GameEngagementExperiment

March 9, 2024

[54]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import seaborn as sns
from scipy.stats import shapiro, ttest_ind, mannwhitneyu, chi2_contingency
from scipy import stats
```

[2]:
```
game_data = pd.read_csv("cookie_cats.csv")
```

[3]:
```
game_data.head()
```

[3]:

	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

[4]:
```
game_data.dtypes
```

[4]:
```
userid              int64
version            object
sum_gamerounds      int64
retention_1          bool
retention_7          bool
dtype: object
```

[5]:
```
# Check for duplicate User IDs
has_duplicates = game_data.duplicated(subset=['userid']).any()

print("Dataset contains duplicate User IDs:", has_duplicates)
```

```
Dataset contains duplicate User IDs: False
```

No duplicates were found among the user IDs, affirming the necessity of unique user identifiers to ensure each observation corresponds to a distinct user

[77]:
```
negative_instances = game_data[game_data['sum_gamerounds'] < 0]
print("Dataset contains instances with negative game rounds:", not␣
↪negative_instances.empty)
```

```
Dataset contains instances with negative game rounds: False
```

[79]:
```
has_null = game_data.isna().sum().any()
print("Dataset contains missing values:", has_null)
```

```
Dataset contains missing values: False
```

[6]:
```
percentiles = [0.01, 0.05, 0.1, 0.2, 0.5, 0.8, 0.9, 0.95, 0.99]
summary_stats = game_data['sum_gamerounds'].describe(percentiles=percentiles).
↪transpose()

print("Summary statistics for gamerounds:")
print(summary_stats)
```

```
Summary statistics for gamerounds:
count    90189.000000
mean        51.872457
std        195.050858
min          0.000000
1%           0.000000
5%           1.000000
10%          1.000000
20%          3.000000
50%         16.000000
80%         67.000000
90%        134.000000
95%        221.000000
99%        493.000000
max      49854.000000
Name: sum_gamerounds, dtype: float64
```

The summary statistics reveal that the dataset comprises 90,188 observations, with players averaging approximately 51 rounds. However, there is significant variability, as indicated by the standard deviation of 102.68 rounds. Notably, 99% of players engaged in up to 493 rounds, with a few outliers playing as many as 2961 rounds

[7]:
```
# summary statistics for 'sum_gamerounds' grouped by A/B groups
group_stats = game_data.groupby("version")['sum_gamerounds'].agg(
    count=lambda x: len(x),
    median=lambda x: np.median(x),
    mean=lambda x: np.mean(x),
    std=lambda x: np.std(x),
    max=lambda x: np.max(x)
).reset_index()
```

```python
colors = {'gate_30': 'lightblue', 'gate_40': 'lightcoral',}

plt.figure(figsize=(12, 2))
plt.axis('off')
table = plt.table(cellText=group_stats.values,
                  colLabels=group_stats.columns,
                  cellLoc='center',
                  rowLoc='center',
                  loc='center',
                  cellColours=[[colors.get(version, 'lightgrey')] * len(group_stats.columns) for version in group_stats['version']])

table.auto_set_font_size(False)
table.set_fontsize(10)

plt.title("Summary statistics by A/B groups for 'sum_gamerounds'", fontsize=14)

plt.show()
```

Summary statistics by A/B groups for 'sum_gamerounds'

version	count	median	mean	std	max
gate_30	44700	17.0	52.45626398210291	256.71355155162723	49854
gate_40	45489	16.0	51.29877552814966	103.29328083233604	2640

[8]:
```python
def create_game_rounds_plot(game_data):
    fig = make_subplots(rows=1, cols=3, subplot_titles=("Distribution of Game Rounds (A)",
                                                        "Distribution of Game Rounds (B)",
                                                        "A/B Groups Boxplot"))

    colors = {'gate_30': 'blue', 'gate_40': 'red'}

    # Bar charts for Gate 30 and Gate 40
    for version, subplot in zip(["gate_30", "gate_40"], [1, 2]):
        filtered_data = game_data[game_data.version == version]
        counts, bins = np.histogram(filtered_data.sum_gamerounds, bins=20)
        fig.add_trace(go.Bar(x=bins[:-1], y=counts, marker_color=colors[version], name=version), row=1, col=subplot)

    # Boxplot for A/B groups with the same color as the respective bar chart
    for version in ["gate_30", "gate_40"]:
```

```
        filtered_data = game_data[game_data.version == version]
        fig.add_trace(go.Box(x=[version] * len(filtered_data),
                             y=filtered_data.sum_gamerounds,
                             marker_color=colors[version],
                             boxpoints='outliers',
                             boxmean=False,
                             showlegend=False),
                      row=1, col=3)

    fig.update_xaxes(title_text="Gamerounds", row=1, col=1)
    fig.update_xaxes(title_text="Gamerounds", row=1, col=2)
    fig.update_xaxes(title_text="A/B Groups", row=1, col=3)

    fig.update_yaxes(title_text="Frequency", row=1, col=1, title_standoff=1)
    fig.update_yaxes(title_text="Frequency", row=1, col=2, title_standoff=1)
    fig.update_yaxes(title_text="Sum Gamerounds", row=1, col=3,␣
↪title_standoff=1)

    fig.update_layout(height=500,
                      width=1000,
                      showlegend=True)

    return fig
```

[9]:
```
fig = create_game_rounds_plot(game_data)
fig.show()
```

For a more detailed examination of the distribution, we'll focus on values within the range of 0 to 500 game rounds, as this range contains the majority of our instances

[10]:
```
# Filter the data for values in the range of 0-500 game rounds
filtered_data = game_data[(game_data['sum_gamerounds'] >= 0) &␣
↪(game_data['sum_gamerounds'] <= 500)]

fig = create_game_rounds_plot(filtered_data)
fig.show()
```

[11]:
```
def create_game_rounds_by_version_plot(game_data):
    traces = []
    for version in game_data['version'].unique():
        filtered_data = game_data[game_data['version'] == version]
        trace = go.Scatter(x=filtered_data.index,
                           y=filtered_data.sum_gamerounds,
                           mode='lines',
                           name=version,
                           legendgroup=version)  # legendgroup
        traces.append(trace)
```

```
    layout = go.Layout(title='Game Rounds by Version',
                       xaxis=dict(title='Index'),
                       yaxis=dict(title='Sum Gamerounds'),
                       legend=dict(x=1.1, y=1),
                       updatemenus=[
                           dict(
                               buttons=[
                                   dict(label="All",
                                        method="update",
                                        args=[{"visible": [True] * len(traces)},
                                              {"title": "Game Rounds by Version␣
↪- All"}]),
                                   *[
                                       dict(label=version,
                                            method="update",
                                            args=[{"visible": [trace.visible if␣
↪trace.name == version else False for trace in traces]},
                                                  {"title": f"Game Rounds by␣
↪Version - {version}"}])
                                       for version in game_data['version'].
↪unique()
                                   ]
                               ],
                               direction="down",
                               pad={"r": 10, "t": 10},
                               showactive=True,
                               x=1,
                               xanchor="right",
                               y=1,
                               yanchor="top"
                           ),
                       ]
                       )

    fig = go.Figure(data=traces, layout=layout)
    return fig
```

[12]:
```
fig = create_game_rounds_by_version_plot(game_data)
fig.show()
```

As observed earlier, an extreme outlier was affecting the summary statistics. To address this issue, we will remove the outlier from the dataset

[13]:
```
# Filter the data for game rounds greater than 30,000 for "gate_30" version
filtered_data_gate_30 = game_data[(game_data['version'] == 'gate_30') &␣
 ↪(game_data['sum_gamerounds'] > 30000)]
```

```
print(filtered_data_gate_30)
```

```
        userid  version  sum_gamerounds  retention_1  retention_7
57702  6390605  gate_30           49854        False         True
```

Given that the average number of rounds played by most players is around 51, it seems highly improbable that a player would engage in 50,000 rounds, suggesting a potential data entry error rather than a plausible gaming behavior.

[14]:
```
game_data.drop(filtered_data_gate_30.index, inplace=True)
```

[15]:
```
fig = create_game_rounds_plot(game_data)
fig.show()
```

[16]:
```
fig = create_game_rounds_by_version_plot(game_data)
fig.show()
```

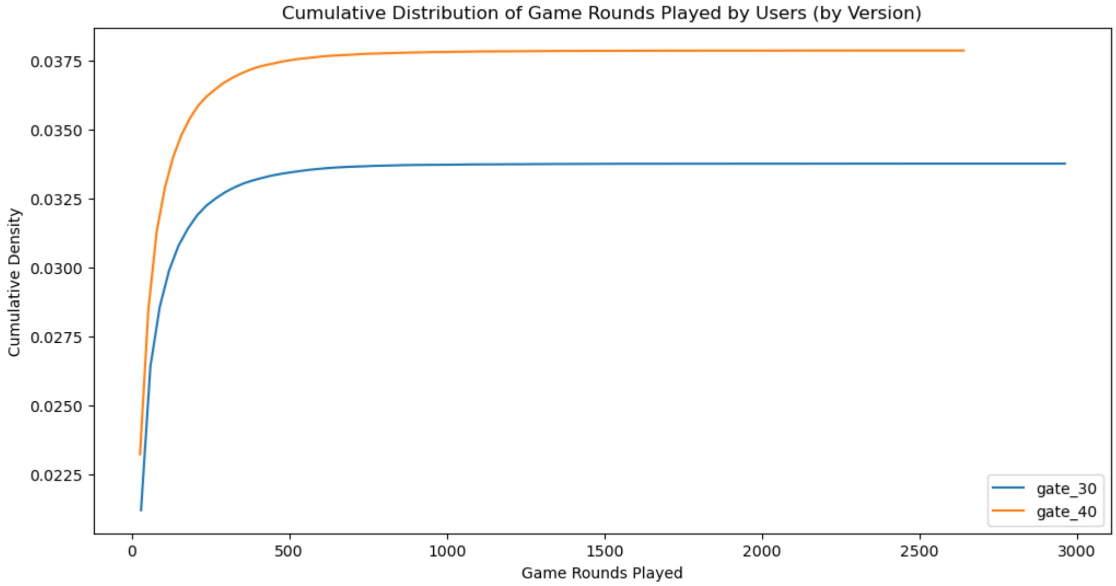
1 Retention

[17]:
```
plt.figure(figsize=(12, 6))
for version in game_data['version'].unique():
    # Filter data for the current version
    version_data = game_data[game_data['version'] == version]

    # Compute cumulative distribution for the current version
    counts, bin_edges = np.histogram(version_data["sum_gamerounds"], bins=100,␣
 ↪density=True)
    cdf = np.cumsum(counts)

    plt.plot(bin_edges[1:], cdf, label=version)

plt.title("Cumulative Distribution of Game Rounds Played by Users (by Version)")
plt.xlabel("Game Rounds Played")
plt.ylabel("Cumulative Density")
plt.legend(loc="lower right")
plt.show()
```



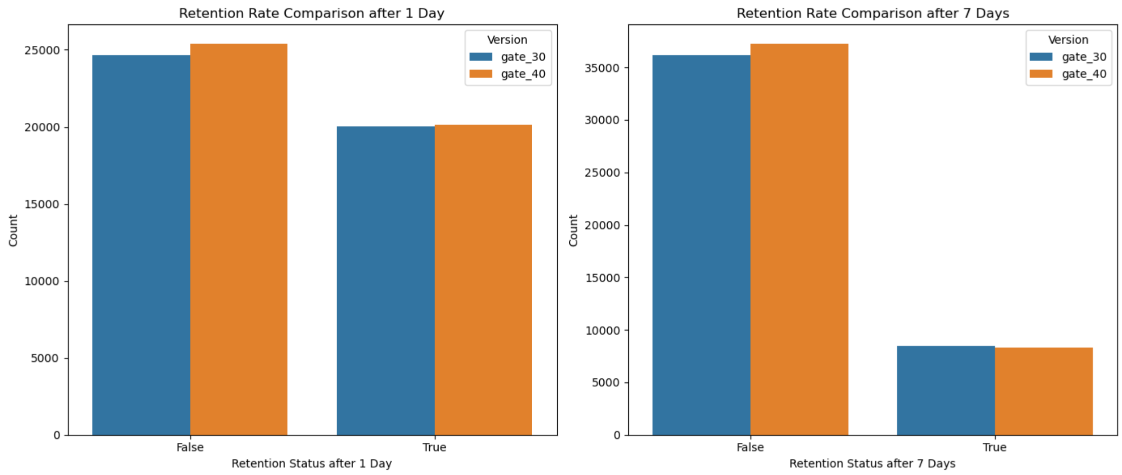
Both gate_30 and gate_40 exhibit a plateau in user engagement after approximately 450-500 game rounds, where gate_40 starts with a higher density of 0.0375 and gate_30 with a slightly lower density of 0.0325. This difference in initial density suggests higher initial user engagement in gate_40. However, both versions maintain a consistent density after the plateau, with gate_40 remaining at 0.0375 and gate_30 at 0.0335, indicating a stabilization in user activity despite variations in initial engagement levels

[61]:
```
fig, axs = plt.subplots(1, 2, figsize=(14, 6))

# Retention Rate Comparison after 1 Day
sns.countplot(x='retention_1', hue='version', data=game_data, ax=axs[0])
axs[0].set_title('Retention Rate Comparison after 1 Day')
axs[0].set_xlabel('Retention Status after 1 Day')
axs[0].set_ylabel('Count')
axs[0].legend(title='Version')

# Retention Rate Comparison after 7 Days
sns.countplot(x='retention_7', hue='version', data=game_data, ax=axs[1])
axs[1].set_title('Retention Rate Comparison after 7 Days')
axs[1].set_xlabel('Retention Status after 7 Days')
axs[1].set_ylabel('Count')
axs[1].legend(title='Version')

plt.tight_layout()
plt.show()
```



[62]:
```
# maximum number of rounds
max_rounds = game_data['sum_gamerounds'].max()

# intervals for game rounds
round_intervals = [50, 100, 150, 200, 250, 300, 350]

fig = go.Figure()

# traces for each interval of game rounds
for round_end in round_intervals:
    filtered_data = game_data[game_data['sum_gamerounds'] <= round_end]
    fig.add_trace(go.Scatter(x=filtered_data.groupby("sum_gamerounds").userid.
 ↪count().index,
                             y=filtered_data.groupby("sum_gamerounds").userid.
 ↪count(),
                             mode='lines',
                             name=f"0-{round_end} Game Rounds",
                             visible=(round_end == round_intervals[-1]),
                             showlegend=False))  # Remove the legend for each␣
 ↪trace

# dropdown menu
buttons = [{'method': 'update',
            'label': 'All',
            'args': [{'visible': [True] * len(round_intervals)},
                     {'title': "The number of users in all game rounds"}]}]  #␣
 ↪Update the title dynamically

for round_end in round_intervals:
    buttons.append(dict(method='update',
```

```
                        label=f"0-{round_end} Game Rounds",
                        args=[{'visible': [(round_end == round_intervals[-1])␣
↪or (x <= round_end) for x in round_intervals]},
                              {'title': f"The number of users in {round_end}␣
↪game rounds"}]))  # Update the title dynamically

fig.update_layout(title="The number of users in all game rounds",  # Initial␣
↪title
                  title_font_size=25,
                  xaxis_title="Game Rounds Played",
                  yaxis_title="Number of Users",
                  updatemenus=[{'buttons': buttons,
                                'direction': 'down',
                                'showactive': True,
                                'x': 0.8,
                                'xanchor': 'left',
                                'y': 1.20,
                                'yanchor': 'top'}],
                  height=400,
                  width=1000)

fig.show()
```

[20]:
```
metrics = {
    "RET1_COUNT": game_data["retention_1"].value_counts(),
    "RET7_COUNT": game_data["retention_7"].value_counts(),
    "RET1_RATIO": game_data["retention_1"].value_counts() / len(game_data),
    "RET7_RATIO": game_data["retention_7"].value_counts() / len(game_data)
}
result_df = pd.DataFrame(metrics)

plt.figure(figsize=(12, 2))
plt.axis('off')
table = plt.table(cellText=result_df.values,
                  colLabels=result_df.columns,
                  rowLabels=result_df.index,
                  cellLoc='center',
                  rowLoc='center',
                  loc='center')

table.auto_set_font_size(False)
table.set_fontsize(10)

plt.title("Retention Rates", fontsize=25, y=0.7)
plt.show()
```

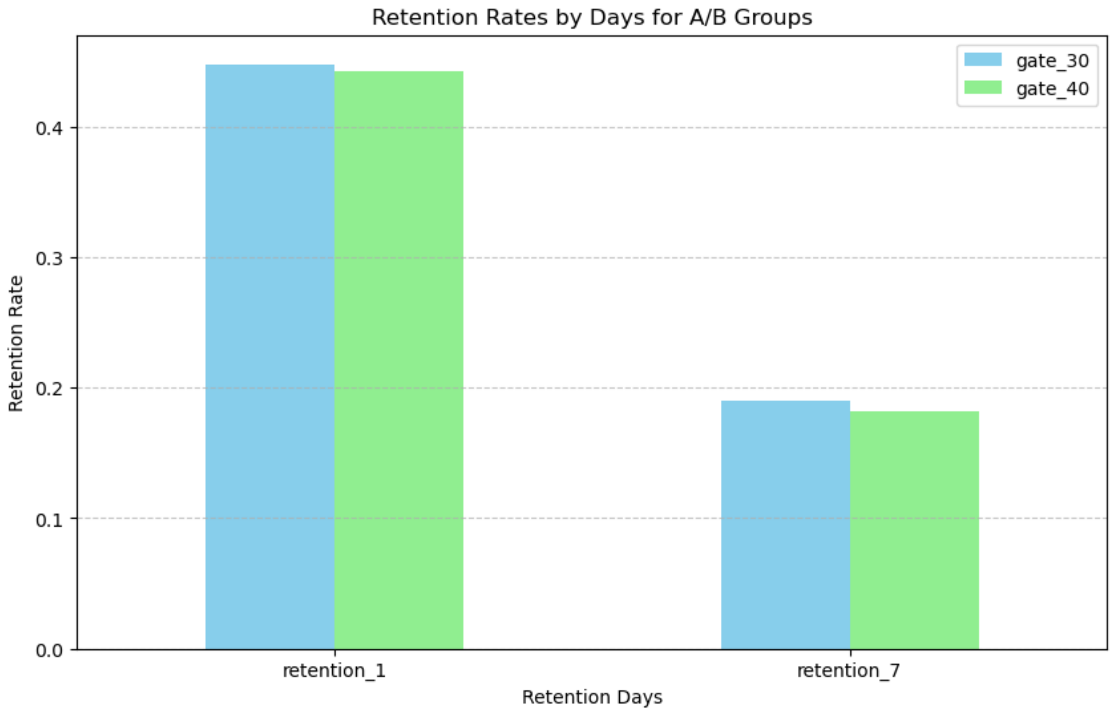
Retention Rates

	RET1_COUNT	RET7_COUNT	RET1_RATIO	RET7_RATIO
False	50035.0	73408.0	0.5547855590544196	0.8139442054375305
True	40153.0	16780.0	0.4452144409455803	0.1860557945624695

The provided data illustrates user retention rates after the first and seventh days of engagement. In particular, it shows that a substantial portion of users did not retain after both time intervals. Specifically, after the first day, approximately 50,035 users, accounting for 55.48% of the total, did not retain, while after the seventh day, this number increased to 73,408 users, representing around 81.39% of the total. Conversely, a smaller number of users successfully retained after both intervals, with 40,153 users (44.52%) and 16,780 users (18.61%) retaining after the first and seventh days, respectively

[21]:
```
retention_by_group = game_data.groupby("version")[["retention_1", "retention_7"]].mean()
retention_by_group_transposed = retention_by_group.T

retention_by_group_transposed.plot(kind='bar', figsize=(10, 6), color=['skyblue', 'lightgreen'])
plt.title("Retention Rates by Days for A/B Groups")
plt.xlabel("Retention Days")
plt.ylabel("Retention Rate")
plt.xticks(rotation=0)
plt.legend(["gate_30", "gate_40"])
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



The plot clearly illustrates that the retention rate for the ‘gate_30’ version is notably higher compared to the ‘gate_40’ version for both retention days 1 and 7

1.1 Retention Week-1

[22]:
```
game_data.groupby(["version", "retention_1"]).sum_gamerounds.agg(["count",␣
↪"median", "mean", "std", "max"])
```

[22]:

version	retention_1	count	median	mean	std	max
gate_30	False	24665	6.0	16.359092	36.528426	1072
	True	20034	48.0	94.411700	135.037697	2961
gate_40	False	25370	6.0	16.340402	35.925756	1241
	True	20119	49.0	95.381182	137.887256	2640

Across both versions, users who retained after the specified period exhibited notably higher median and mean values of game rounds played compared to those who did not retain. For instance, in ‘gate_30’, users retained after the period demonstrated a median of 48 game rounds and a mean of approximately 94.41, significantly exceeding the median of 6 and mean of approximately 16.36 observed for non-retained users. Similarly, in ‘gate_40’, retained users displayed a median of 49 game rounds and a mean of approximately 95.38, contrasting with non-retained users’ median of 6 and mean of approximately 16.34.

1.2 Retention Week-7

[23]:
```python
game_data.groupby(["version", "retention_7"]).sum_gamerounds.agg(["count",
    "median", "mean", "std", "max"])
```

[23]:

version	retention_7	count	median	mean	std	max
gate_30	False	36198	11.0	25.796508	43.316158	981
	True	8501	105.0	160.117516	179.358560	2961
gate_40	False	37210	11.0	25.856356	44.406112	2640
	True	8279	111.0	165.649837	183.792499	2294

In the 'gate_30' version, retained users displayed a median of 105 game rounds and a mean of approximately 160.12, in contrast to non-retained users, who had a median of 11 game rounds and a mean of approximately 25.80. Similarly, in the 'gate_40' version, retained users showed a median of 111 game rounds and a mean of approximately 165.65, while non-retained users had a median of 11 game rounds and a mean of approximately 25.86.

2 A/B Testing

[60]:
```python
def perform_ab_test(data, group_col, target_col, alpha=0.05, effect_size=None,
    power=None):
    # Splitting A/B groups
    group_a = data[data[group_col] == "gate_30"][target_col]
    group_b = data[data[group_col] == "gate_40"][target_col]

    # Checking normality assumption using Shapiro-Wilk test
    normality_a = shapiro(group_a)[1] >= alpha
    normality_b = shapiro(group_b)[1] >= alpha

    # Perform t-test or Mann-Whitney U test based on normality
    if normality_a and normality_b:
        # Parametric test (t-test)
        test_stat, p_value = ttest_ind(group_a, group_b)
        test_type = "Parametric (t-test)"
    else:
        # Non-parametric test (Mann-Whitney U test)
        test_stat, p_value = mannwhitneyu(group_a, group_b)
        test_type = "Non-parametric (Mann-Whitney U)"

    # Calculate effect size (Cohen's d)
    if effect_size is None:
        if normality_a and normality_b:
            pooled_std = np.sqrt((np.var(group_a) + np.var(group_b)) / 2)
            effect_size = (group_a.mean() - group_b.mean()) / pooled_std
        else:
```

```
            # Use rank-biserial correlation as effect size for non-parametric␣
↪test
            effect_size = 2 * test_stat / len(data)

    # Interpretation of results
    hypothesis = "Reject H0" if p_value < alpha else "Fail to Reject H0"
    comment = "A/B groups are not similar!" if hypothesis == "Reject H0" else␣
↪"A/B groups are similar!"

    result_df = pd.DataFrame({
        "Test Type": [test_type],
        "AB Hypothesis": [hypothesis],
        "p-value": [p_value],
        "Effect Size (Cohen's d)": [effect_size],
        "Comment": [comment]
    })

    plt.figure(figsize=(12, 1))
    plt.axis('off')
    table = plt.table(cellText=result_df.values,
                      colLabels=result_df.columns,
                      cellLoc='center',
                      rowLoc='center',
                      loc='center')

    table.auto_set_font_size(False)
    table.set_fontsize(7)

    plt.title("A/B Test Results", fontsize=25, y=0.7)
    plt.show()

    return ""

perform_ab_test(data=game_data, group_col="version",␣
↪target_col="sum_gamerounds")
```

A/B Test Results

Test Type	AB Hypothesis	p-value	Effect Size (Cohen's d)	Comment
Non-parametric (Mann-Whitney U)	Fail to Reject H0	0.05089155279145376	22714.45783252761	A/B groups are similar!

[60]: ''

In the following analysis, we will conduct A/B testing to compare the performance of two groups: Group A, consisting of players who were assigned the gate_30 version and retained in the game, and Group B, comprising players assigned the gate_40 version who were also retained. The objective is to determine if there are any significant differences between these two versions in terms of player retention.

Additionally, we will explore the differences between the two groups concerning non-retained players. This comparison will help us understand if the gate_30 and gate_40 versions have distinct effects on player retention among those who did not continue playing the game.

By analyzing both retained and non-retained players in each group, we aim to gain insights into the effectiveness of the different game versions in retaining players over time.

[45]:
```
def perform_ab_test(data, group_col, target_col, retention_day, alpha=0.05,␣
↪effect_size=None, power=None):
    # Splitting A/B groups
    group_a = data[(data[group_col] == "gate_30") & (data["retention_" +␣
↪str(retention_day)] == True)][target_col]
    group_b = data[(data[group_col] == "gate_40") & (data["retention_" +␣
↪str(retention_day)] == True)][target_col]

    # Checking normality assumption using Shapiro-Wilk test
    normality_a = shapiro(group_a)[1] >= alpha
    normality_b = shapiro(group_b)[1] >= alpha

    # Perform t-test or Mann-Whitney U test based on normality
    if normality_a and normality_b:
        # Parametric test (t-test)
        test_stat, p_value = ttest_ind(group_a, group_b)
        test_type = "Parametric (t-test)"
    else:
        # Non-parametric test (Mann-Whitney U test)
        test_stat, p_value = mannwhitneyu(group_a, group_b)
        test_type = "Non-parametric (Mann-Whitney U)"

    # Calculate effect size (Cohen's d)
    if effect_size is None:
        if normality_a and normality_b:
            pooled_std = np.sqrt((np.var(group_a) + np.var(group_b)) / 2)
            effect_size = (group_a.mean() - group_b.mean()) / pooled_std
        else:
            # Use rank-biserial correlation as effect size for non-parametric␣
↪test
            effect_size = 2 * test_stat / len(data)

    # Interpretation of results
    hypothesis = "Reject H0" if p_value < alpha else "Fail to Reject H0"
    comment = "A/B groups are not similar!" if hypothesis == "Reject H0" else␣
↪"A/B groups are similar!"

    result_df = pd.DataFrame({
```

```
        "Test Type": [test_type],
        "AB Hypothesis": [hypothesis],
        "p-value": [p_value],
        "Effect Size (Cohen's d)": [effect_size],
        "Comment": [comment]
    })

    plt.figure(figsize=(12, 1))
    plt.axis('off')
    table = plt.table(cellText=result_df.values,
                      colLabels=result_df.columns,
                      cellLoc='center',
                      rowLoc='center',
                      loc='center')

    table.auto_set_font_size(False)
    table.set_fontsize(7)

    plt.title("A/B Test Results for Retention " + str(retention_day), fontsize=25, y=0.7)
    plt.show()

    return " "

perform_ab_test(data=game_data, group_col="version", target_col="sum_gamerounds", retention_day=1)
perform_ab_test(data=game_data, group_col="version", target_col="sum_gamerounds", retention_day=7)
```

A/B Test Results for Retention 1

Test Type	AB Hypothesis	p-value	Effect Size (Cohen's d)	Comment
Non-parametric (Mann-Whitney U)	Fail to Reject H0	0.7095456166954823	4459.561704439615	A/B groups are similar!

A/B Test Results for Retention 7

Test Type	AB Hypothesis	p-value	Effect Size (Cohen's d)	Comment
Non-parametric (Mann-Whitney U)	Reject H0	0.010082490209441287	762.467224020934	A/B groups are not similar!

[45]: ' '

While the retention_1 groups showed similarities, indicating consistent user behavior or experience between the A and B groups initially, the retention_7 groups exhibited significant differences. This implies that over the weeks following the initial observation, disparities in user engagement or experience emerged between the two groups, suggesting that any initial impact introduced by the experimental variations did not persist uniformly over time

[47]:
```
def perform_ab_test(data, group_col, target_col, retention_day, alpha=0.05,␣
↪effect_size=None, power=None):
    # Splitting A/B groups
    group_a = data[(data[group_col] == "gate_30") & (data["retention_" +␣
↪str(retention_day)] == False)][target_col]
    group_b = data[(data[group_col] == "gate_40") & (data["retention_" +␣
↪str(retention_day)] == False)][target_col]

    # Checking normality assumption using Shapiro-Wilk test
    normality_a = shapiro(group_a)[1] >= alpha
    normality_b = shapiro(group_b)[1] >= alpha

    # Perform t-test or Mann-Whitney U test based on normality
    if normality_a and normality_b:
        # Parametric test (t-test)
        test_stat, p_value = ttest_ind(group_a, group_b)
        test_type = "Parametric (t-test)"
    else:
        # Non-parametric test (Mann-Whitney U test)
        test_stat, p_value = mannwhitneyu(group_a, group_b)
        test_type = "Non-parametric (Mann-Whitney U)"

    # Calculate effect size (Cohen's d)
    if effect_size is None:
        if normality_a and normality_b:
            pooled_std = np.sqrt((np.var(group_a) + np.var(group_b)) / 2)
            effect_size = (group_a.mean() - group_b.mean()) / pooled_std
        else:
            # Use rank-biserial correlation as effect size for non-parametric␣
↪test
            effect_size = 2 * test_stat / len(data)

    # Interpretation of results
    hypothesis = "Reject H0" if p_value < alpha else "Fail to Reject H0"
    comment = "A/B groups are not similar!" if hypothesis == "Reject H0" else␣
↪"A/B groups are similar!"

    result_df = pd.DataFrame({
        "Test Type": [test_type],
        "AB Hypothesis": [hypothesis],
        "p-value": [p_value],
        "Effect Size (Cohen's d)": [effect_size],
        "Comment": [comment]
    })
```

```
    plt.figure(figsize=(12, 1))
    plt.axis('off')
    table = plt.table(cellText=result_df.values,
                      colLabels=result_df.columns,
                      cellLoc='center',
                      rowLoc='center',
                      loc='center')

    table.auto_set_font_size(False)
    table.set_fontsize(7)

    plt.title("A/B Test Results for Retention " + str(retention_day),␣
 ↪fontsize=25, y=0.7)
    plt.show()

    return " "

perform_ab_test(data=game_data, group_col="version",␣
 ↪target_col="sum_gamerounds", retention_day=1)
perform_ab_test(data=game_data, group_col="version",␣
 ↪target_col="sum_gamerounds", retention_day=7)
```

A/B Test Results for Retention 1

Test Type	AB Hypothesis	p-value	Effect Size (Cohen's d)	Comment
Non-parametric (Mann-Whitney U)	Fail to Reject H0	0.17902479366766655	6986.347119350689	A/B groups are similar!

A/B Test Results for Retention 7

Test Type	AB Hypothesis	p-value	Effect Size (Cohen's d)	Comment
Non-parametric (Mann-Whitney U)	Fail to Reject H0	0.2919906205799716	15001.69772031756	A/B groups are similar!

```
[47]: ' '
```