In [1]:
```
from math import asin
from typing import Union
from tqdm import tqdm

import pandas as pd
import numpy as np

from scipy import stats
from statsmodels.stats.meta_analysis import effectsize_smd
from statsmodels.stats import proportion
from statsmodels.stats.power import tt_ind_solve_power
from statsmodels.stats.power import zt_ind_solve_power
```

Python применение продвинутых методов

Задача. Проанализируйте результаты эксперимента и напишите свои рекомендации менеджеру.

Mobile Games AB Testing with Cookie Cats

In [2]:
```
df = pd.read_csv('gb_sem_9_hw.csv')
df.head()
```

Out[2]:

	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 [3]:
```
df.describe()
```

Out[3]:

	userid	sum_gamerounds
count	9.018900e+04	90189.000000
mean	4.998412e+06	51.872457
std	2.883286e+06	195.050858
min	1.160000e+02	0.000000
25%	2.512230e+06	5.000000
50%	4.995815e+06	16.000000
75%	7.496452e+06	51.000000
max	9.999861e+06	49854.000000

In [4]:
```
df.shape
```

Out[4]:
```
(90189, 5)
```

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

```
<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 [6]:
```
df.userid.nunique() # все userid уникальны
```

Out[6]:
```
90189
```

In [7]:
```
df.groupby('version').count() # сплитование прошло равномерно
```

Out[7]:

version	userid	sum_gamerounds	retention_1	retention_7
gate_30	44700	44700	44700	44700
gate_40	45489	45489	45489	45489

In [8]:
```
control_gr = df[df['version'] == 'gate_30'].copy(deep = True)
test_gr = df[df['version'] == 'gate_40'].copy(deep=True)
```

In [9]:
```
control_gr.shape, test_gr.shape
```

Out[9]:
```
((44700, 5), (45489, 5))
```

In [10]:
```
def continious_result(control: pd.DataFrame,
                      treatment: pd.DataFrame,
                      column: str,
                      n_iters: int = 10_000) -> pd.DataFrame:
    # Статистика по выборкам
    size = control.loc[:, column].shape[0]

    control_mean = control.loc[:, column].mean()
    treatment_mean = treatment.loc[:, column].mean()

    control_std = control.loc[:, column].std(ddof=1)
    treatment_std = treatment.loc[:, column].std(ddof=1)

    # Бутсрап
    booted_diff = []
    for _ in tqdm(range(n_iters)):
        control_sample = control.loc[:, column].sample(n=size, replace=True).values
        treatment_sample = treatment.loc[:, column].sample(n=size, replace=True).values
        booted_diff.append(np.mean(control_sample - treatment_sample))

    # Считаем статистику после бустрапа
    md_ci, std_ci = np.mean(booted_diff), np.std(booted_diff, ddof=1)
    left_ci, right_ci = np.percentile(booted_diff, [2.5, 97.5])
```

```
    p_value_ci = 2 * (1 - stats.norm.cdf(np.abs(md_ci / std_ci)))

    # Считаем мощность эксперимента
    effect_size, _ = effectsize_smd(mean1=treatment_mean, sd1=treatment_std, nobs1=size,
                                    mean2=control_mean, sd2=control_std, nobs2=size)
    power = tt_ind_solve_power(effect_size=effect_size,
                               nobs1=size,
                               alpha=.05,
                               power=None,
                               ratio=1)
    # Формируем отчёт
    result = pd.DataFrame({'effect_size': effect_size,
                           'alpha': p_value_ci,
                           'beta': (1-power),
                           'CI': f'[{np.round(left_ci, 3)}, {np.round(right_ci, 3)}]',
                           'difference': md_ci,},
                          index=[column])
    return result
```

In [11]:

```
def proportion_result(control: pd.DataFrame,
                      treatment: pd.DataFrame,
                      column: str,
                      n_iters: int = 10_000) -> pd.DataFrame:
    # Вероятность событий
    size = control.loc[:, column].shape[0]
    prop_control = control.loc[:, column].sum() / size
    prop_treatment = treatment.loc[:, column].sum() / size

    # Бутсрап
    booted_diff = []
    for _ in tqdm(range(n_iters)):
        control_sample = stats.bernoulli.rvs(p=prop_control, size=size)
        treatment_sample = stats.bernoulli.rvs(p=prop_treatment, size=size)
        booted_diff.append(np.mean(control_sample - treatment_sample))

    # Считаем статистику после бустрапа
    md_ci, std_ci = np.mean(booted_diff), np.std(booted_diff, ddof=1)
    left_ci, right_ci = np.percentile(booted_diff, [2.5, 97.5])
    p_value_ci = 2 * (1 - stats.norm.cdf(np.abs(md_ci / std_ci)))

    # Считаем мощность эксперимента
    effect_size = proportion.proportion_effectsize(prop_control, prop_treatment)

    power = zt_ind_solve_power(effect_size=effect_size,
                               nobs1=size,
                               alpha=.05,
                               power=None,
                               ratio=1)
    # Формируем отчёт
    result = pd.DataFrame({'effect_size': effect_size,
                           'alpha': p_value_ci,
                           'beta': (1-power),
                           'CI': f'[{np.round(left_ci, 3)}, {np.round(right_ci, 3)}]',
                           'difference': md_ci,},
                          index=[column])
    return result
```

In [12]:

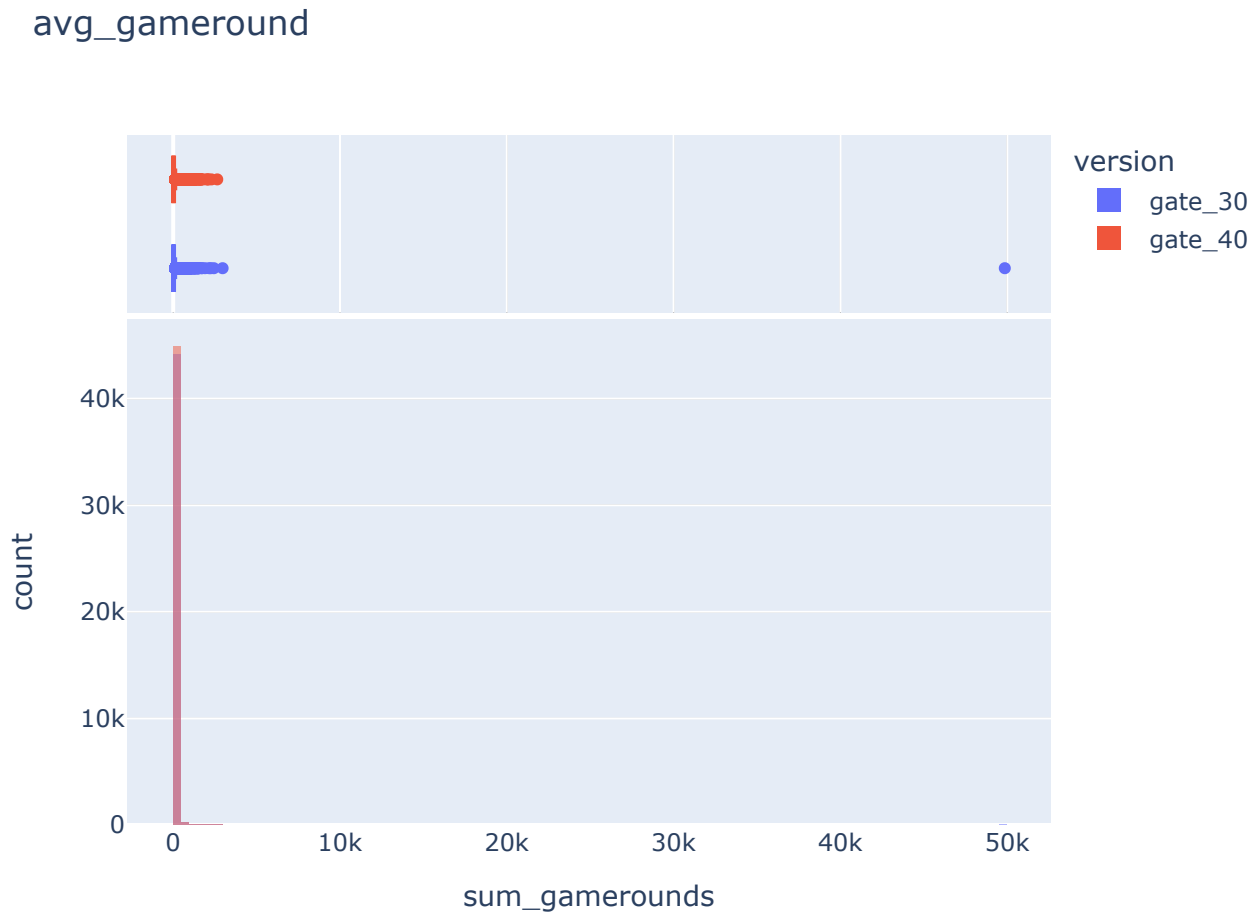
```
import plotly.express as px
```

1. проверяем метрику sum_gamerounds

In [13]:
```
# посмотрим зависимость sum_gamerounds от группы

fig = px.histogram(df,
                   x='sum_gamerounds',
                   color = 'version',
                   title='avg_gameround',
                   marginal = 'box',
                   nbins = 100,
                   barmode='overlay')

fig.show()
```



In [ ]:
```
```

На графике почти нет различий по группам по метрике sum_gamerounds

In [14]:
```
# sum_gamerounds - Количественная метрика
continious_result(control_gr, test_gr, 'sum_gamerounds')
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 10000/10
000 [00:58<00:00, 169.86it/s]
```

Out[14]:

	effect_size	alpha	beta	CI	difference
sum_gamerounds	-0.005915	0.367343	0.856725	[-0.963, 4.105]	1.18241

In [15]:
```
# разбиваем на корзины
a=[]
for i in range(100, 1001):
    if df.shape[0] % i == 0:
        print(i)
        a.append(i)
print(a)
```

```
911
[911]
```

In [16]:
```
# создаем новую таблицу с корзинами
n_buckets = a[0]
df_b = (df
 .sample(n=df.shape[0], replace=False)
 .reset_index(drop=True)
 .assign(bucket=list(range(n_buckets)) * int(df.shape[0] / n_buckets))).copy(deep = True)
```

In [17]:
```
df_b.head(3)
```

Out[17]:

	userid	version	sum_gamerounds	retention_1	retention_7	bucket
0	2723521	gate_30	5	True	False	0
1	6395727	gate_40	26	True	False	1
2	8738240	gate_30	11	False	False	2

In [ ]:
```
```

In [18]:
```
# создаем новую таблицу из средних данных по корзинам по метрике sum_gamerounds
bucketed_df1 = df_b.groupby(['version', 'bucket'])['sum_gamerounds'].agg(mu=np.mean, std=r
bucketed_df1.shape
```

Out[18]:
```
(1822, 4)
```

In [19]:
```
bucketed_df1
```

Out[19]:

	version	bucket	mu	std
0	gate_30	0	94.854545	316.518703
1	gate_30	1	56.888889	93.975843
2	gate_30	2	67.262295	139.155058
3	gate_30	3	48.551020	67.827619
4	gate_30	4	40.948276	50.355377
...	...	...	...	...
1817	gate_40	906	43.340426	68.032050
1818	gate_40	907	57.038462	80.978564
1819	gate_40	908	46.245283	72.460044
1820	gate_40	909	48.851064	80.571863
1821	gate_40	910	75.055556	166.501450

1822 rows × 4 columns

In [20]:
```
bucketed_df1.groupby('version').count()
```

Out[20]:

version	bucket	mu	std
gate_30	911	911	911
gate_40	911	911	911

In [21]:
```
# Сравним исходное выборочное среднее и среднее бакетных средних
round(np.mean(df_b["sum_gamerounds"]), 1), round(np.mean(bucketed_df1["mu"]), 1)
```

Out[21]:
```
(51.9, 52.0)
```

In [22]:
```
round(np.std(df_b["sum_gamerounds"]), 1), round(np.mean(bucketed_df1["std"]), 1)
# стандартные отклонения несильно отличаются. Разбиение на корзины НЕ возможно
```

Out[22]:
```
(195.0, 97.5)
```

In [23]:
```
# ради интереса, проверимЮ будут ли отличаться данные бакетированные от основной выборки
```

In [24]:
```
control_bucket = bucketed_df1[bucketed_df1.version == 'gate_30']
treatment_bucket = bucketed_df1[bucketed_df1.version == 'gate_40']
continious_result(control_bucket, treatment_bucket, 'mu')
```

```
100%|██████████████████████████████████████████████████████████████████████████| 10000/100
00 [00:04<00:00, 2058.38it/s]
```

Out[24]:

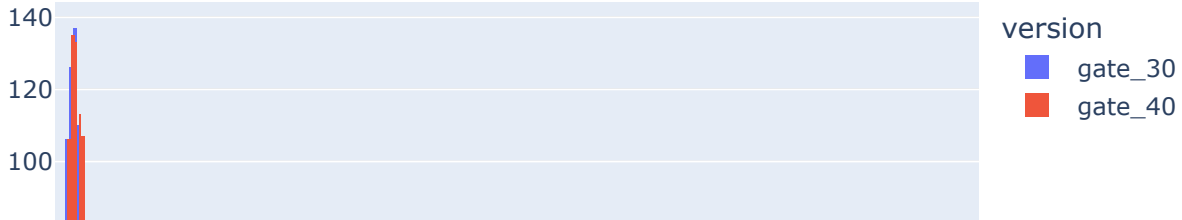
	effect_size	alpha	beta	CI	difference
mu	-0.040747	0.388142	0.859984	[-1.067, 4.639]	1.286524

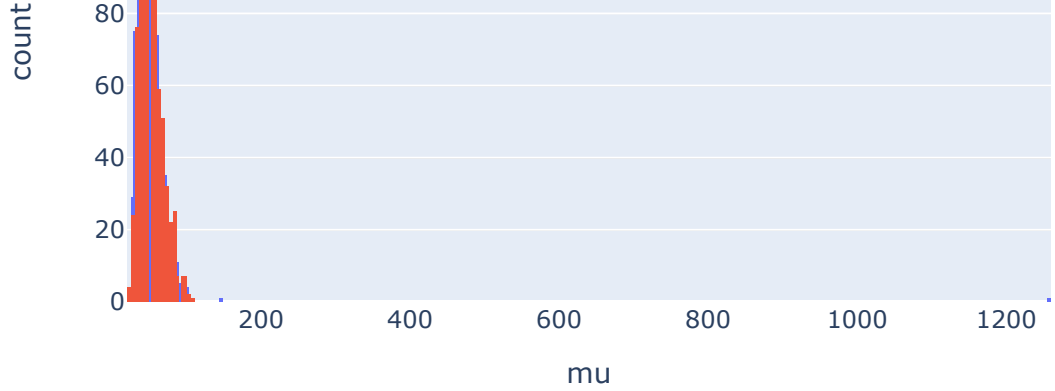
результаты практически совпадают с результатами по начальной таблице

effect_size alpha beta CI difference

sum_gamerounds -0.005915 0.372362 0.856725 [-0.98, 4.115] 1.17467

In [25]:
```
fig = px.histogram(bucketed_df1, x="mu",
                   color='version', barmode='group',
                   height=400)
fig.show()
```





ВЫВОД1.

альфа > 0.05. Очень высокая мощность теста >80%. sum_gamerounds = mean 51.872457. У нас разница 1.17467. Довертиельный интервал немного захватывает 0. Статистически значемых отличий нет, принимаем нулевую гипотезу. На количество игровых раундом тест не повлиял.

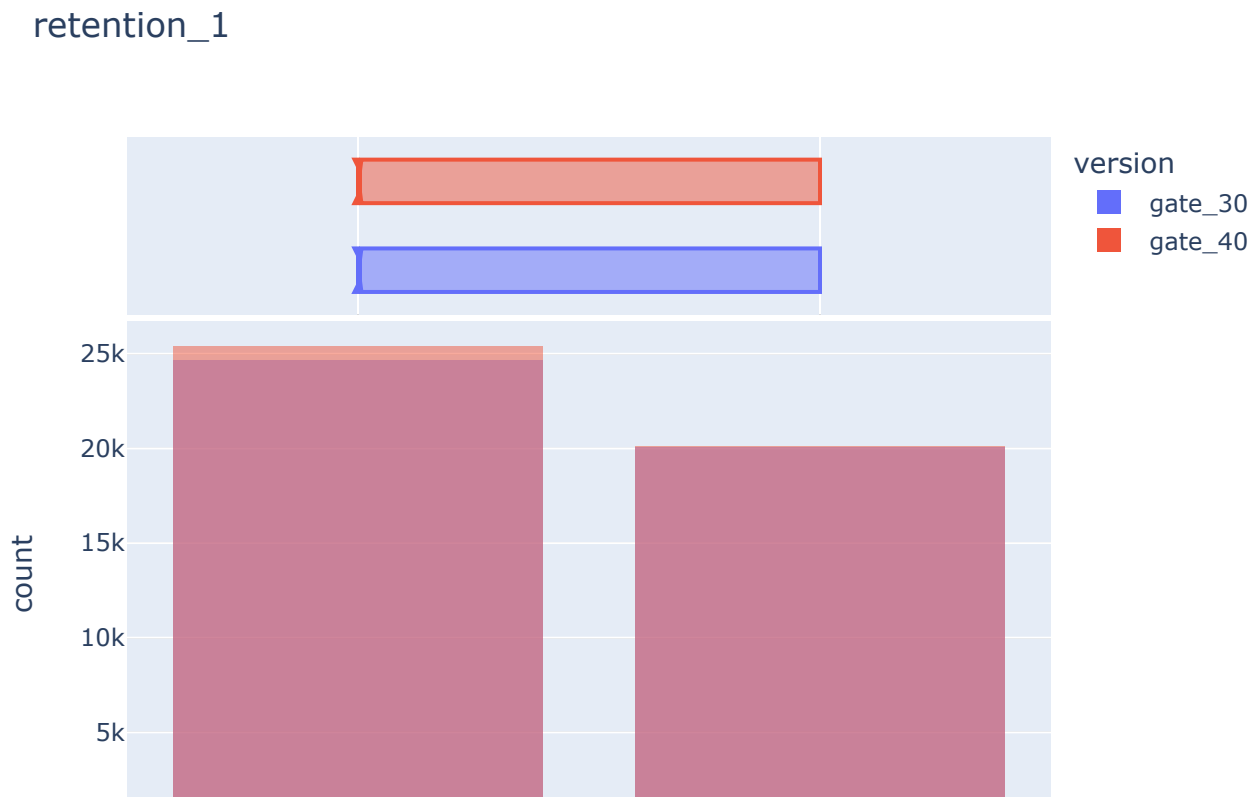
2. проверяем метрику retention_1

In [26]:

```
# посмотрим зависимость retention_1 от группы

fig = px.histogram(df,
                   x='retention_1',
                   color = 'version',
                   title='retention_1',
                   marginal = 'box',
                   nbins = 100,
                   barmode='overlay')

fig.show()
```



In [27]:
```
fig = px.histogram(df_b, x="retention_1",
                   color='version', barmode='group',
                   height=400)
fig.show()
```

метрика качественная - разбить на корзины нельзя

In [28]:
```
proportion_result(control_gr, test_gr, 'retention_1')
```

```
100%|██████████████████████████████████████████████████████████████████████████| 10000/10000 [00:39<00:00, 252.84it/s]
```

Out[28]:

	effect_size	alpha	beta	CI	difference
retention_1	-0.003823	0.573981	0.911819	[-0.008, 0.005]	-0.001861

Вывод 2.

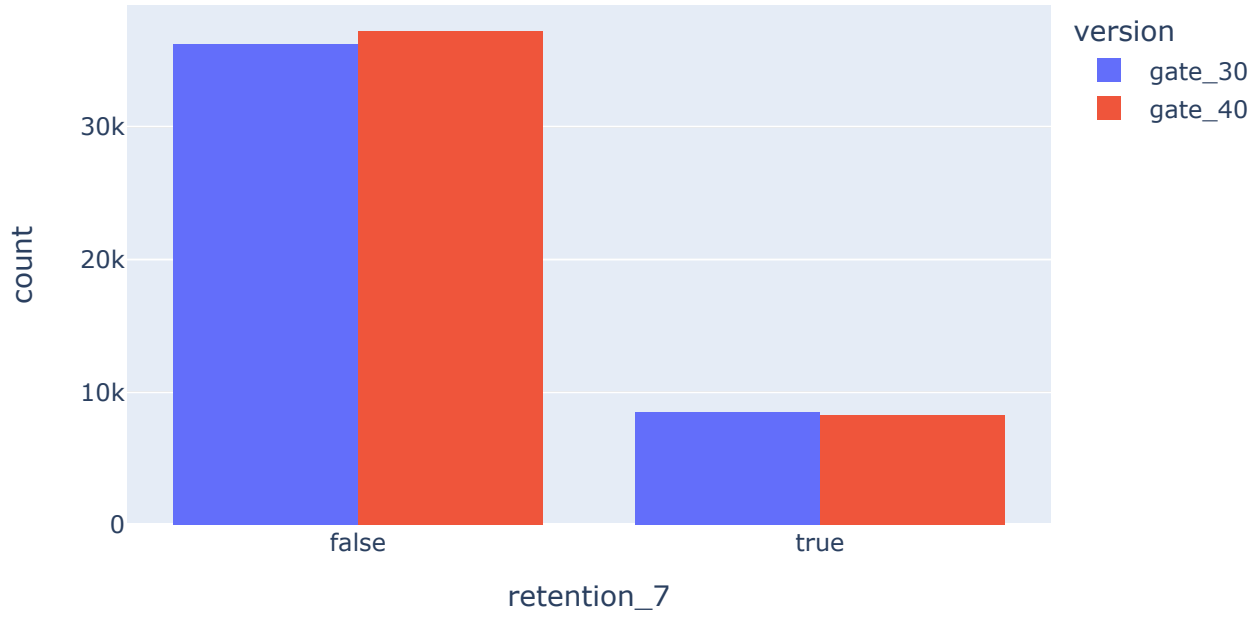
a > 0.05. beta > 80 очень высокая мощность. Доверительный интервал около 0 , Разницы почти нет
вывод - принимаем нулевую гипотезу - статистически значимых различий нет

3. проверяем метрику retention_7

In [29]:
```
# посмотрим зависимость retention_7 от группы

fig = px.histogram(df_b, x="retention_7",
```

```
                    color='version', barmode='group',
                    height=400)
fig.show()
```



In [30]:
```
# на графиках разница незначительная
```

In [31]:
```
proportion_result(control_gr, test_gr, 'retention_7')
```

```
100%|██████████████████████████████████████████████████████████████████████████| 10000/10000 [00:31<00:00, 322.14it/s]
```

Out[31]:

	effect_size	alpha	beta	CI	difference
retention_7	0.012776	0.052398	0.519844	[-0.0, 0.01]	0.005007

Вывод

a > 0.05. beta > 50 средняя мощность. Доверительный интервал около 0 , Разницы почти нет вывод - принимаем нулевую гипотезу - статистически значимых различий нет

Общий вывод - гипотеза не позволила улучишь ни одну метрику.

требуется формирование других гипотез.

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:

In [ ]: