreenAppear
p-значение: 0.2288
Не получилось отвергнуть нулевую гипотезу, нет оснований считать доли разными

Событие: PaymentScreenSuccessful
p-значение: 0.1146
Не получилось отвергнуть нулевую гипотезу, нет оснований считать доли разными

5.4.3 Итоги проверки статистически значимой разницы между группами A1 и A2

Статистические критерии не находят разницу между выборками № 246 и № 247. По результатам проведения тестирования, по каждому из произведенных действий клиентов по группам доли между группами не являются статистически разными.

5.4.4 Проверим наличие статистически значимой разницы между группами A1 и B

Применим функцию z_test(df1, df2, event, alpha, n) к датафреймам df_a_1 и df_b

In [44]:

```
for event in funnel_group['event_name'].unique():
    z_test(df_a_1, df_b
           , event, 0.05, 4)
    print()
```

Событие: MainScreenAppear
p-значение: 0.295
Не получилось отвергнуть нулевую гипотезу, нет оснований считать доли разными

Событие: OffersScreenAppear
p-значение: 0.2084
Не получилось отвергнуть нулевую гипотезу, нет оснований считать доли разными

Событие: CartScreenAppear
p-значение: 0.0784
Не получилось отвергнуть нулевую гипотезу, нет оснований считать доли разными

Событие: PaymentScreenSuccessful
p-значение: 0.2123
Не получилось отвергнуть нулевую гипотезу, нет оснований считать доли разными

5.4.5 Итоги проверки статистически значимой разницы между группами A1 и B

Статистические критерии не находят разницу между выборками № 246 и № 248. По результатам проведения тестирования, по каждому из произведенных действий клиентов по группам доли между группами не являются статистически разными

5.4.6 Проверим наличие статистически значимой разницы между группами A2 и B

Применим функцию z_test(df1, df2, event, alpha, n) к датафреймам df_a_2 и df_b

In [45]:

```
for event in funnel_group['event_name'].unique():
    z_test(df_a_2, df_b
           , event, 0.05, 4)
    print()
```

Событие: MainScreenAppear
p-значение: 0.4587
Не получилось отвергнуть нулевую гипотезу, нет оснований считать доли разными

Событие: OffersScreenAppear
p-значение: 0.9198
Не получилось отвергнуть нулевую гипотезу, нет оснований считать доли разными

Событие: CartScreenAppear
p-значение: 0.5786
Не получилось отвергнуть нулевую гипотезу, нет оснований считать доли разными

Событие: PaymentScreenSuccessful
p-значение: 0.7373
Не получилось отвергнуть нулевую гипотезу, нет оснований считать доли разными

5.4.7 Итоги проверки статистически значимой разницы между группами A2 и B

Статистические критерии не находят разницу между выборками № 247 A2 и № 248 B. По результатам проведения тестирования, по каждому из произведенных действий клиентов по группам доли между группами не являются статистически разными

5.4.8 Проверим наличие статистически значимой разницы между группами A и B

Проверим есть ли статистически значимая разница между объединённой контрольной группой и экспериментальной 248 группами.
Применим функцию z_test(df1, df2, event, alpha, n) к датафреймам df_a и df_b

In [46]:

```
for event in funnel_group['event_name'].unique():
    z_test(df_a, df_b
           , event, 0.05, 4)
    print()
```

Событие: MainScreenAppear
p-значение: 0.2942
Не получилось отвергнуть нулевую гипотезу, нет оснований считать доли разными

Событие: OffersScreenAppear
p-значение: 0.4343
Не получилось отвергнуть нулевую гипотезу, нет оснований считать доли разными

Событие: CartScreenAppear
p-значение: 0.1818
Не получилось отвергнуть нулевую гипотезу, нет оснований считать доли разными

Событие: PaymentScreenSuccessful
p-значение: 0.6004
Не получилось отвергнуть нулевую гипотезу, нет оснований считать доли разными

5.4.9 Итоги проверки статистически значимой разницы между группами A и B

По результатам проведения тестирования, по каждому из произведенных действий клиентов по группам доли между группами не являются статистически разными

6 Вывод

Мы проанализировали поведение покупателей на основании логов пользователей, а так же, результаты A/A/B-теста.
Всего событий в логе после обработки 240887 (243713 до обработки) После обработки потеряно 2826 событий или1.173 %.
Всего уникальных пользователей в логе после обработки 7534 (7551 до обработки). После обработки потеряно 17 пользователей или 0.2 %.
На одного пользователя медианное количество событие равно 19. На одного пользователя среднее количество событие равно 31.
Мы располагаем полными данными за период с 2019-08-01 00:07:28 по 2019-08-07 21:15:17 Максимальная дата 2019-08-01 00:07:28. Минимальная дата2019-08-07 21:15:17.
После предобработки рассмотрели поведение пользователей мобильного приложения и выявили следующее:

- MainScreenAppear - Отображение главного экрана -самое популярное событие. Уникальных пользователей, посмотревших на главный экран 7419 чел.
    - OffersScreenAppear - Отображение экрана предложений. К экрану предложений обратились 4593 уникальных пользователей.
    - CartScreenAppear - Отображение экрана корзины. Дошли до корзину уже 3734 человек.
    - PaymentScreenSuccessful - Отображение экрана об успешной оплате. Успешно оплатили корзину 3539 пользователей.

- Tutorial - Инструкция к приложению. Из всех пользователей обратились к инструкции 840 человека. Как видим кто-то из пользователей прочитал инструкцию не один раз, но таких мало, ведь действий совершено всего 1005 на 840 человека. При этом событие Tutorial в воронке событий было исключено из анализа. Так как оно необязательного для пользователя, не оказывает влияния на остальные шаги.

Выявили, что больше всего потерь пользователей происходит после первого шага (более 38%).
Мы сопоставили доли по каждому событию между:

- контрольными группами A1 №эксперимента 246 и A2 №эксперимента 247;
- каждой из контрольной группы по отдельности и экспериментальной (A1 № 246 и B № 248 и A2 № 247 и B № 248);
- объединенной контрольной группой и экспериментальной (A1 № 246 + A2 № 247) и B № 248.

Получилось 4 вида событий, 4 A/A теста и 12 A/B, всего 16.
Множество A/B-тестов, проведённых по каждому из событий, не обнаружили статистически значимой разницы между группами.
Можно сделать вывод, что изменение шрифтов во всём приложении на поведение пользователей не повлияло. Дизайнеры могут менять шрифты без риска, что пользователям будет непривычно пользоваться приложением и это повлияет на их поведение.

7 Риск ошибок в тесте.

При проверке статистических гипотез был выбран уровень значимости 0.05. Это означает, что допустимая вероятность ложноположительного результата для одного теста равна 5%.
Всего было сделано 16 проверок статистических гипотез.
Найдем допустимую вероятность ложноположительного результата хотя бы в одном из тестов.

7.1 Риск ошибок при уровне значимости 0.05

In [47]:

```
print(f'Вероятность того, что не будет ни одного ложноположительного\
результата при 16 тестах, равна: {round(((1 - 0.05)**16) * 100, 1)}%')
print(f'Вероятность того, что хотя бы один тест даст ложноположительный\
результат, равна: {round((1 - (1 - 0.05)**16) * 100, 1)}%')
```

Вероятность того, что не будет ни одного ложноположительного результата при 16 тестах, равна: 44.0%
Вероятность того, что хотя бы один тест даст ложноположительный результат, равна: 56.0%

7.2 Риск ошибок при уровне значимости 0.025

Что если мы снизим уровень значимости до 0.025 и затем до 0.01

In [48]:

```
print(f'Вероятность того, что не будет ни одного ложноположительного\
результата при 16 тестах, равна: {round(((1 - 0.025)**16) * 100, 1)}%')
print(f'Вероятность того, что хотя бы один тест даст ложноположительный\
результат, равна: {round((1 - (1 - 0.025)**16) * 100, 1)}%')
```

Вероятность того, что не будет ни одного ложноположительного результата при 16 тестах, равна: 66.7%
Вероятность того, что хотя бы один тест даст ложноположительный результат, равна: 33.3%

7.3 Риск ошибок при уровне значимости 0.01

In [49]:

```
print(f'Вероятность того, что не будет ни одного ложноположительного\
результата при 16 тестах, равна: {round(((1 - 0.01)**16) * 100, 1)}%')
print(f'Вероятность того, что хотя бы один тест даст ложноположительный\
результат, равна: {round((1 - (1 - 0.01)**16) * 100, 1)}%')
```

Вероятность того, что не будет ни одного ложноположительного результата при 16 тестах, равна: 85.1%
Вероятность того, что хотя бы один тест даст ложноположительный результат, равна: 14.9%

7.4 Резюмируем по уровню значимости.

Оставим уровень значимости в пределах 0.05. Понижая уровень ошибок первого рода, мы увеличиваем риск ошибок второго рода, а именно ложнонегативный результат.