Алгоритм проведения анализа данных. Примеры и реализация.

Проведения анализа данных включает в себя следующие этапы:

- **Постановка задачи.** На данном этапе вырабатывается понимание предметной области, поиск данных и общий подход к решению задачи.
- **Изучение данных.** Включает в себя разведовательный анализ данных, профайлинг, оценку полноты данных и добавление показателей при необходимости.
- **Очистка данных.** Работа с пропущенными значениями, дубликатами, выбросами.
- **Преобразование данных.** Приведение типов данных, нормализация, создание новых признаков и обогащение данных.
- **Решение задачи.** Решение задачи может быть представлено в различных видах: статья, дашборды, обученная модель

Пример 1. Дашборд по продажам кофе

Описание задачи. Иван Иванов недавно открыл кофейную лавку: арендовал помещение, нанял сотрудника-баристу, купил терминал и огромное количество молока. Все обязанности по обслуживанию клиентов он возложил на баристу, а сам сидел в кабинете и строил планы на бизнес. Однажды с утра сотрудник бариста опаздал к открытию и Иванову пришлось самостоятельно стоять за прилавком, в результате к нему в магазин забежал лишь один бизнесмен, заказавший эспрессо с оплатой наличными. После данного случая Иванов Иван задумался: а не зря ли он купил терминал и тысячу литров молока, оправдается ли его покупка, если всех кого он видел это бизнесмен с наличкой и эспрессо? Для решения данной задачи он нанял аналитиков, которым предоставил данные из кассы и попросил посмотреть как часто совершают покупки картой, следует ли ему закупать молоко и будет ли успешен его бизнес в будущем. Результаты он хотел видеть каждый месяц в формате картинки для удобства.

In [1]:
```
import pandas as pd # Импортируем библиотеку для работы с данными
import plotly.graph_objects as go # Библиотека для работы с визуализацией
from plotly.subplots import make_subplots
df = pd.read_csv('index.csv') # Импортируем данные о продаже кофе
# Подробнее по ссылке: https://www.kaggle.com/datasets/ihelon/coffee-sales/data
df.head() # Проверка загрузки
```

Out[1]:

	date	datetime	cash_type	card	money	coffee_name
0	2024-03-01	2024-03-01 10:15:50.520	card	ANON-0000-0000-0001	38.7	Latte
1	2024-03-01	2024-03-01 12:19:22.539	card	ANON-0000-0000-0002	38.7	Hot Chocolate
2	2024-03-01	2024-03-01 12:20:18.089	card	ANON-0000-0000-0002	38.7	Hot Chocolate
3	2024-03-01	2024-03-01 13:46:33.006	card	ANON-0000-0000-0003	28.9	Americano
4	2024-03-01	2024-03-01 13:48:14.626	card	ANON-0000-0000-0004	38.7	Latte

In [2]:
```
df.info() # Проверка на пропущенные значения
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2623 entries, 0 to 2622
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   date         2623 non-null   object 
 1   datetime     2623 non-null   object 
 2   cash_type    2623 non-null   object 
 3   card         2534 non-null   object 
 4   money        2623 non-null   float64
 5   coffee_name  2623 non-null   object 
dtypes: float64(1), object(5)
memory usage: 123.1+ KB
```

In [3]:
```
print("Дублирующихся строк: ", df[df.duplicated()].shape[0]) # Проверка на дубликаты
```

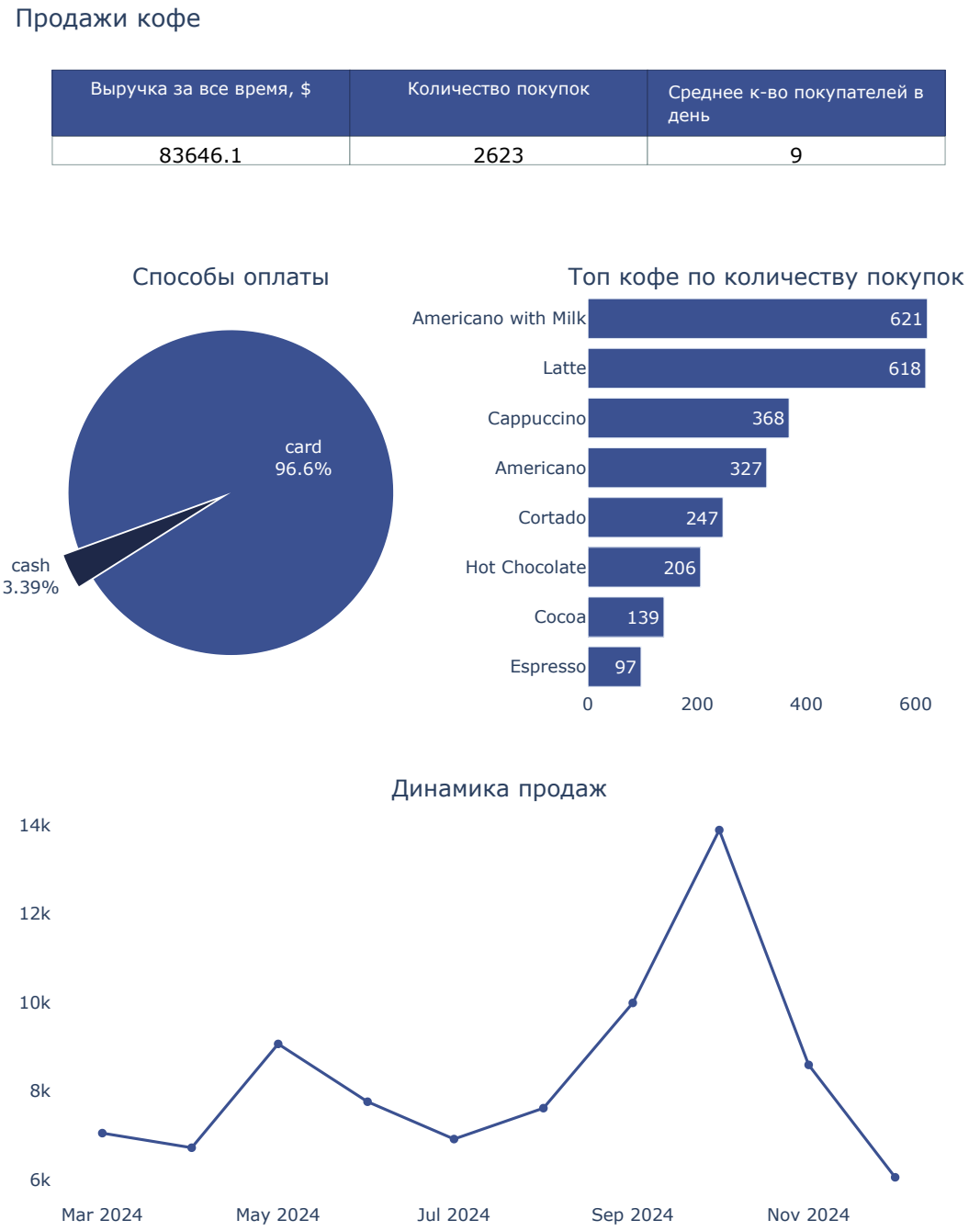
```
Дублирующихся строк:  0
```

In [4]:

```
# Создадим для каждой визуализации новые наборы данных
df_main_info = pd.DataFrame([[df['money'].sum(), df['money'].count(), round(df['money'].count()/len
(list(df['date'].unique())))]],
                            columns = ["Выручка за все время, $", "Количество покупок", "Среднее к
-во покупателей в день"]) # Датафрейм с основными показателями
df_cash_sum = df[['cash_type', 'money']].groupby(by = 'cash_type', as_index= False).count() # Датаф
рейм с методами оплаты
df_coffe_count = pd.DataFrame(df['coffee_name'].value_counts(), columns = ['count']).reset_index()
# Датафрейм с разбивкой по видам кофе
df_coffe_count = df_coffe_count.sort_values(by = 'count')
df['month'] = df['date'].apply( lambda x: x[:7]) # Дополнительный показатель месяца покупки
df_date_money = df[['month', 'money']].groupby(by = 'month', as_index= False).sum() # Датафрейм с с
уммой покупок за месяц
```

In [5]:

```
colors = ['#3b5191', '#1e2848', 'white'] # Задаем цветовую палитру
fig = make_subplots( # Используем фукнцию для совмещения графиков
    rows=3, cols=2, # Задаем сетку для графиков
    vertical_spacing=0.1, # Настройка расстояний
    horizontal_spacing = 0.2,
    column_widths = [1,1],
    row_heights = [0.25,0.9,0.9],
    specs= [[{"type": "table",'colspan': 2}, None], # Параметры графиков в сетке
            [{"type": "pie"}, {"type": "bar"}],
            [{"type": "scatter",'colspan': 2}, None]],
    subplot_titles=[None,'Способы оплаты', 'Топ кофе по количеству покупок', 'Динамика продаж'] # Названия графиков
)

fig.add_trace( # Метод add_trace добавляет график на сетку
    go.Table( # Добавляем таблицу
        header=dict( # Настройка шапки таблицы
            values=["Выручка за все время, $", "Количество покупок", "Среднее к-во покупателей в день"], # Названия столбцов
            font=dict(color=colors[2], size=12), # Настройка шрифта
            align='center', # Выравнивание
            line_color=colors[1], # Цвет обводки
            fill_color=colors[0] # Цвет шапки таблицы
        ),
        cells=dict( # Настройка ячеек таблицы
            values=[df_main_info[k].tolist() for k in df_main_info.columns[0:]], # Значения внутри таблицы
            align =  'center', # Выравнивание
            line_color='darkslategray', # Цвет обводки
            fill_color = colors[2], # Цвет заполнения ячеек
            font=dict(color='black', size=14)) # Настройка текста
    ),
    row=1, col=1 # Указываем место на сетке графиков
)

fig.add_trace( # Добавляем круговую диаграмму
    go.Pie(labels=df_cash_sum['cash_type'], # Подписи
           values=df_cash_sum['money'], # Значения
           marker_colors=colors[0:2], # Цвет секторов круговой диаграммы
           rotation = 250, # Вращение
           textinfo='label+percent', # Подписи секторов
           pull=[0, 0.1]), # Регулировка выдвижения сектора
    row=2, col=1
)

fig.add_trace( # Добавляем гистрограмму
    go.Bar(x=df_coffe_count['count'], # Значения по оси x
           y=df_coffe_count['coffee_name'], # Значения по оси y
           marker=dict(color=colors[0]), # Цвет графика
           orientation = 'h', # Ориентация графика
           text=df_coffe_count['count'], # Подписи внутри столбцов
           textposition='auto', # Расположение текста
           showlegend=False), # Выключаем отображение легенды
    row=2, col=2
)

fig.add_trace(
    go.Scatter( # Добавляем линейчатую диаграмму
        x=df_date_money["month"], # Значения по x
        y=df_date_money["money"], # Значения по y
        mode='lines+markers', # Выбираем вид графика - линия
        line_color=colors[0] # Задаем цвет линии

    ),
    row=3, col=1
)

fig.update_layout( # Дополнительная настройка объекта
    height=900, width= 720, # Высота и ширина
    showlegend=False, # Отключаем отображение легенды
    title_text="Продажи кофе", # Название графика
    paper_bgcolor=colors[2], # Цвет фона
    plot_bgcolor=colors[2], # Цвет фона графиков
    margin=dict(l=0, r=35, t=60, b=50) # Настройка отступов
)

fig.show()
```



В результате мы построили простейший дашборд, отвечающий на все вопросы заказчика: терминал оставляем, кофе с молоком пользуется популярностью. Дальнейшее развитие задачи: согласование дашборда с заказчиком, определение формата предосталения (картинкой на почту, или в папку на гугл диске и т.д), какой период использовать при отправке дашборда (за все время, или, например, за последние 3 месяца)

Задание 1

В график "Динамика продаж" добавить прогнозные значения на 3 месяца вперед, выделить отдельным цветом. (Необходимо учитывать, что декабрь не полный месяц)

Пример 2. A/B тестирование

Описание задачи. Иванов Иван решил расширять свой кофейный бизнес через мобильные игры. Он заказал у разработчиков игру, в которой покупатели ловят промокоды на кофе и решил, что изменение цвета кнопки "Играть" на красный привлечет еще больше игроков, а следовательно и покупателей в его кофейный магазин. Оказалось, что изменение цвета кнопки будет стоить Иванову очень дорого и он бы не хотел так рисковать без подтвержений своих догадок. Поэтому он снова нанял команду аналитиков, у которых попросил помочь ему принять решение: обновлять игру или нет. Для этого аналитики попросили в тестовом режиме у половины пользователей поменять цвет кнопки и предоставить им следующие данные: идентификатор пользователя, версия игры, количество игр, вернулся ли пользователь в течении одного дня, вернулся ли пользователь в течении недели.

In [6]:
```
import plotly.express as px
df = pd.read_csv('Cookie_Cats_cleaned_v01.csv') # Импортируем данные о мобильной игре
# Подробнее по ссылке: https://www.kaggle.com/datasets/matinmahmoudi/rounds-and-retention
df.head() # Проверка загрузки
```

Out[6]:

	userid	version	sum_gamerounds	retention_1	retention_7
0	116	gate_30	3	False	False
1	337	gate_30	38	True	False
2	377	gate_40	165	True	False
3	483	gate_40	1	False	False
4	488	gate_40	179	True	True

In [7]:
```
df.info() # Выводим информацию о данных, проверяем наличие пропущенных значений
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90189 entries, 0 to 90188
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   userid          90189 non-null  int64 
 1   version         90189 non-null  object
 2   sum_gamerounds  90189 non-null  int64 
 3   retention_1     90189 non-null  bool  
 4   retention_7     90189 non-null  bool  
dtypes: bool(2), int64(2), object(1)
memory usage: 2.2+ MB
```

In [8]:
```
print("Дублирующихся строк: ", df[df.duplicated()].shape[0]) # Проверка дубликатов
df.set_index('userid', inplace=True) # Идентификатор пользователя переносим в индекс
```

```
Дублирующихся строк:  0
```

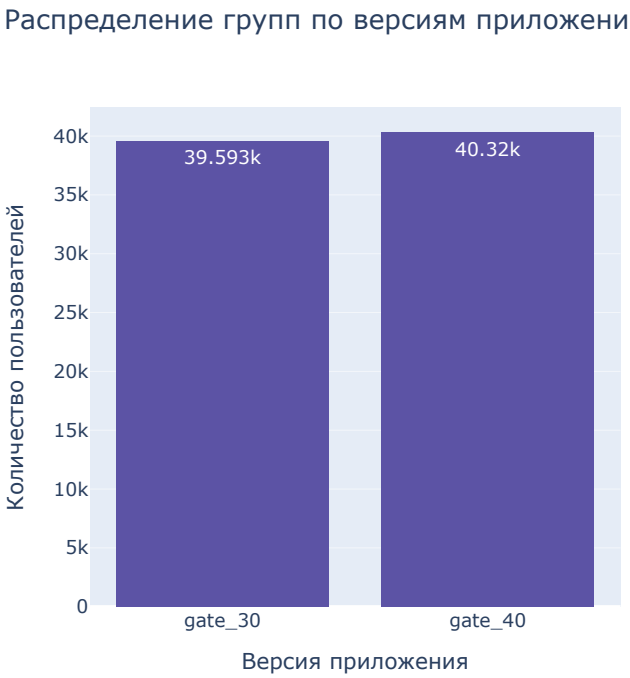
In [9]:
```
df.describe() # Выводим описательную статистику по числовым параментрам
# Заметим, что максамальное значение сильно отличается от квартилей, следовательно, является выброс
ом
```

Out[9]:

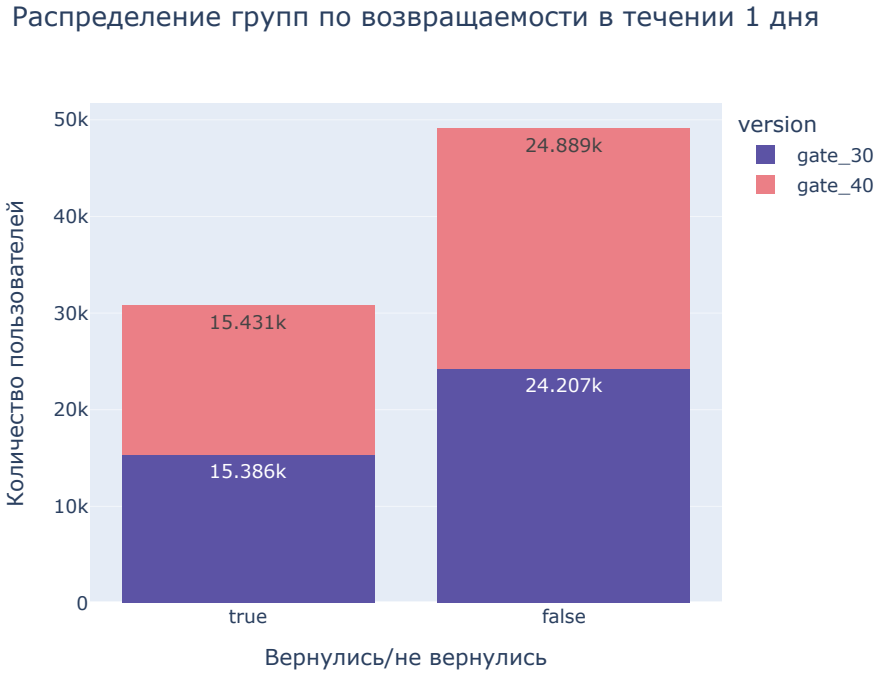
	sum_gamerounds
count	90189.000000
mean	51.872457
std	195.050858
min	0.000000
25%	5.000000
50%	16.000000
75%	51.000000
max	49854.000000

In [10]:
```
# Удалим выбросы по методу межквартильного размаха
Q1 = df['sum_gamerounds'].quantile(0.25) # Расчитываем нижний квартиль
Q3 = df['sum_gamerounds'].quantile(0.75) # Расчитываем верхний квартиль
IQR = Q3 - Q1 # Межквартильный размах
df = df[(df['sum_gamerounds'] < Q3 + 1.5 * IQR) & (df['sum_gamerounds'] > Q1 - 1.5 * IQR)]
```

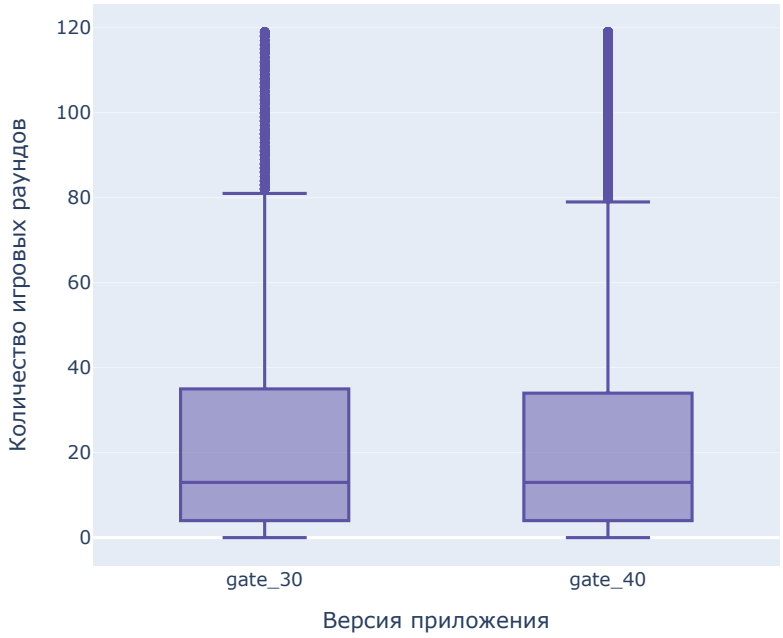
In [11]:
```
# Проведем EDA в разрезе групп
fig = px.histogram(df, x="version",
                   width=500, height=500, title="Распределение групп по версиям приложения" ,
                   color_discrete_sequence=px.colors.sequential.Sunset[::-3], text_auto=True)
fig.update_xaxes(title = 'Версия приложения')
fig.update_yaxes(title = 'Количество пользователей')
fig.show()
# Группы по версионности разделены равномерно, следовательно данные подходят для A/B тестирования
```



In [12]:
```
fig = px.histogram(df, x="retention_1", color= 'version',
                   width=600, height=500, title="Распределение групп по возвращаемости в течении 1 дня" ,
                   color_discrete_sequence=px.colors.sequential.Sunset[::-3], text_auto=True)
fig.update_xaxes(categoryorder='total ascending', title = 'Вернулись/не вернулись')
fig.update_yaxes(title = 'Количество пользователей')
fig.show()
# возвращаемость в зависимости от группы распределена равномерно, 37,5% пользователей вернулись в приложении в течении 1 дня с момента установки
```



In [13]:
```
fig = px.box(df, x="version", y="sum_gamerounds", color_discrete_sequence=px.colors.sequential.Sunset[::-3], width=600, height=500)
fig.update_xaxes( title = 'Версия приложения')
fig.update_yaxes(title = 'Количество игровых раундов')
fig.show()
# Показатель количества игровых раундов распределен неравномерно в каждой группе, имеются выбросы и ассиметрия
# Данные выводы следует учитывать при проведении тестирования
```



Далее необходимо определить вопросы и составить гипотезы для A/B тестирования. Мы будем тестировать две группы:

- пользователи, играющие в старую версию приложения gate_30 (группа 1 или контрольная группа)
- пользователи, играющие с новой версией приложения gate_40 (группа 2 или тестовая группа)

Вопрос 1 (качественные данные). При обновлении приложения увеличилось ли удержание игроков на 2% через 1 день (то есть стали ли игроки возвращаться чаще с нововведения)?
Нулевая гипотеза (H0): коэффициент удержания игроков через день после установки в группе 2 не увеличился на 2% по сравнению с группой 1.
Атернативная гипотеза (H1): коэффициент удержания игроков через день после установки в группе 2 увеличился на 2% по сравнению с группой 1.

Вопрос 2 (количественные данные). При обновлении приложения увеличилось ли среднее количество игровых сессий на 5 сеансов (то есть стали ли пользователи из группы 2 играть чаще)?
Нулевая гипотеза (H0): среднее количество игровых сеансов в группе 2 не увеличилось на 5 сеансов по сравнению с контрольной группой.
Альтернативная гипотеза (H1): Среднее количество игровых сеансов в группе 2 увеличилось на 5 сеансов по сравнению с группой 1.

In [14]:
```
import scipy.stats as stats # Импортируем статистический модуль
import statsmodels.stats.api as sms
import numpy as np

group_1 = df[df['version'] == 'gate_30'] # Определяем группу 1 (без обновлений)
group_2 = df[df['version'] == 'gate_40'] # Определяем группу 2 (с обновлениями)
alpha = 0.05 # Задаем ошибку первого рода для проверки статистических гипотез
beta = 0.20 # Ошибка второго рода
power = 1 - beta # Мощность критерия - вероятность обнаружения эффекта, если он действительно существует
```

Вопрос 1

In [15]:
```
# Для начала рассчитаем минимальный размер выборки, который нужно взять для наблюдения желаемого эффекта
crosstab_retention = pd.crosstab(df['version'], df['retention_1'], normalize='index') # Строим таблицу частот, метод normalize='index' переводит числа в относительную шкалу
p1_retention = crosstab_retention.loc['gate_30', True] # Извлекаем метрику, от которой будем отталкиваться, в данном случае это процент пользователей из группы 1, которые вернулись
p2_retention = p1_retention + 0.02  # Изменение нашей метрики, т.е. увеличение на 2%
# Расчитываем необходимое к-во наблюдений
effect_size = sms.proportion_effectsize(p1_retention, p2_retention)
n_retention = sms.NormalIndPower().solve_power(effect_size=effect_size, power=power, alpha=alpha, ratio=1)
print("Размер выборки:", int(n_retention))
```

```
Размер выборки: 9406
```

In [16]:
```
# Рандомно отбираем наблюдения из групп
np.random.seed(42)
group_1_retention = group_1.sample(n=int(n_retention), random_state=42)
group_2_retention = group_2.sample(n=int(n_retention), random_state=42)
```

In [17]:
```
from statsmodels.stats.proportion import proportions_ztest # Импортируем метод Z-теста для пропорций
n_obs_retention = [len(group_2_retention), len(group_1_retention)] # Общее к-во наблюдений для группы 1 и группы 2
successes_retention = [group_2_retention['retention_1'].sum(), group_1_retention['retention_1'].sum()] # Количество вернувшихся пользователей для группы 1 и 2
z_stat_retention, p_val_retention = proportions_ztest(successes_retention, n_obs_retention, value=0, alternative='larger') # Проводим Z-тест для пропорций
print(f"Z-статистика: {z_stat_retention}")
print(f"P-value: {p_val_retention}")
if p_val_retention < alpha:
    print("Отвергаем нулевую гипотезу. Уровень возвратившихся пользователей через 1 день вырос")
else:
    print("Нет оснований отвергнуть нулевую гипотезу, уровень возратившихся пользователей через 1 день не изменился.")
```

```
Z-статистика: -1.7207238612991111
P-value: 0.9573495278312576
Нет оснований отвергнуть нулевую гипотезу, уровень возратившихся пользователей через 1 день не изменился.
```

Вопрос 2

In [18]:
```
# Определим размер выборки
effect_size_sessions = 5 # Задаем количество игровых сессий, увеличение на которое мы хотели бы наблюдать
std_dev_sessions = pd.concat([group_1['sum_gamerounds'],group_2['sum_gamerounds']]).std() # Стандартное отклонение между группами
n_sessions = sms.NormalIndPower().solve_power(effect_size=effect_size_sessions / std_dev_sessions, power=power, alpha=alpha, ratio=1) # Вычисляем размер выборки
print("Размер выборки:", int(n_sessions))
```

```
Размер выборки: 467
```

In [19]:
```
# Рандомно отбираем наблюдения из групп
group_1_gamerounds = group_1.sample(n=int(n_sessions), random_state=42)
group_2_gamerounds = group_2.sample(n=int(n_sessions), random_state=42)
```

In [20]:
```
from scipy.stats import ttest_ind, levene # Импортируем t-тест и тест Левена

mean_group_1_gamerounds = group_1_gamerounds['sum_gamerounds'].mean() # Среднее в 1 группе
mean_group_2_gamerounds = group_2_gamerounds['sum_gamerounds'].mean() # Среднее в 2 группе
std_group_1_gamerounds = group_1_gamerounds['sum_gamerounds'].std() # Стандартное отклонение в 1 гр
уппе
std_group_2_gamerounds = group_2_gamerounds['sum_gamerounds'].std() # Стандартное отклонение в 2 гр
уппе

# Проводим тест Левена, который проверяет равенство дисперсий двух выборок
stat, p_levene = levene(group_1_gamerounds['sum_gamerounds'], group_2_gamerounds['sum_gamerounds'])
# Если условие равенства дисперсий выполняется, то мы можем выполнять t-тест
if p_levene < alpha:
    print("Допущение не выполнено: дисперсии в двух группах не равны. Переходим к t-критерию Уэлч
а")
    equal_var = False
else:
    print("Допущение выполнено: дисперсии в двух группах равны.")
    equal_var = True
# Выполняем односторонний t-тест для сравнения двух групп
t_stat_gamerounds, p_val_gamerounds = ttest_ind(group_2_gamerounds['sum_gamerounds'], group_1_gamer
ounds['sum_gamerounds'], equal_var=equal_var, alternative='greater')
print(f"T-статистика: {t_stat_gamerounds}")
print(f"P-value: {p_val_gamerounds}")
if p_val_gamerounds < alpha:
    print("Отвергаем нулевую гипотезу: среднее количество игровых сессий в группе 2 значительно бол
ьше, чем в группе 1.")
else:
    print("Нет оснований отвергать нулевую гипотезу: среднее количество игровых сессий в группе 2 н
енамного больше, чем в группе 1.")
```

```
Допущение выполнено: дисперсии в двух группах равны.
T-статистика: 0.5607755623342829
P-value: 0.28754266646381954
Нет оснований отвергать нулевую гипотезу: среднее количество игровых сессий в группе 2 ненамного бо
льше, чем в группе 1.
```

В результате мы установили, что обновления в приложении не оказывает влияния на возвращаемость пользователей и количество игровых сессий, следовательно, перекрашивать кнопку нет необходимости.

Задание 2

Провести A/B тест для данных возвращения пользователей в течении 7 дней (retention_7).