Изучение поведения пользователей мобильного приложения заказа еды

Описание проекта

Вы работаете в стартапе, который продаёт продукты питания. Нужно разобраться, как ведут себя пользователи вашего мобильного приложения.

Изучите воронку продаж. Узнайте, как пользователи доходят до покупки. Сколько пользователей доходит до покупки, а сколько — «застревает» на предыдущих шагах? На каких именно?

После этого исследуйте результаты A/A/B-эксперимента. Дизайнеры захотели поменять шрифты во всём приложении, а менеджеры испугались, что пользователям будет непривычно. Договорились принять решение по результатам A/A/B-теста. Пользователей разбили на 3 группы: 2 контрольные со старыми шрифтами и одну экспериментальную — с новыми. Выясните, какой шрифт лучше.

Описание данных

Данные представлены в файле _/datasets/logs*exp.csv*

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.

Оглавление

Шаг 1. Загрузка данных и изучение общей информации

In [1]:
```
# For better figure's quality
%config InlineBackend.figure_format = 'retina'
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from plotly import graph_objects as go
from scipy import stats as st
import math
import numpy as np
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
```

In [2]:
```
# For better printing
pd.set_option('display.max_columns', 8)
```

In [3]:
```
# workaround to use praktikum file system as well as local windows system
try:
    df = pd.read_csv('/datasets/logs_exp.csv', sep='\t')
except:
    df = pd.read_csv('datasets/logs_exp.csv', sep='\t')
```

In [4]:
```
df.head()
```

Out[4]:

	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

In [5]:
```
df.info()
```

```
<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
```

Данные прочитаны, в столбцах ожидаемая информация, timestamp времени стоит сконвертировать в дату-время.

Шаг 2. Подготовка данных

Заменим названия столбцов на общепринятые для датафреймов.

In [6]:
```
t1 = df.columns[0]
```

In [7]:
```
df.columns = ['event_name', 'device_id', 'event_timestamp', 'exp_id']
```

In [8]:
```
df.head()
```

Out[8]:

	event_name	device_id	event_timestamp	exp_id
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

Теперь с названиями столбцов будет легче работать.

Проверим наличие пропусков, дубликатов и типы данных.

In [9]:
```
df.isna().sum()
```

Out[9]:
```
event_name         0
device_id          0
event_timestamp    0
exp_id             0
dtype: int64
```

Пропусков данных нет.

In [10]:
```
df.duplicated().sum()
```

Out[10]:
```
413
```

В исходных данных есть 413 полностью одинаковых строчек. Проверим данные.

In [11]:
```
df[df.duplicated()]
```

Out[11]:

	event_name	device_id	event_timestamp	exp_id
453	MainScreenAppear	5613408041324010552	1564474784	248
2350	CartScreenAppear	1694940645335807244	1564609899	248

	event_name	device_id	event_timestamp	exp_id
3573	MainScreenAppear	434103746454591587	1564628377	248
4076	MainScreenAppear	3761373764179762633	1564631266	247
4803	MainScreenAppear	2835328739789306622	1564634641	248
...	...	...	...	...
242329	MainScreenAppear	8870358373313968633	1565206004	247
242332	PaymentScreenSuccessful	4718002964983105693	1565206005	247
242360	PaymentScreenSuccessful	2382591782303281935	1565206049	246
242362	CartScreenAppear	2382591782303281935	1565206049	246
242635	MainScreenAppear	4097782667445790512	1565206618	246

413 rows × 4 columns

In [12]:
```
df.query('device_id == 5613408041324010552')
```

Out[12]:

	event_name	device_id	event_timestamp	exp_id
452	MainScreenAppear	5613408041324010552	1564474784	248
453	MainScreenAppear	5613408041324010552	1564474784	248
104383	MainScreenAppear	5613408041324010552	1564857690	248
104628	MainScreenAppear	5613408041324010552	1564858279	248
104637	MainScreenAppear	5613408041324010552	1564858297	248
145276	MainScreenAppear	5613408041324010552	1564986831	248
145550	MainScreenAppear	5613408041324010552	1564987332	248
205860	MainScreenAppear	5613408041324010552	1565112335	248
205869	MainScreenAppear	5613408041324010552	1565112351	248
205915	MainScreenAppear	5613408041324010552	1565112400	248

Как видно, с технической стороны для одного и того же устройства и одного типа событий генерится 2 события в одну и ту же секунду Unix time. Возможно, программа отрабатывает неверно, но так как дубликатов всего

In [13]:
```
(df.duplicated().sum() / len(df)).round(3)
```

Out[13]:
```
0.002
```

всего 0.2%, то пока что оставим дубликаты как есть. Возможно стоит сообщить коллегам, что в некоторых случаях событие дублируется.

Добавим столбца даты и времени, а также отдельный столбец дат.

In [14]:
```
df['event_datetime'] = pd.to_datetime(df['event_timestamp'], unit='s')
```

In [15]:
```
df['event_date'] = df['event_datetime'].astype('datetime64[D]')
```

In [16]:
```
df.head()
```

Out[16]:

	event_name	device_id	event_timestamp	exp_id	event_datetime	event_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

Мы получили готовый к обработке массив данных.

Шаг 3. Изучение и проверка данных

Изучим количественные показатели в исходных данных.

In [17]:
```
len(df)
```

Out[17]:
```
244126
```

Всего 244 тыс. событий в исходных данных.

In [18]:
```
df.device_id.nunique()
```

Out[18]:
```
7551
```

Для 7551 пользователя.

In [19]:
```
len(df) / df.device_id.nunique()
```

Out[19]:
```
32.33028737915508
```

На каждого пользователя приходится почти по 32 события.

Рассмотрим данные по времени.

In [20]:
```
df.event_date.min()
```

Out[20]:
```
Timestamp('2019-07-25 00:00:00')
```

In [21]:
```
df.event_date.max()
```

Out[21]:
```
Timestamp('2019-08-07 00:00:00')
```

В исходных данных события за период с 25 июля по 7 августа 2019 года.

In [22]:
```
sns.set_style("whitegrid");
plt.figure(figsize=(15,4));
df['event_date'].hist(bins=50);
plt.xlabel('Дата');
plt.ylabel('Количество событий');
plt.title('Распределение событий по времени');
```

Как видно на распределении, большая часть событий касаются периода с 1 августа до 7 августа 2019 года. Проверим сколько событий лежат вне этого диапазона.

In [23]:
```
len(df.query('event_date < "2019-08-01"'))
```

Out[23]:
```
2828
```

In [24]:
```
len(df.query('event_date < "2019-08-01"')) / len(df)
```

Out[24]:
```
0.011584181938834865
```

С учетом того, что только 1% всех событий в исходных данных касаются периода до августа 2019 года, то мы можем исключить эти события и упростить анализ, не потеряв 99% данных.

In [25]:
```
logs = df.query('not event_date < "2019-08-01"')
```

In [26]:
```
logs.head()
```

Out[26]:

	event_name	device_id	event_timestamp	exp_id	event_datetime	event_date
2828	Tutorial	3737462046622621720	1564618048	246	2019-08-01 00:07:28	2019-08-01
2829	MainScreenAppear	3737462046622621720	1564618080	246	2019-08-01 00:08:00	2019-08-01
2830	MainScreenAppear	3737462046622621720	1564618135	246	2019-08-01 00:08:55	2019-08-01
2831	OffersScreenAppear	3737462046622621720	1564618138	246	2019-08-01 00:08:58	2019-08-01
2832	MainScreenAppear	1433840883824088890	1564618139	247	2019-08-01 00:08:59	2019-08-01

In [27]:
```
logs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 241298 entries, 2828 to 244125
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype         
---  ------           --------------   -----         
 0   event_name       241298 non-null  object        
 1   device_id        241298 non-null  int64         
 2   event_timestamp  241298 non-null  int64         
 3   exp_id           241298 non-null  int64         
 4   event_datetime   241298 non-null  datetime64[ns]
 5   event_date       241298 non-null  datetime64[ns]
dtypes: datetime64[ns](2), int64(3), object(1)
memory usage: 12.9+ MB
```

В оставшемся массиве данных 241298 событий, чего вполне достаточно для дальнейшего анализа. События распеределены в течение 7 дней с четверга 2019-08-01 по среду 2019-08-07.

In [28]:
```
df.groupby('exp_id').device_id.nunique().reset_index()
```

Out[28]:

	exp_id	device_id
0	246	2489
1	247	2520
2	248	2542

In [29]:
```
df.device_id.nunique()
```

Out[29]:
```
7551
```

In [30]:
```
logs.groupby('exp_id').device_id.nunique().reset_index()
```

Out[30]:

	exp_id	device_id
0	246	2484
1	247	2513
2	248	2537

In [31]:
```
logs.device_id.nunique()
```

Out[31]:
```
7534
```

In [32]:
```
logs.device_id.nunique() / df.device_id.nunique()
```

Out[32]:
```
0.9977486425638988
```

При удалении части событий мы не потеряли большую часть пользователей (99.8%), распределение между экспериментальными группами не изменилось.

Шаг 4. Изучение воронки событий

Изучим какие события есть в логах.

In [33]:
```
logs.event_name.value_counts().reset_index()
```

Out[33]:

	index	event_name

	index	event_name
0	MainScreenAppear	117431
1	OffersScreenAppear	46350
2	CartScreenAppear	42365
3	PaymentScreenSuccessful	34113
4	Tutorial	1039

Чаще всего встречаются события по открытию основаного экрана (`MainScreenAppear`), далее идут события с предложениями (`OffersScreenAppear`), затем экрана корзины (`CartScreenAppear`), платежа (`PaymentScreenSuccessful`) и реже всего - прохождение обучения (`Tutorial`).

Посмотрим на количество уникальных пользователей, который совершали каждое и этих событий.

In [34]:
```
users_events = logs.groupby('event_name')['device_id'].nunique().sort_values(ascending=False).reset_index()
users_events
```

Out[34]:

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

In [35]:
```
users_events['frac'] = users_events['device_id'] / logs.device_id.nunique()
```

In [36]:
```
users_events
```

Out[36]:

	event_name	device_id	frac
0	MainScreenAppear	7419	0.984736
1	OffersScreenAppear	4593	0.609636
2	CartScreenAppear	3734	0.495620
3	PaymentScreenSuccessful	3539	0.469737
4	Tutorial	840	0.111495

In [37]:
```
users_events.columns = ['event_name', 'users_count', 'frac']
```

In [38]:
```
users_events
```

Out[38]:

	event_name	users_count	frac
0	MainScreenAppear	7419	0.984736
1	OffersScreenAppear	4593	0.609636
2	CartScreenAppear	3734	0.495620
3	PaymentScreenSuccessful	3539	0.469737
4	Tutorial	840	0.111495

Оказалось, что только 98% пользователей открывали главный экран приложения. То есть есть пользователи, для которых не был зафиксирован первичный вход. Возможно, он состоялся в дату ранее 1 августа 2019 года, поэтому он не попал в выборку.

По этой таблице видно, что сначала пользователи попадают на главный экран, затем опционально просматривают экран с предложениями, затем переходят на экран с корзиной и в конце концов покупают товар. Из общего ряда выпадает **экран с обучением (`Tutorial`)**, который не требуется для совершения покупки и поэтому **может быть исключен из расчета воронки событий**.

In [39]:
```
funnel = users_events[:4]
```

In [40]:
```
funnel
```

Out[40]:

	event_name	users_count	frac
0	MainScreenAppear	7419	0.984736
1	OffersScreenAppear	4593	0.609636
2	CartScreenAppear	3734	0.495620
3	PaymentScreenSuccessful	3539	0.469737

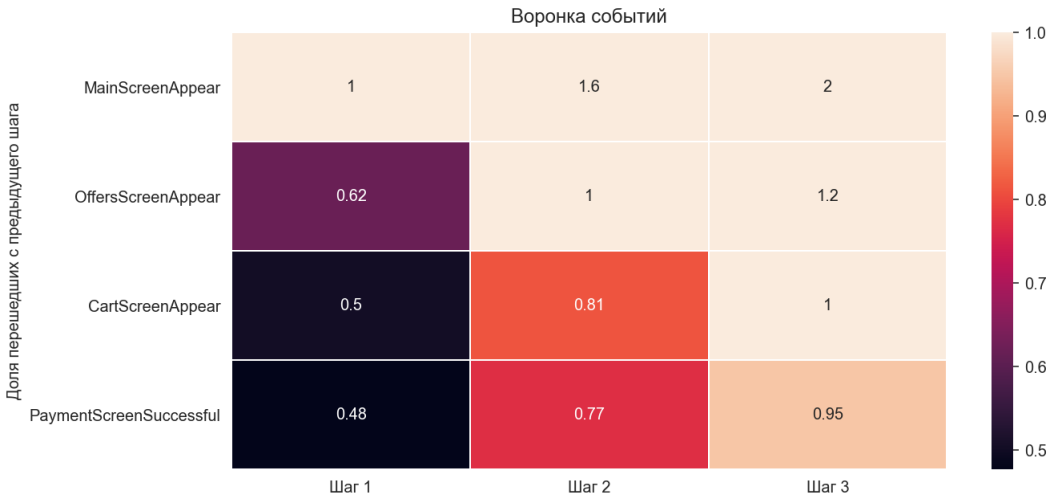
In [41]:
```
for i in range(len(funnel)-1):
    funnel.loc[:, ('frac_step'+str(i+1))] = funnel.loc[:, 'users_count'] / funnel.loc[i, 'users_count']
```

In [42]:
```
funnel
```

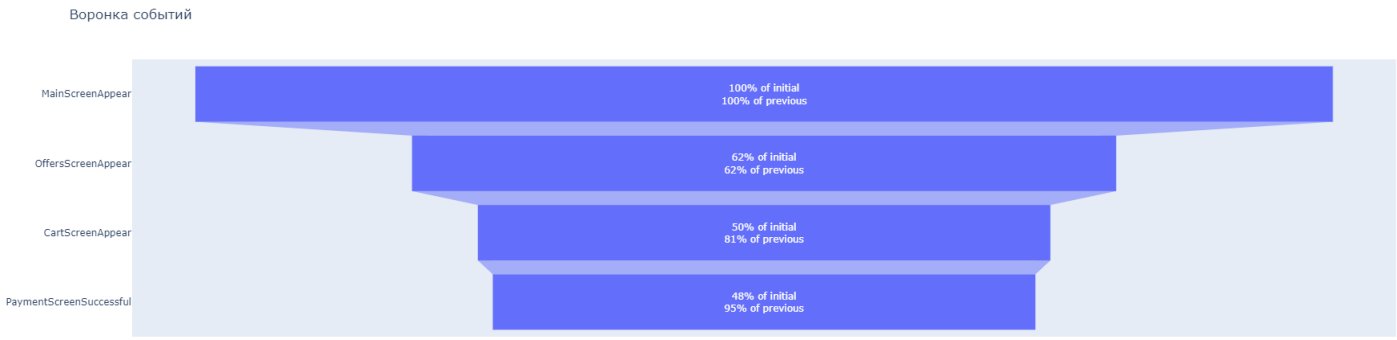
Out[42]:

	event_name	users_count	frac	frac_step1	frac_step2	frac_step3
0	MainScreenAppear	7419	0.984736	1.000000	1.615284	1.986877
1	OffersScreenAppear	4593	0.609636	0.619086	1.000000	1.230048
2	CartScreenAppear	3734	0.495620	0.503302	0.812976	1.000000
3	PaymentScreenSuccessful	3539	0.469737	0.477018	0.770520	0.947777

In [43]:
```
plt.figure(figsize=(10,5));
sns.heatmap(data=funnel[['frac_step1', 'frac_step2', 'frac_step3']],
            vmax=1.0,
            linewidths=.5,
            annot=True,
            yticklabels=funnel['event_name'],
            xticklabels=['Шаг 1', 'Шаг 2', 'Шаг 3']);
plt.ylabel('Доля перешедших с предыдущего шага');
plt.title('Воронка событий');
```



In [44]:
```
fig = go.Figure(go.Funnel(
    y = funnel['event_name'],
    x = funnel['users_count'],
    textposition = "inside",
    textinfo = "percent initial+percent previous"))
fig.update_layout(
    title_text='Воронка событий'
)
fig.show()
```



Как видно по воронке событий **больше всего пользователей теряются на шаге "Просмотр предложения (`OffersScreenAppear`)" (38%)**. После этого шага **81% всех пользователей переходят в корзину и 95% из них совершают покупку**. Из всех пользователей 48% совершают покупку.

Шаг 5. Изучение результатов эксперимента

Рассчитаем количество пользователей в каждой экспериментальной группе.

In [45]:
```
logs.groupby('exp_id')['device_id'].nunique().reset_index()
```

Out[45]:

	exp_id	device_id
0	246	2484
1	247	2513
2	248	2537

Как видно, во всех группах примерно одинаковое количество пользователей, можно продолжать анализ.

Проверим на контрольной группе правильность расчетов. В исходных данных эти группы носят отметки 246 и 247.

In [46]:
```
aa_pivot = logs.query('exp_id!=248') \
    .pivot_table(index='event_name', columns='exp_id', values='device_id', aggfunc='nunique') \
    .reset_index().sort_values(by=246, ascending=False).reset_index(drop=True)
```

In [47]:
```
aa_pivot
```

Out[47]:

exp_id	event_name	246	247
0	MainScreenAppear	2450	2476
1	OffersScreenAppear	1542	1520
2	CartScreenAppear	1266	1238
3	PaymentScreenSuccessful	1200	1158
4	Tutorial	278	283

Получили сводную таблицу для экспериментов 246 и 247, которая позволит нам проверить, есть ли статистически значимые различия между выборками для A/A теста.

In [48]:
```
aa_pivot[['246_frac','247_frac']] = aa_pivot[[246,247]] / aa_pivot[[246,247]].iloc[0]
```

In [49]:
```
aa_pivot
```

Out[49]:

exp_id	event_name	246	247	246_frac	247_frac
0	MainScreenAppear	2450	2476	1.000000	1.000000
1	OffersScreenAppear	1542	1520	0.629388	0.613893
2	CartScreenAppear	1266	1238	0.516735	0.500000
3	PaymentScreenSuccessful	1200	1158	0.489796	0.467690
4	Tutorial	278	283	0.113469	0.114297

В первом приближении можно заключить, что группа 247 показала на 2 процентных пункта худший результат по сравнению с группой 246 (конверсия в покупку 47% вместо 49%). Проверим, есть ли статистически значимая разница.

Так как согласно центральной предельной теореме выборки выборочные средние нормально распределены вокруг среднего всей совокупности независимо от того, как распределена сама генеральная совокупность. Разница между пропорциями, наблюдаемыми на выборках, будет нашей статистикой (переменная, значения которой рассчитываются только по выборочным данным). Тогда если настоящие пропорции обеих совокупностей не отличаются, то можно расчитать значение z, которое будет распределено нормально и легко можно будет рассчитать насколько полученные пропорции статистически различны. Так как нам надо подтвердить, что они равны, либо не равны, нужно использовать двусторонний тест.

Формула для расчета значения Z:

$$Z \approx \frac{P_1 - P_2) - (\pi_1 - \pi_2)}{\sqrt{P(1-P)(1/n_1 + 1/n_2)}}$$

здесь n_1 и n_2 — размеры двух сравниваемых выборок, то есть количества наблюдений в них; P_1, P_2 — пропорции, наблюдаемые в выборках; P — пропорция в выборке, скомбинированной из двух наблюдаемых; π_1 и π_2 — настоящие пропорции в сравниваемых генеральных совокупностях. Мы будем проверять гипотезу о равенстве π_1 и π_2, поэтому при верной нулевой гипотезе критерий Z можно рассчитывать только по выборочным данным. Это значение будет распределено нормально со средним в 0 и стандартным отклонением 1, поэтому p-значение можно расчитать по формуле нормального распределения.

Выберем для АА-теста критический уровень значимости равный `0.05` . Проверим гипотезы о каждом из событий. Нулевые гипотезы будут о том, что нет статистической разницы между долями пользователей, совершивших действие в выборках 246 и 247, а альтернативная - о том что разница есть. Соответственно, нулевые гипотезы могут быть отвергнуты при p-value ниже выбранного уровня.

In [50]:
```
alpha_aa = 0.05
```

Создадим вспомогательную функцию для проведения Z теста и расчета серий для добавления в датафрейм.

In [51]:
```
def z_test(sources, results):
    # Calculates p-value for two-sided z-test,
    # sources - 2-dimensional array of all visitors for example
    # results - 2-dimensional array of buyers for example

    # fractions
    p1 = results[0]/sources[0]
    p2 = results[1]/sources[1]

    p_combined = (results[0]+results[1])/(sources[0]+sources[1])

    z_value = (p1 - p2) / math.sqrt(p_combined * (1 - p_combined) * (1/sources[0] + 1/sources[1]))

    distr = st.norm(0, 1)

    p_value = (1 - distr.cdf(abs(z_value))) * 2

    return p_value
```

Так как расчет мы проводить исходя из количества всех уникальных пользователей в каждой из групп, создадим вспомогательный массив.

In [52]:
```
users_num = logs.pivot_table(columns='exp_id', values='device_id', aggfunc='nunique').iloc[0]
```

In [53]:
```
users_num
```

Out[53]:
```
exp_id
246    2484
247    2513
248    2537
Name: device_id, dtype: int64
```

In [54]:
```
def get_p_value_column(series1, series2, users1, users2):
    res = pd.Series([np.nan]*len(series1))
    sources = [users1, users2]
    for i in range(len(series1)):
        res[i] = z_test(sources, [series1.iloc[i].values[0], series2.iloc[i].values[0]])
    return res
```

In [55]:
```
def get_z_test_result_column(p_value_column, alpha):
    return p_value_column < alpha
```

Добавим новую колонку для значений p-value.

In [56]:
```
aa_pivot['p_value'] = get_p_value_column(aa_pivot[[246]], aa_pivot[[247]], users_num[246], users_num[247])
```

Добавим столбец с результатами z-теста.

In [57]:
```
aa_pivot['z-test_result'] = get_z_test_result_column(aa_pivot['p_value'], alpha_aa)
```

In [58]:
```
aa_pivot
```

Out[58]:

exp_id	event_name	246	247	246_frac	247_frac	p_value	z-test_result
0	MainScreenAppear	2450	2476	1.000000	1.000000	0.757060	False
1	OffersScreenAppear	1542	1520	0.629388	0.613893	0.248095	False
2	CartScreenAppear	1266	1238	0.516735	0.500000	0.228834	False
3	PaymentScreenSuccessful	1200	1158	0.489796	0.467690	0.114567	False
4	Tutorial	278	283	0.113469	0.114297	0.937700	False

Таким образом по всем событиям нулевую гипотезу нельзя отбросить. Статистически значимых различий между контрольными группами нет.

Теперь можно провести A/B тест сравнение с экспериментальной группой 248.

In [59]:
```
ab_pivot = logs \
    .pivot_table(index='event_name', columns='exp_id', values='device_id', aggfunc='nunique') \
    .reset_index().sort_values(by=246, ascending=False).reset_index(drop=True)
```

In [60]:
```
ab_pivot['246+247'] = ab_pivot[246] + ab_pivot[247]
```

In [61]:
```
ab_pivot
```

Out[61]:

exp_id	event_name	246	247	248	246+247
0	MainScreenAppear	2450	2476	2493	4926
1	OffersScreenAppear	1542	1520	1531	3062
2	CartScreenAppear	1266	1238	1230	2504
3	PaymentScreenSuccessful	1200	1158	1181	2358
4	Tutorial	278	283	279	561

Сравним результаты с каждой из контрольных групп и с объединенной контрольной. Так как мы будем сравнивать одну и ту же выборку 3 раза, то применим поправку Бонферрони к выбору критического уровня значимости.

In [62]:
```
alpha_ab = 0.05 / 3
```

In [63]:
```
ab_pivot['p_value_246_248'] = get_p_value_column(ab_pivot[[246]], ab_pivot[[248]], users_num[246], users_num[248])
```

In [64]:
```
ab_pivot['z-test_result_246_248'] = get_z_test_result_column(ab_pivot['p_value_246_248'], alpha_ab)
```

In [65]:
```
ab_pivot
```

Out[65]:

exp_id	event_name	246	247	248	246+247	p_value_246_248	z-test_result_246_248
0	MainScreenAppear	2450	2476	2493	4926	0.294972	False
1	OffersScreenAppear	1542	1520	1531	3062	0.208362	False
2	CartScreenAppear	1266	1238	1230	2504	0.078429	False
3	PaymentScreenSuccessful	1200	1158	1181	2358	0.212255	False
4	Tutorial	278	283	279	561	0.826429	False

Статистически значимых различий между контрольной группой 246 и экспериментальной 248 нет.

In [66]:
```
ab_pivot['p_value_247_248'] = get_p_value_column(ab_pivot[[247]], ab_pivot[[248]], users_num[247], users_num[248])
```

In [67]:
```
ab_pivot['z-test_result_247_248'] = get_z_test_result_column(ab_pivot['p_value_247_248'], alpha_ab)
```

In [68]:
```
ab_pivot
```

Out[68]:

exp_id	event_name	246	247	248	...	p_value_246_248	z-test_result_246_248	p_value_247_248	z-test_result_247_248
0	MainScreenAppear	2450	2476	2493	...	0.294972	False	0.458705	False
1	OffersScreenAppear	1542	1520	1531	...	0.208362	False	0.919782	False
2	CartScreenAppear	1266	1238	1230	...	0.078429	False	0.578620	False
3	PaymentScreenSuccessful	1200	1158	1181	...	0.212255	False	0.737342	False
4	Tutorial	278	283	279	...	0.826429	False	0.765324	False

5 rows × 9 columns

Статистически значимых различий между контрольной группой 247 и экспериментальной 248 нет.

In [69]:
```
ab_pivot['p_value_246_247_248'] = get_p_value_column(ab_pivot[['246+247']],
                                                     ab_pivot[[248]],
                                                     users_num[246]+users_num[247],
                                                     users_num[248])
```

In [70]:
```
ab_pivot['z-test_result_246_247_248'] = get_z_test_result_column(ab_pivot['p_value_246_247_248'], alpha_ab)
```

In [71]:
```
ab_pivot
```

Out[71]:

exp_id	event_name	246	247	248	...	p_value_247_248	z-test_result_247_248	p_value_246_247_248	z test_result_246_247_24
0	MainScreenAppear	2450	2476	2493	...	0.458705	False	0.294245	Fals
1	OffersScreenAppear	1542	1520	1531	...	0.919782	False	0.434255	Fals
2	CartScreenAppear	1266	1238	1230	...	0.578620	False	0.181759	Fals
3	PaymentScreenSuccessful	1200	1158	1181	...	0.737342	False	0.600429	Fals
4	Tutorial	278	283	279	...	0.765324	False	0.764862	Fals

5 rows × 11 columns

Статистически значимых различий между объединенной контрольной группой 246+247 и экспериментальной 248 нет.

Расчитаем отдельно конверсии в каждом случае (отношение количество покупателей к общему количеству зашедших на главный экран приложения).

In [72]:
```
ab_pivot[248][3] / users_num[248]
```

Out[72]:
```
0.46551044540796216
```

In [73]:
```
ab_pivot[246][3] / users_num[246]
```

Out[73]:
```
0.4830917874396135
```

In [74]:
```
ab_pivot[247][3] / users_num[247]
```

Out[74]:
```
0.46080382013529647
```

In [75]:
```
ab_pivot['246+247'][3] / (users_num[246]+users_num[247])
```

Out[75]:
```
0.47188312987792674
```

Таким образом, несмотря на то, что есть разница в конверсии в покупателей пользователей контрольных групп 246, 247 и экспериментальной 248, **статистически значимой разницы в результатах нет, причем для всех событий и всех гипотез**.

Так как для уменьшения влияния накопления ошибки при множественном сравнении мы использовали поправку

Бонферрони, можно отдельно рассмотреть ситуацию, как изменится результат при выборе критического уровня значимости `0.1` .

In [76]:
```
alpha_new = 0.1
```

In [77]:
```
aa_pivot['z-test_result'] = get_z_test_result_column(aa_pivot['p_value'], alpha_new)
aa_pivot
```

Out[77]:

exp_id	event_name	246	247	246_frac	247_frac	p_value	z-test_result
0	MainScreenAppear	2450	2476	1.000000	1.000000	0.757060	False
1	OffersScreenAppear	1542	1520	0.629388	0.613893	0.248095	False
2	CartScreenAppear	1266	1238	0.516735	0.500000	0.228834	False
3	PaymentScreenSuccessful	1200	1158	0.489796	0.467690	0.114567	False
4	Tutorial	278	283	0.113469	0.114297	0.937700	False

Результаты для A/A теста не изменились, проверим для A/B теста.

In [78]:
```
ab_pivot
```

Out[78]:

exp_id	event_name	246	247	248	...	p_value_247_248	z-test_result_247_248	p_value_246_247_248	z test_result_246_247_24
0	MainScreenAppear	2450	2476	2493	...	0.458705	False	0.294245	Fals
1	OffersScreenAppear	1542	1520	1531	...	0.919782	False	0.434255	Fals
2	CartScreenAppear	1266	1238	1230	...	0.578620	False	0.181759	Fals
3	PaymentScreenSuccessful	1200	1158	1181	...	0.737342	False	0.600429	Fals
4	Tutorial	278	283	279	...	0.765324	False	0.764862	Fals

5 rows × 11 columns

In [79]:
```
ab_pivot['z-test_result_246_248'] = get_z_test_result_column(ab_pivot['p_value_246_248'], alpha_new)
```

In [80]:
```
ab_pivot['z-test_result_247_248'] = get_z_test_result_column(ab_pivot['p_value_247_248'], alpha_new)
```

In [81]:
```
ab_pivot['z-test_result_246_247_248'] = get_z_test_result_column(ab_pivot['p_value_246_247_248'], alpha_new)
```

In [82]:
```
ab_pivot
```

Out[82]:

exp_id	event_name	246	247	248	...	p_value_247_248	z-test_result_247_248	p_value_246_247_248	z test_result_246_247_24
0	MainScreenAppear	2450	2476	2493	...	0.458705	False	0.294245	Fals
1	OffersScreenAppear	1542	1520	1531	...	0.919782	False	0.434255	Fals
2	CartScreenAppear	1266	1238	1230	...	0.578620	False	0.181759	Fals
3	PaymentScreenSuccessful	1200	1158	1181	...	0.737342	False	0.600429	Fals
4	Tutorial	278	283	279	...	0.765324	False	0.764862	Fals

5 rows × 11 columns

Без поправок Бонферрони наблюдается отказ от нулевой гипотезы для случая сравнения групп 246 и 248 по событию `CartScreenAppear` .

При внесении поправки Бонферрони:

In [83]:
```
ab_pivot['z-test_result_246_248'] = get_z_test_result_column(ab_pivot['p_value_246_248'], alpha_new/3)
```

In [84]:
```
ab_pivot['z-test_result_247_248'] = get_z_test_result_column(ab_pivot['p_value_247_248'], alpha_new/3)
```

In [85]:
```
ab_pivot['z-test_result_246_247_248'] = get_z_test_result_column(ab_pivot['p_value_246_247_248'], alpha_new/3)
```

In [86]:
```
ab_pivot
```

Out[86]:

exp_id	event_name	246	247	248	...	p_value_247_248	z-test_result_247_248	p_value_246_247_248	z test_result_246_247_24
0	MainScreenAppear	2450	2476	2493	...	0.458705	False	0.294245	Fals
1	OffersScreenAppear	1542	1520	1531	...	0.919782	False	0.434255	Fals
2	CartScreenAppear	1266	1238	1230	...	0.578620	False	0.181759	Fals
3	PaymentScreenSuccessful	1200	1158	1181	...	0.737342	False	0.600429	Fals
4	Tutorial	278	283	279	...	0.765324	False	0.764862	Fals

5 rows × 11 columns

Таким образом, поправка Бонферрони позволяет избежать накопления ошибки при множественном сравнении и получить одинаковый результат в большем диапазоне критических уровней значимости (в данном случае для 0.05 и 0.10).

Общий вывод и рекомендации

В работе был проведен анализ различных показателей пользователей мобильного приложения заказчика и проведены следующие шаги:

- Загружены и подготовлены данных к анализу
- Изучена воронка событий
- Изучены результаты эксперимента по изменению размера шрифта

Как показал анализ:

- По исходным данным для 244 тыс. событий 7.5 тыс. пользователей за август 2019 года порядка 48% пользователей, открывших приложение, совершают успешную покупку.
- Больше всего пользователей (38%) теряются на этапе просмотра экрана с предложениями и не доходят далее до выбора товара и его покупки.
- При проведении A/A теста была подтверждена правильность проведения теста, статистически значимой разницы между контрольными группами не было обнаружено.
- По результатам проведения A/B теста как с каждой из контрольных групп пользователей, так и с объединенной, статистически значимых различий в долях пользователей, переходящих на следующий экран приложения **не выявлено**.

С учетом результатов проведенного анализа можно заключить, что изменения, проверянные в экспериментальной группе 248 **не отразились на конверсии пользователей приложения в покупателей продуктов**.

Предалагается проверка следующей гипотезы и проведение A/B теста для нее с целью нахождения оптимального варианта увеличения конверсии.