weekly-users-trend

June 22, 2024

1 Importing Necessary Libraries

[51]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

2 Reading Data

[52]:
```
df = pd.read_excel('data.xls')
```

[53]:
```
df.head()
```

[53]:
```
                 w1                w2                w3                w4  \
0  fd7c28f9fd8045f2  fd7c28f9fd8045f2  fd7c28f9fd8045f2  fd7c28f9fd8045f2
1  54910d2b363221e1  520443b0b8128202  a4bce0d054266d68  a4bce0d054266d68
2  520443b0b8128202  a4bce0d054266d68  7b042fcc54a45882  d98da6eaa4bb452f
3  a4bce0d054266d68  d1afc6d7c4661d7e  aed9597fc6984d64  7b042fcc54a45882
4  3792a1c9395e3e2a  7b042fcc54a45882  407d67f50877e6f9  aed9597fc6984d64

                 w5                w6                w7                w8  \
0  fd7c28f9fd8045f2  231d91be38352d7a  306243851b716bf6  149f7dd1efe25ebc
1  c0bb01dbe2b2de0f  53010d4139ed029f  d1afc6d7c4661d7e  9ab44ee389767d59
2  7b042fcc54a45882  a4bce0d054266d68  3792a1c9395e3e2a  839d5042ee4d8988
3  c885df69f0e13074  d1afc6d7c4661d7e  7b042fcc54a45882  a4bce0d054266d68
4  aed9597fc6984d64  3792a1c9395e3e2a  a455b3d89d7d6a3b  306243851b716bf6

                 w9               w10  …               w47  \
0  191a909000d7123d  191a909000d7123d  …   ffedb2a5b3b4838
1  91e804eb002a580d  5109246885c54360  …  ffe3d17a83edd05a
2  a4bce0d054266d68  e254fb2201bf1419  …  ffde16048235a32f
3  306243851b716bf6  306243851b716bf6  …  ffb36133fb3c44e2
4  a99a477e2c336bb9  a99a477e2c336bb9  …  ff7e5bf81a779007
```

```
                 w48               w49               w50               w51  \
0   ffedb2a5b3b4838  fff444dcd8f9808f   ffedb2a5b3b4838   ffedb2a5b3b4838
1  ffe3d17a83edd05a   ffedb2a5b3b4838  ffde16048235a32f  ffde16048235a32f
2  ffde16048235a32f  ffde16048235a32f  ffd99d6c632283a9  ffc6c128db97ab1d
3  ffdce5869723d832   ffbafb7cc49be72  ffb36133fb3c44e2  ffb36133fb3c44e2
4  ffb36133fb3c44e2  ffb36133fb3c44e2  ffa96fa38b711342  ffa96fa38b711342

                 w52               w53               w54               w55  \
0   ffedb2a5b3b4838   ffedb2a5b3b4838  fffe76c3a948cdfb  fffe76c3a948cdfb
1  ffe7939306264854  ffe7939306264854  fff444dcd8f9808f  fff444dcd8f9808f
2  ffe3d17a83edd05a  ffb36133fb3c44e2   ffedb2a5b3b4838   ffedb2a5b3b4838
3  ffde16048235a32f  ffa8eb6c18e09543  ffe7939306264854  ffde16048235a32f
4  ffb36133fb3c44e2  ff7e5bf81a779007  ffde16048235a32f  ffb36133fb3c44e2

                 w56
0  fffe76c3a948cdfb
1  ffe7939306264854
2  ffde16048235a32f
3  ffb94deefa8aa79f
4  ffb36133fb3c44e2

[5 rows x 56 columns]
```

3 Data Exploration

[54]:

```
df.describe(include = 'O')
```

[54]:

```
                      w1                w2                w3  \
count               1759              1654              1732
unique              1759              1654              1732
top     fd7c28f9fd8045f2  fd7c28f9fd8045f2  fd7c28f9fd8045f2
freq                   1                 1                 1

                      w4                w5                w6  \
count               2116              2193              2157
unique              2116              2193              2155
top     fd7c28f9fd8045f2  fd7c28f9fd8045f2  231d91be38352d7a
freq                   1                 1                 2

                      w7                w8                w9  \
count               2551              2875              2795
unique              2549              2875              2795
top     d1afc6d7c4661d7e  149f7dd1efe25ebc  191a909000d7123d
freq                   2                 1                 1

                     w10  …              w47               w48  \
```

```
count                 2821  …              3607              3822
unique                2821  …              3607              3822
top       191a909000d7123d  …  ffedb2a5b3b4838  ffedb2a5b3b4838
freq                     1  …                 1                 1

                       w49              w50              w51              w52  \
count                 3863             3754             3801             3768
unique                3863             3754             3801             3768
top       fff444dcd8f9808f  ffedb2a5b3b4838  ffedb2a5b3b4838  ffedb2a5b3b4838
freq                     1                1                1                1

                      w53               w54               w55               w56
count                3741              3909              3806              3696
unique               3741              3909              3806              3696
top       ffedb2a5b3b4838  fffe76c3a948cdfb  fffe76c3a948cdfb  fffe76c3a948cdfb
freq                    1                 1                 1                 1

[4 rows x 56 columns]
```

[55]:
```
df.dtypes
```

[55]:
```
w1      object
w2      object
w3      object
w4      object
w5      object
w6      object
w7      object
w8      object
w9      object
w10     object
w11     object
w12     object
w13     object
w14     object
w15     object
w16     object
w17     object
w18     object
w19     object
w20     object
w21     object
w22     object
w23     object
w24     object
w25     object
w26     object
```

```
w27      object
w28      object
w29      object
w30      object
w31      object
w32      object
w33      object
w34      object
w35      object
w36      object
w37      object
w38      object
w39      object
w40      object
w41      object
w42      object
w43      object
w44      object
w45      object
w46      object
w47      object
w48      object
w49      object
w50      object
w51      object
w52      object
w53      object
w54      object
w55      object
w56      object
dtype: object
```

4 Data Preprocessing

[56]:
```python
def remove_duplicates_inplace(df):
    for col in df.columns:
        df[col] = df[col].drop_duplicates(inplace=False)  # Keep original order

# Apply the function to remove duplicates in-place
remove_duplicates_inplace(df)

df.describe(include = 'O')
```

[56]:
```
                         w1                  w2                  w3  \
count                  1759                1654                1732
unique                 1759                1654                1732
```

```
top     fd7c28f9fd8045f2  fd7c28f9fd8045f2  fd7c28f9fd8045f2
freq                   1                 1                 1

                      w4                w5                w6  \
count               2116              2193              2155
unique              2116              2193              2155
top     fd7c28f9fd8045f2  fd7c28f9fd8045f2  231d91be38352d7a
freq                   1                 1                 1

                      w7                w8                w9  \
count               2549              2875              2795
unique              2549              2875              2795
top     306243851b716bf6  149f7dd1efe25ebc  191a909000d7123d
freq                   1                 1                 1

                     w10  …               w47               w48  \
count               2821  …              3607              3822
unique              2821  …              3607              3822
top     191a909000d7123d  …  ffedb2a5b3b4838   ffedb2a5b3b4838
freq                   1  …                 1                 1

                     w49              w50              w51              w52  \
count               3863             3754             3801             3768
unique              3863             3754             3801             3768
top     fff444dcd8f9808f  ffedb2a5b3b4838  ffedb2a5b3b4838  ffedb2a5b3b4838
freq                   1                1                1                1

                    w53               w54               w55               w56
count              3741              3909              3806              3696
unique             3741              3909              3806              3696
top     ffedb2a5b3b4838  fffe76c3a948cdfb  fffe76c3a948cdfb  fffe76c3a948cdfb
freq                  1                 1                 1                 1

[4 rows x 56 columns]
```

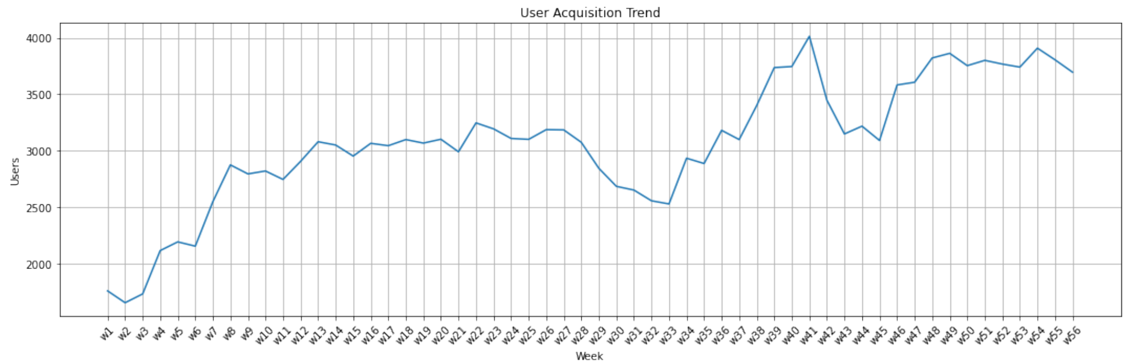
Some weeks have duplicate IDs (freq>1). We will remove them in Pre-processing part.

5 Data Visualization

[57]:
```
df_describe = df.describe(include = 'O')
user_count = df_describe.loc['count',:]
df_user_count = user_count.reset_index()
df_user_count = df_user_count.set_index('index')
```

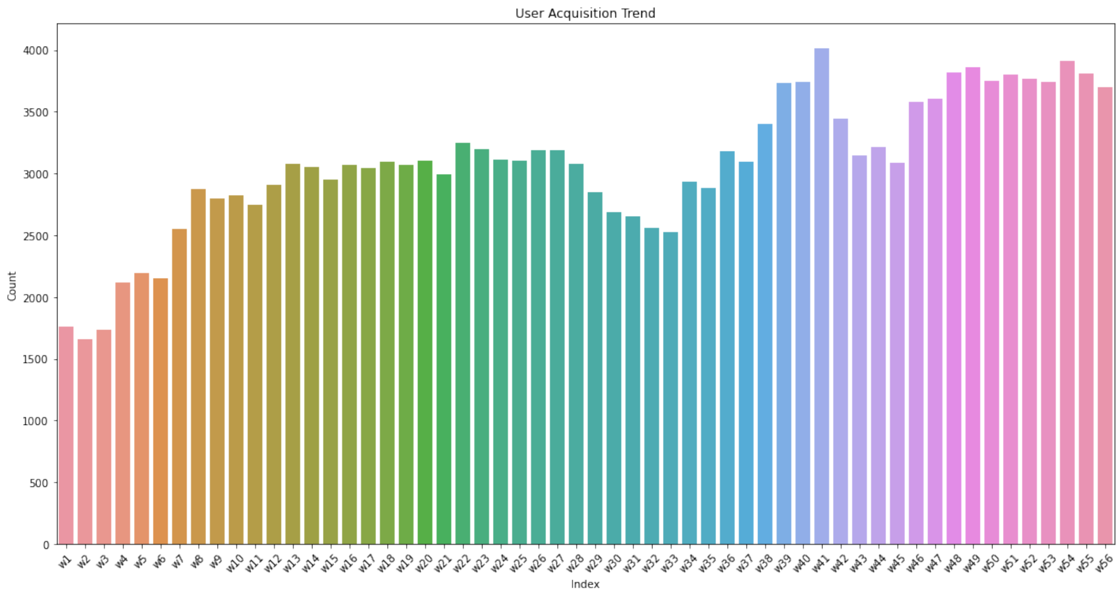
[58]:
```
# Create the line plot
plt.figure(figsize=(15, 5))
plt.plot(df_user_count['count'])
```

```
plt.xlabel('Week')
plt.ylabel('Users')
plt.title('User Acquisition Trend')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()  # Adjust layout to prevent overlapping elements (optional)
plt.show()
```



[59]:

```
# Create the count plot using sns.countplot
plt.figure(figsize=(15, 8))
sns.barplot(x=df_user_count.index, y=df_user_count['count'])
plt.xlabel('Index')
plt.ylabel('Count')
plt.title('User Acquisition Trend')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Interpretation: User Acquisition: * Strong Initial Growth: We observed a rapid rise in weekly active users (WAU) at the beginning, indicating a successful launch and user acquisition strategy.

- Sustained Engagement: This growth stabilized around week 27, suggesting a period of consistent user engagement. Mid-Phase Challenges: A decline in WAU emerged during the middle phase, potentially due to factors requiring investigation (e.g., changes in marketing strategy, product updates).
- Impressive User Growth: The user base then rebounded with a high weekly user growth rate, peaking at week 41. This resurgence indicates successful efforts to re-engage users or attract new ones.
- Retention Focus: However, a noticeable drop in user growth rate followed shortly after, suggesting a need to focus on retention strategies to maintain the user base.
- Gradual Recovery and Stability: WAU gradually picked up again from week 46 and remained stable until the end of the analyzed period. This final trend suggests a potential return to user acquisition efforts or successful retention strategies.

6 Creating and Analyzing Metrics

[60]:
```
import pandas as pd

def remove_duplicates_inplace(df):
  for col in df.columns:
    df[col] = df[col].drop_duplicates(inplace=False)  # Keep original order

# Apply the function to remove duplicates in-place
```

```
remove_duplicates_inplace(df)

# Initialize the new DataFrame
weeks = df.columns.tolist()
new_df = pd.DataFrame(index=weeks, columns=['total_users', 'new_users',␣
 ↪'churn_users', 'resurrected_users', 'retained_users'])

# Initialize sets to keep track of user statuses
cumulative_users = set()
previous_week_users = set()

# Iterate over each week to calculate the required features
for week in weeks:
  current_week_users = set(df[week].dropna())

  # Calculate new users
  new_users = current_week_users - cumulative_users

  # Calculate churn users
  churn_users = previous_week_users - current_week_users

  # Calculate resurrected users
  resurrected_users = current_week_users - previous_week_users - new_users

  # Calculate retained users (users from previous week who are still active)
  retained_users = previous_week_users & current_week_users  # Intersection of␣
 ↪sets

  # Update cumulative users
  cumulative_users.update(current_week_users)

  # Update the new dataframe with calculated values
  new_df.at[week, 'total_users'] = len(cumulative_users)
  new_df.at[week, 'current_week_users'] = len(current_week_users)
  new_df.at[week, 'new_users'] = len(new_users)
  new_df.at[week, 'churn_users'] = len(churn_users)
  new_df.at[week, 'resurrected_users'] = len(resurrected_users)
  new_df.at[week, 'retained_users'] = len(retained_users)

  # Update previous week users
  previous_week_users = current_week_users

new_df.head()
```

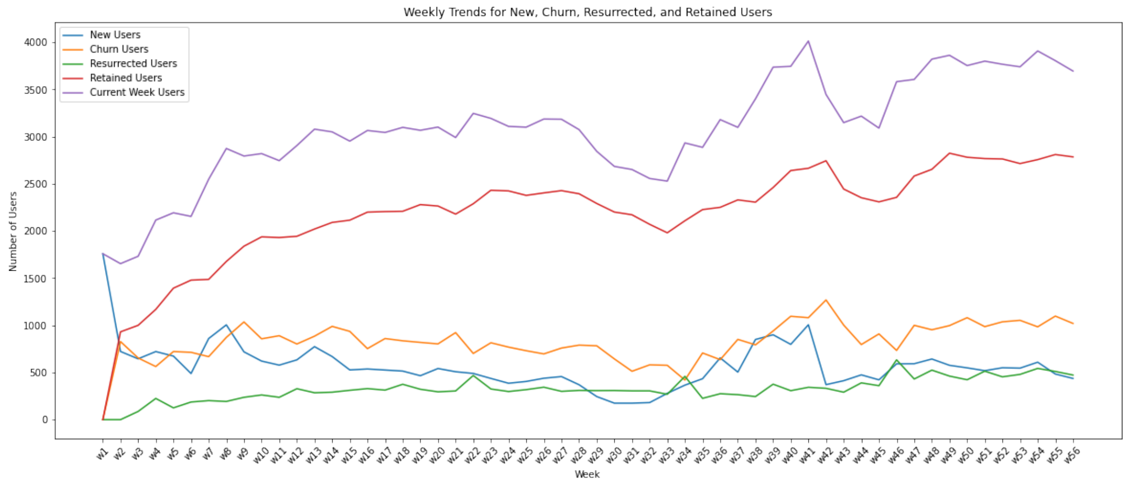
[60]:

```
   total_users new_users churn_users resurrected_users retained_users  \
w1        1759      1759           0                 0              0
w2        2482       723         828                 0            931
```

```
w3        3127       645        654                 87          1000
w4        3849       722        563                225          1169
w5        4523       674        722                125          1394

    current_week_users
w1              1759.0
w2              1654.0
w3              1732.0
w4              2116.0
w5              2193.0
```

[61]:

```
# Drop 'total_users' column for plotting
plot_df = new_df.drop(columns=['total_users'])

# Reset index to use week numbers as a column for plotting
plot_df.reset_index(inplace=True)
plot_df.rename(columns={'index': 'week'}, inplace=True)



plt.figure(figsize=(20, 8))
sns.lineplot(data=plot_df, x='week', y='new_users', label='New Users')
sns.lineplot(data=plot_df, x='week', y='churn_users', label='Churn Users')
sns.lineplot(data=plot_df, x='week', y='resurrected_users', label='Resurrected␣
 ↪Users')
sns.lineplot(data=plot_df, x='week', y='retained_users', label='Retained Users')
sns.lineplot(data=plot_df, x='week', y='current_week_users', label='Current␣
 ↪Week Users')

# Set plot labels and title
plt.xlabel('Week')
plt.ylabel('Number of Users')
plt.title('Weekly Trends for New, Churn, Resurrected, and Retained Users')
plt.xticks(rotation=45)

# Add a legend
plt.legend()

plt.show()
```



Overall Trends:

1) New Users (Blue Line):

- Initially high, indicating a significant influx of new users.
- Experiences fluctuations with noticeable peaks and troughs, indicating varying success in attracting new users week over week.
- There is a stabletrend over time, suggesting a stability in the rate of new user acquisition.

2) Churn Users (Orange Line):

- Begins at a moderate level and remains largely stable over time.
- Occasional peaks indicate periods where user attrition increases, possibly due to dissatisfaction or unmet expectations.
- Occasional dips indicate the effectiveness of new strategies aimed at re-engaging inactive users and preventing further attrition.

3) Resurrected Users (Green Line):

- Starts low and has occasional spikes, indicating successful re-engagement efforts during those periods.
- The overall trend is upward, showing that some users are returning after inactivity.
- Consistency in these spikes could stabilize the user base if maintained or improved.

4) Retained Users (Red Line):

- Start low but immediately picks up the pace afterwards.
- Indicates a Effective retention strategy that keeps a core group of users active.
- Occasional dips suggest periods where retention efforts were less effective.

5) Current Week Users (Purple Line):

- Highest line, showing a cumulative effect of all user activities (new, churned, resurrected, retained).

- Steadily increases over time, indicating overall growth in the active user base despite fluctuations in other metrics.
- A strong indicator of the startup's overall growth trajectory.

6.0.1 Weekly Active Users (WAU)

Understanding the nature of the Growth Consider the following two accounting identities.

WAU(t) = new(t) + retained(t) + resurrected(t)

WAU(t – 1 month) = retained(t) + churned(t)

- The first one says that active users today (for the trailing 7 days) are either new users, retained from the previous week or resurrected from some prior period. (Note that this is a mutually exclusive and completely exhaustive classification of current users.)

- The second identity says that the MAU from last month either came back and were retained or did not and thus churned.

Putting them together:

- WAU(t) – WAU(t – 1 month) = new(t) + resurrected(t) – churned(t)

Which means that WAU growth receives

- Positive contributions from new and resurrected users
- Negative contribution from losing users to churn.
- Here's an even better way to look at the WAU growth accounting quantities for the above company.

6.0.2 What influences growth?

Some Ratios that help us understand this even better.

1) Retention rate

- Retention Rate measures the percentage of users who continue to use the app in the current week out of the users who were active in the previous week.

Retention Rate = (Previous Week's Users / Retained Users) × 100

2) User Replacement Ratio.

- This User Replacement Ratio needs to be greater than one if the app is to be growing, otherwise churn is overwhelming growth. We call this the "quick ratio".

User Replacement Ratio = (New Users + Resurrected Users) / Churned Users

- The User Replacement Ratio thus indicates how effectively the new and resurrected users replace the churned users. A ratio greater than 1 implies that the combined number of new and resurrected users exceeds the number of churned users, indicating positive user growth or retention dynamics.

3) Churn Rate

- Churn Rate measures the percentage of users who stop using the app in the current week out of the users who were active in the previous week.

Churn Rate = (Churned Users / Previous Week's Users) × 100

4) Resurrection Rate

- Resurrection Rate measures the percentage of previously inactive users who return to the app in the current week out of the users who were inactive in the previous week.

Resurrection Rate = (Resurrected Users / Previous Week's Users) × 100

5) User Growth Rate

- User Growth Rate measures the percentage increase or decrease in the number of active users compared to the previous week.

User Growth Rate = (Net Change in Users / Previous Week's Users) × 100

[62]:
```python
# Initialize new columns with NaN or appropriate starting values
new_df['net_change'] = pd.NA
new_df['retention_rate'] = pd.NA
new_df['user_growth_rate'] = pd.NA
new_df['churn_rate'] = pd.NA
new_df['resurrection_rate'] = pd.NA
new_df['user_replacement_ratio'] = pd.NA
# new_df['cumulative_new_users'] = pd.NA
# new_df['cumulative_churned_users'] = pd.NA

cumulative_new_users = 0
cumulative_churned_users = 0
previous_week_users = 0

for week in new_df.index:
#     current_total_users = new_df.at[week, 'total_users']
    current_week_users = new_df.at[week, 'current_week_users']
    new_users = new_df.at[week, 'new_users']
    churn_users = new_df.at[week, 'churn_users']
    resurrected_users = new_df.at[week, 'resurrected_users']
    retained_users = new_df.at[week, 'retained_users']

    # Calculate net change
    net_change = new_users + resurrected_users - churn_users
    new_df.at[week, 'net_change'] = net_change

    # Calculate user growth rate
    if previous_week_users > 0:
        retention_rate = (retained_users / previous_week_users) * 100
        user_growth_rate = (net_change / previous_week_users) * 100
        churn_rate = (churn_users / previous_week_users) * 100
        resurrection_rate = (resurrected_users / previous_week_users) * 100

        if churn_users > 0:
```

```
            user_replacement_ratio = (new_users + resurrected_users) /␣
↪churn_users
        else:
            user_replacement_ratio = 0

    else:
        retention_rate = user_growth_rate = churn_rate = resurrection_rate =␣
↪user_replacement_ratio = 0


    new_df.at[week, 'retention_rate'] = retention_rate
    new_df.at[week, 'user_growth_rate'] = user_growth_rate
    new_df.at[week, 'churn_rate'] = churn_rate
    new_df.at[week, 'resurrection_rate'] = resurrection_rate
    new_df.at[week, 'user_replacement_ratio'] = user_replacement_ratio


    # Calculate cumulative new users and churned users
#     cumulative_new_users += new_users
#     cumulative_churned_users += churn_users

#     new_df.at[week, 'cumulative_new_users'] = cumulative_new_users
#     new_df.at[week, 'cumulative_churned_users'] = cumulative_churned_users

    # Update previous week users
    previous_week_users = current_week_users

# Display the updated dataframe
new_df.head()
```

[62]:

```
   total_users new_users churn_users resurrected_users retained_users  \
w1        1759      1759           0                 0              0
w2        2482       723         828                 0            931
w3        3127       645         654                87           1000
w4        3849       722         563               225           1169
w5        4523       674         722               125           1394

    current_week_users net_change retention_rate user_growth_rate churn_rate  \
w1              1759.0       1759              0                0          0
w2              1654.0       -105        52.9278        -5.969301    47.0722
w3              1732.0         78      60.459492          4.71584  39.540508
w4              2116.0        384      67.494226        22.170901  32.505774
w5              2193.0         77      65.879017         3.638941  34.120983

   resurrection_rate user_replacement_ratio
w1                 0                      0
w2               0.0               0.873188
```

```
w3          5.259976                 1.119266
w4         12.990762                  1.68206
w5          5.907372                 1.106648
```

[63]:
```python
# Changing new_df columns' datatype from object to float/int.

to_int = ['total_users', 'new_users', 'churn_users', 'resurrected_users',␣
 ↪'retained_users', 'current_week_users', 'net_change']
to_float = ['retention_rate', 'user_growth_rate', 'churn_rate',␣
 ↪'resurrection_rate', 'user_replacement_ratio']

new_df[to_int] = new_df[to_int].astype(int)
new_df[to_float] = new_df[to_float].astype(float)

print(new_df.dtypes)
```

```
total_users                 int32
new_users                   int32
churn_users                 int32
resurrected_users           int32
retained_users              int32
current_week_users          int32
net_change                  int32
retention_rate            float64
user_growth_rate          float64
churn_rate                float64
resurrection_rate         float64
user_replacement_ratio    float64
dtype: object
```

[64]:
```python
new_df.describe()[to_float]
```

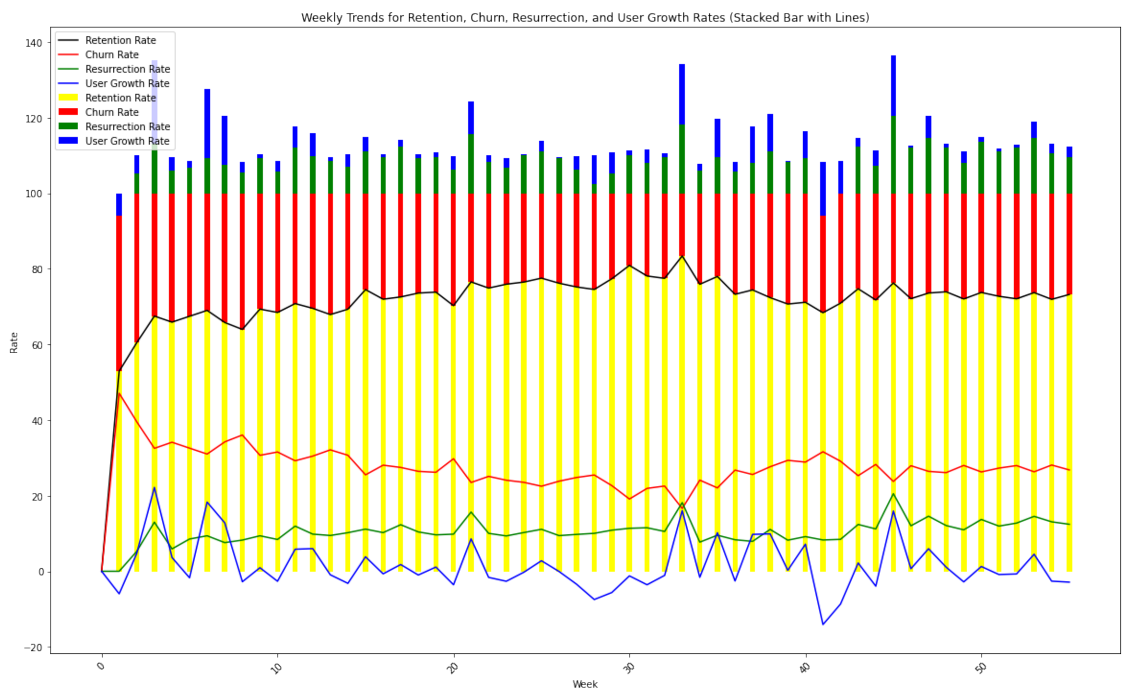
[64]:
```
       retention_rate  user_growth_rate  churn_rate  resurrection_rate  \
count       56.000000         56.000000   56.000000          56.000000
mean        71.001504          1.547984   27.212782          10.340585
std         10.813287          6.754694    6.108299           3.314817
min          0.000000        -14.100648    0.000000           0.000000
25%         69.493851         -2.659284   24.627025           8.999108
50%         72.631431         -0.160036   27.041042          10.201060
75%         74.973989          4.547043   29.929806          11.965742
max         83.392645         22.170901   47.072200          20.511161

       user_replacement_ratio
count               56.000000
mean                 1.044973
std                  0.294536
min                  0.000000
```

```
25%                 0.902524
50%                 0.982319
75%                 1.155233
max                 1.964286
```

6.0.3 Weekly Trends for Retention, Churn, and Resurrection Rates

[66]:
```
# Drop 'total_users' column for plotting
plot_df = new_df.drop(columns=['total_users'])

# Reset index to use week numbers as a column for plotting
plot_df.reset_index(inplace=True)
plot_df.rename(columns={'index': 'week'}, inplace=True)

# Set the figure size for better visibility
plt.figure(figsize=(20, 12))

# Define custom colors for the bars (adjust as desired)
bar_colors = ['yellow', 'red', 'green', 'blue']  # Add blue for user_growth_rate

# Create a stacked bar plot for rates
bar_width = 0.3
index = plot_df.index
plt.bar(index, plot_df['retention_rate'], bar_width, label='Retention Rate',␣
 ↪color=bar_colors[0])
plt.bar(index, plot_df['churn_rate'], bar_width,␣
 ↪bottom=plot_df['retention_rate'], label='Churn Rate', color=bar_colors[1])
plt.bar(index, plot_df['resurrection_rate'], bar_width,␣
 ↪bottom=plot_df['retention_rate'] + plot_df['churn_rate'],␣
 ↪label='Resurrection Rate', color=bar_colors[2])
plt.bar(index, plot_df['user_growth_rate'], bar_width,␣
 ↪bottom=plot_df['retention_rate'] + plot_df['churn_rate'] +␣
 ↪plot_df['resurrection_rate'], label='User Growth Rate', color=bar_colors[3])␣
 ↪ # Add user_growth_rate bar

# Extract data for line plots
weeks = plot_df.index.to_numpy()
retention_rates = plot_df['retention_rate'].to_numpy()
churn_rates = plot_df['churn_rate'].to_numpy()
resurrection_rates = plot_df['resurrection_rate'].to_numpy()
user_growth_rates = plot_df['user_growth_rate'].to_numpy()  # Add␣
 ↪user_growth_rates

# Plot lines for each rate with different linestyles
# line_styles = ['-', '--', ':']  # Feel free to adjust linestyles
```

```python
plt.plot(weeks, retention_rates, label='Retention Rate', linestyle='-',␣
 ↪color='black')
plt.plot(weeks, churn_rates, label='Churn Rate', linestyle='-',␣
 ↪color=bar_colors[1])
plt.plot(weeks, resurrection_rates, label='Resurrection Rate', linestyle='-',␣
 ↪color=bar_colors[2])
plt.plot(weeks, user_growth_rates, label='User Growth Rate', linestyle='-',␣
 ↪color=bar_colors[3])  # Add user_growth_rate line plot

# Set plot labels and title
plt.xlabel('Week')
plt.ylabel('Rate')
plt.title('Weekly Trends for Retention, Churn, Resurrection, and User Growth␣
 ↪Rates (Stacked Bar with Lines)')
plt.xticks(rotation=45)

# Add a legend
plt.legend()

# Display the plot
plt.show()
```



The graph presents weekly trends for various user metrics, with the x-axis representing the weeks and the y-axis displaying the rates for retention, churn, resurrection, and user growth. Each metric is depicted using both bars and lines, providing a clear visualization of their behavior over time.

Interpretations:

1) Retention Rate (Black Line and Yellow Bars): Trend: The retention rate remains relatively stable over the weeks, with minor fluctuations.

Insights: A stable retention rate indicates that a consistent proportion of users continue to use the app week after week. This stability is crucial for maintaining a loyal user base and suggests that the app provides ongoing value to its users.

2) Churn Rate (Red Line and Bars): Trend: The Churn rate continuously decline in the first half but then starts increasing there after. The churn rate shows periodic peaks, indicating times when user attrition increases.

Insights: The periodic peaks in the churn rate highlight periods where user dissatisfaction might be higher or where there may be issues with the app. These peaks are critical to investigate to understand the reasons behind user drop-off and to implement strategies to mitigate this churn.

3) Resurrection Rate (Green Line and Bars): Trend: The resurrection rate exhibits a pattern of consistent spikes across different weeks.

Insights: The consistent spikes suggest that there are effective re-engagement strategies in place that periodically bring inactive users back to the app. Analyzing these periods can provide insights into which re-engagement tactics are most successful and how they can be optimized further.

4) User Growth Rate (Blue Line and Bars): Trend: The user growth rate is quite volatile, with significant peaks and troughs.

Insights: The volatility in the user growth rate indicates fluctuations in user acquisition and retention efforts. High growth periods may correlate with successful marketing campaigns or app updates, while troughs could indicate challenges in user retention or acquisition.

7 Phases-wise Analysis

7.0.1 Dividing the 56-week period into groups of 14 weeks (4 parts/phases)

[67]:
```
# Define the number of weeks in each part
part_size = 14
# Calculate the number of parts
num_parts = len(new_df) // part_size  # Integer division for complete parts

# Create a list containing week intervals for each part
week_intervals = []
for i in range(num_parts):
  start_week = 'w' + str(i * part_size + 1)  # Starting week index for each part
  end_week = 'w' + str((i + 1) * part_size)  # Ending week index for each part␣
↪(excluding last week)
  week_intervals.append((start_week, end_week))
print('Week Intervals:', week_intervals)
```

```
# Define part indexes and metrics
part_indexes = ['p1', 'p2', 'p3', 'p4']
metrics = ['retention_rate', 'user_growth_rate', 'churn_rate',␣
↪'resurrection_rate', 'user_replacement_ratio']
```

```
Week Intervals: [('w1', 'w14'), ('w15', 'w28'), ('w29', 'w42'), ('w43', 'w56')]
```

Analysis of Average Metric Trends Across All Phases

[68]:
```
# Create an empty DataFrame to store mean statistics
compare_mean_df = pd.DataFrame(index=part_indexes, columns=metrics)

# Loop through each week interval
for i, (start_week, end_week) in enumerate(week_intervals):
  # Select data for the current part
  part_data = new_df.loc[start_week:end_week]

  # Calculate mean statistics for the current part
  mean_data = (part_data.describe(include='all')[metrics]).loc['mean']

  # Fill corresponding row in compare_mean_df
  compare_mean_df.iloc[i] = mean_data

# Print the DataFrame with means for each part
print(compare_mean_df)
```

```
   retention_rate user_growth_rate churn_rate resurrection_rate  \
p1      61.355622         4.304121  31.501521          7.632073
p2      74.193298         0.106972  25.806702         10.651334
p3      75.417525         1.137661  24.582475         10.183581
p4      73.039572         0.643183  26.960428         12.895351

   user_replacement_ratio
p1               1.066232
p2               1.009246
p3               1.070962
p4               1.033453
```

Analysis of Metrices' Standard Deviation Trends Across All Phases

[80]:
```
# Create an empty DataFrame to store std statistics
compare_std_df = pd.DataFrame(index=part_indexes, columns=metrics)

# Loop through each week interval
for i, (start_week, end_week) in enumerate(week_intervals):
  # Select data for the current part
  part_data = new_df.loc[start_week:end_week]
```

```
    # Calculate std statistics for the current part
    std_data = (part_data.describe(include='all')[metrics]).loc['std']

    # Fill corresponding row in compare_std_df
    compare_std_df.iloc[i] = std_data

# Print the DataFrame with stds for each part
print(compare_std_df)
```

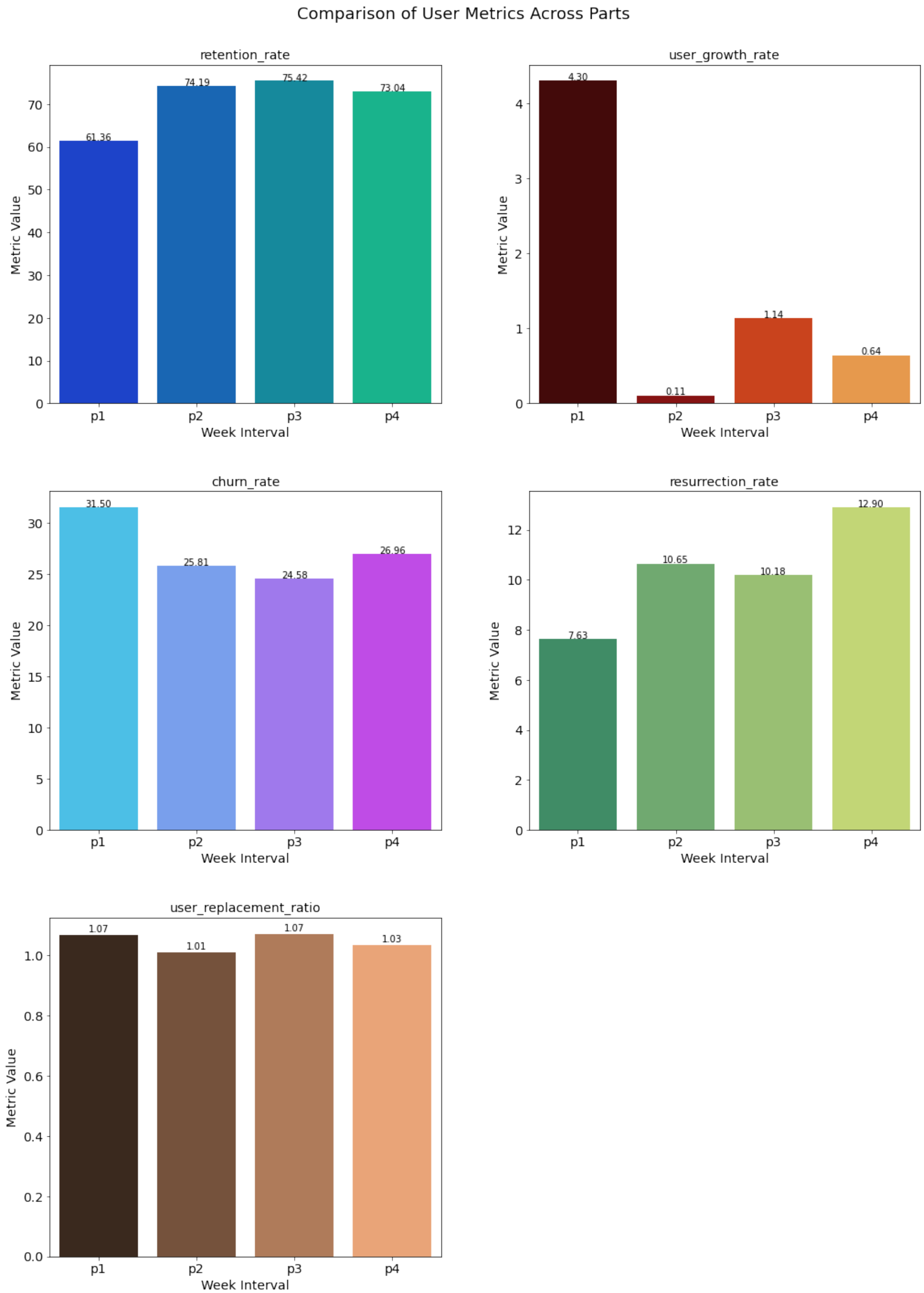
```
   retention_rate user_growth_rate churn_rate resurrection_rate  \
p1       18.24912         8.276521  10.167765           3.80565
p2       2.448866          3.35284   2.448866          1.644209
p3       4.128692         8.306663   4.128692          2.656343
p4       1.404459         5.727987   1.404459          2.680561

   user_replacement_ratio
p1               0.395091
p2               0.135565
p3               0.372234
p4               0.226408
```

7.0.2 Mean Comparision Visualization

[70]:
```
# Columns to be converted to float datatype
to_float = ['retention_rate', 'user_growth_rate', 'churn_rate',␣
 ↪'resurrection_rate', 'user_replacement_ratio']

# Convert specified columns to float datatype
compare_mean_df[to_float] = compare_mean_df[to_float].astype(float)
```

[71]:
```
import matplotlib.pyplot as plt
import seaborn as sns  # Import seaborn for barplot

metrics_to_plot = metrics
palettes = ['winter', 'gist_heat', 'cool', 'summer', 'copper', 'spring']

# Calculate the number of required subplots
num_subplots = len(metrics_to_plot)

# Create the figure and subplots (adjust rows and columns as needed)
fig, axes = plt.subplots(nrows=3,  ncols=2, figsize=(15, 20))

# Hide leftover subplots (if any)
for ax in axes.flat[num_subplots:]:
    ax.axis('off')  # Turn off axes and labels

# Flatten axes for easier access
```

```python
axes = axes.flatten()

# Create bar charts for each metric
for i, metric in enumerate(enumerate(metrics_to_plot)):
  # Access the subplot using its index
  ax = axes[i]
  sns.barplot(x=compare_mean_df.index, y=compare_mean_df[metric[1]], ax=ax,␣
 ↪palette=palettes[i])
  ax.set_title(metric[1], fontsize=14)  # Set subplot title using metric name
  ax.set_xlabel('Week Interval', fontsize=14)
  ax.set_ylabel('Metric Value', fontsize=14)

  # Annotate data labels on top of columns (average)
  for bar_container in axes[i].containers:  # Loop through bar containers in␣
 ↪the current subplot
    for bar in bar_container:
      yval = bar.get_height()  # Get bar height
      axes[i].text(bar.get_x() + bar.get_width() / 2, yval + 0.01, f"{yval:.
 ↪2f}", ha='center')  # Add label with 2 decimal places

# Adjust padding between subplots
plt.subplots_adjust(left=0.1, right=0.9, bottom=0.5, top=0.9)

# Loop through subplots (assuming axes are flattened)
for ax in axes.flat:
  ax.tick_params(labelsize=14)

# Add main title using figtext (positioned at the top center)
plt.figtext(0.5, 0.995, 'Comparison of User Metrics Across Parts', ha='center',␣
 ↪fontsize=18)
plt.tight_layout(pad=4)  # Adjust spacing between subplots

plt.show()
```



Comparison of User Metrics Across Parts
retention_rate
61.36
74.19
75.42
73.04
user_growth_rate
4.30
0.11
1.14
0.64
churn_rate
31.50
25.81
24.58
26.96
resurrection_rate
7.63
10.65
10.18
12.90
user_replacement_ratio
1.07
1.01
1.07
1.03
Metric Value
Week Interval
p1
p2
p3
p4

Interpretation:

1) Retention Rate:

- The retention rate shows a significant increase from Phase 1 to Phase 2 and remains relatively stable with a slight increase in Phase 3 and slight decrease in Phase 4. This indicates that over time, the app has been able to retain a higher percentage of users, suggesting improvements in user experience or engagement strategies.

2) User Growth Rate:

- The data suggests that the startup experienced a very strong start in Phase 1, with the highest user growth rate of 4.30%. This could be due to initial launch activities and marketing campaigns. However, the growth rate significantly dropped in Phase 2 to 0.11%, which might indicate initial market saturation or the need for revised acquisition strategies.
- In Phase 3, the growth rate improved to 1.14%, suggesting that the company managed to address some of the challenges faced in Phase 2, possibly through improved marketing tactics or enhanced user experience. Phase 4 saw a slight decrease to 0.64%, but this rate is still higher than Phase 2, indicating a relatively stable growth trajectory in the later stages.

3) Churn Rate:

- In phase 1, the churn rate is at its highest, indicating initial difficulties in retaining users. This suggests users might be leaving due to dissatisfaction, bugs, or unmet expectations. Phase 2 noticed a drop in churn rate from Phase 1. This improvement suggests that the initial issues were addressed, leading to better user retention.
- The churn rate reached its lowest in Phase 3, indicating improving user retention strategies. The slight increase in Phase 4 suggests a need for ongoing efforts to address user retention to prevent a further rise in churn rates.

4) Resurrection Rate

- The resurrection rate shows a steady increase from Phase 1 to Phase 4, indicating progressively better strategies in re-engaging churned users.
- The highest rate in Phase 4 suggests the implementation of highly effective reactivation campaigns or improvements in the user experience that entice former users to return.

5) User Replacement Ratio

- Phase 1 shows a slight positive growth with a user replacement ratio slightly above 1, indicating that user acquisition and reactivation efforts are sufficient to exceed user attrition. In Phase 2, the ratio drops close to 1, signaling stagnation where new user gains just offset user losses, suggesting potential weaknesses in acquisition or retention strategies.
- Moving to Phase 3, the ratio returns to 1.07, suggesting improved effectiveness in acquiring and reactivating users, leading to modest growth. Phase 4 maintains a ratio of 1.03, indicating continued positive growth, albeit less pronounced than in Phases 1 and 3, highlighting the ongoing need to strengthen user acquisition and retention efforts.

7.0.3 Cohort Retention Rate:

1) Grouping by Acquisition Week:

- Think of each week as a "batch" of new users who signed up for your app or website. We call these groups "cohorts."

2) Tracking Retention Over Time:

- We then track how many users from each "batch" (cohort) remain active in the following weeks.

3) Identifying Effective Acquisition Efforts:

- By comparing the retention rates of different cohorts, you can see which marketing strategies (e.g., social media ads, referral programs) led to "batches" of users who stick around longer.
- The cohorts with higher retention rates indicate more effective acquisition efforts.

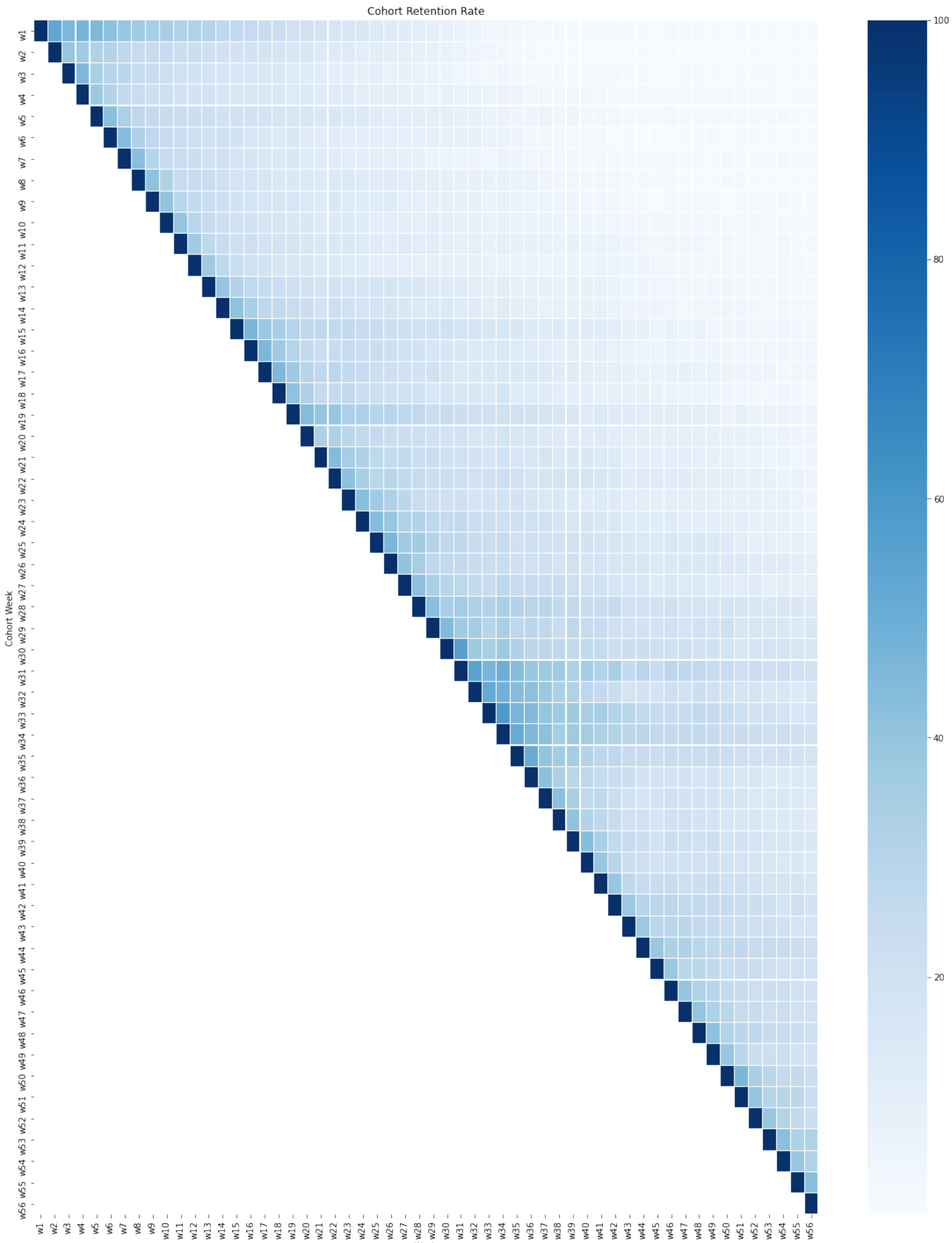
Benifits:

- Understanding User Behavior: You can see if users acquired during a specific promotion are more likely to churn (stop using your app) or become loyal users.
- Optimizing Marketing Strategies: This helps you focus on marketing channels that bring users who are more likely to stick around in the long run.

[121]:
```
# Step 1: Identify Cohorts
user_first_week = {}

for week in df.columns:
    for user in df[week].dropna():
        if user not in user_first_week:
            user_first_week[user] = week

# Create a DataFrame for cohort analysis
cohort_df = pd.DataFrame(list(user_first_week.items()), columns=['user',␣
 ↪'first_week'])
print('Cohort df:\n')
print(cohort_df.head(2))

# Step 2: Track Retention
weeks = df.columns.tolist()
cohort_retention = pd.DataFrame(index=weeks, columns=weeks)

for start_week in weeks:
    cohort_users = cohort_df[cohort_df['first_week'] == start_week]['user']
    for current_week in weeks:
        if weeks.index(current_week) >= weeks.index(start_week):
            active_users = df[current_week].dropna()
            retention_count = len(set(active_users) & set(cohort_users))
            cohort_size = len(cohort_users)
```

```
            retention_rate = retention_count / cohort_size if cohort_size > 0␣
↪else 0
            cohort_retention.at[start_week, current_week] = retention_rate
        else:
            cohort_retention.at[start_week, current_week] = np.nan

# Convert retention rates to percentages
cohort_retention = cohort_retention.astype(float) * 100

# Hitmap to Visualize cohort_retention
plt.figure(figsize=(20, 25))
sns.heatmap(cohort_retention, cmap="Blues", linewidth = 0.1)
plt.title('Cohort Retention Rate')
plt.xlabel('Week')
plt.ylabel('Cohort Week')
plt.show()
```

```
Cohort df:

               user first_week
0  fd7c28f9fd8045f2         w1
1  54910d2b363221e1         w1
```


Cohort Retention Rate
Cohort Week
Week
100
80
60
40
20

7.0.4 Finding Best Weeks

[120]:
```
weeks = cohort_retention.columns.tolist()
avg_retention = {}
for week in weeks:
    avg_retention[week] = cohort_retention.loc[week].mean()

# Define the number of parts
num_parts = 6

# Calculate the number of weeks per part (rounded down)
weeks_per_part = len(weeks) // num_parts

# Divide avg_retention into parts
part_data = {}
for i in range(num_parts):
    start_week_index = i * weeks_per_part
    end_week_index = min((i + 1) * weeks_per_part, len(weeks))  # Handle␣
↪potential last part with fewer weeks
    part_data[f"Part {i+1}"] = {k: v for k, v in avg_retention.items() if weeks.
↪index(k) >= start_week_index and weeks.index(k) < end_week_index}

# Find top 2 weeks in each part
best_weeks_per_part = {}
for part_name, part_dict in part_data.items():
    # Sort weeks in the part by retention value (descending)
    sorted_weeks = sorted(part_dict.items(), key=lambda item: item[1],␣
↪reverse=True)
    # Select top 2 weeks
    best_weeks_per_part[part_name] = sorted_weeks[:2]

# Print results
for part_name, best_weeks in best_weeks_per_part.items():
    print(f"\n**{part_name}**:")
    for week, retention in best_weeks:
    print(f"- Week: {week}, Average Retention: {retention:.2f}")
```

```
**Part 1**:
- Week: w1, Average Retention: 16.51
- Week: w5, Average Retention: 12.58

**Part 2**:
- Week: w15, Average Retention: 17.77
- Week: w17, Average Retention: 16.98

**Part 3**:
- Week: w27, Average Retention: 21.98
```

```
- Week: w25, Average Retention: 21.83

**Part 4**:
- Week: w31, Average Retention: 33.87
- Week: w34, Average Retention: 32.63

**Part 5**:
- Week: w44, Average Retention: 33.38
- Week: w45, Average Retention: 31.30

**Part 6**:
- Week: w54, Average Retention: 56.81
- Week: w53, Average Retention: 51.74
```

We can Target strategies applied in The weeks with High Average Retention to Grow Users.

7.0.5 Part-wise Cohort Retention Matrix Visualization

[100]:

```python
# Function to calculate cohort retention and visualize heatmap
def analyze_cohort_retention(df):
    # Step 1: Identify Cohorts
    user_first_week = {}

    for week in df.columns:
        for user in df[week].dropna():
            if user not in user_first_week:
                user_first_week[user] = week

    cohort_df = pd.DataFrame(list(user_first_week.items()), columns=['user', 'first_week'])

    # Step 2: Track Retention
    weeks = df.columns.tolist()
    cohort_retention = pd.DataFrame(index=weeks, columns=weeks)

    for start_week in weeks:
        cohort_users = cohort_df[cohort_df['first_week'] == start_week]['user']
        for current_week in weeks:
            if weeks.index(current_week) >= weeks.index(start_week):
                active_users = df[current_week].dropna()
                retention_count = len(set(active_users) & set(cohort_users))
                cohort_size = len(cohort_users)
                retention_rate = retention_count / cohort_size if cohort_size > 0 else 0
                cohort_retention.at[start_week, current_week] = retention_rate
            else:
                cohort_retention.at[start_week, current_week] = np.nan
```

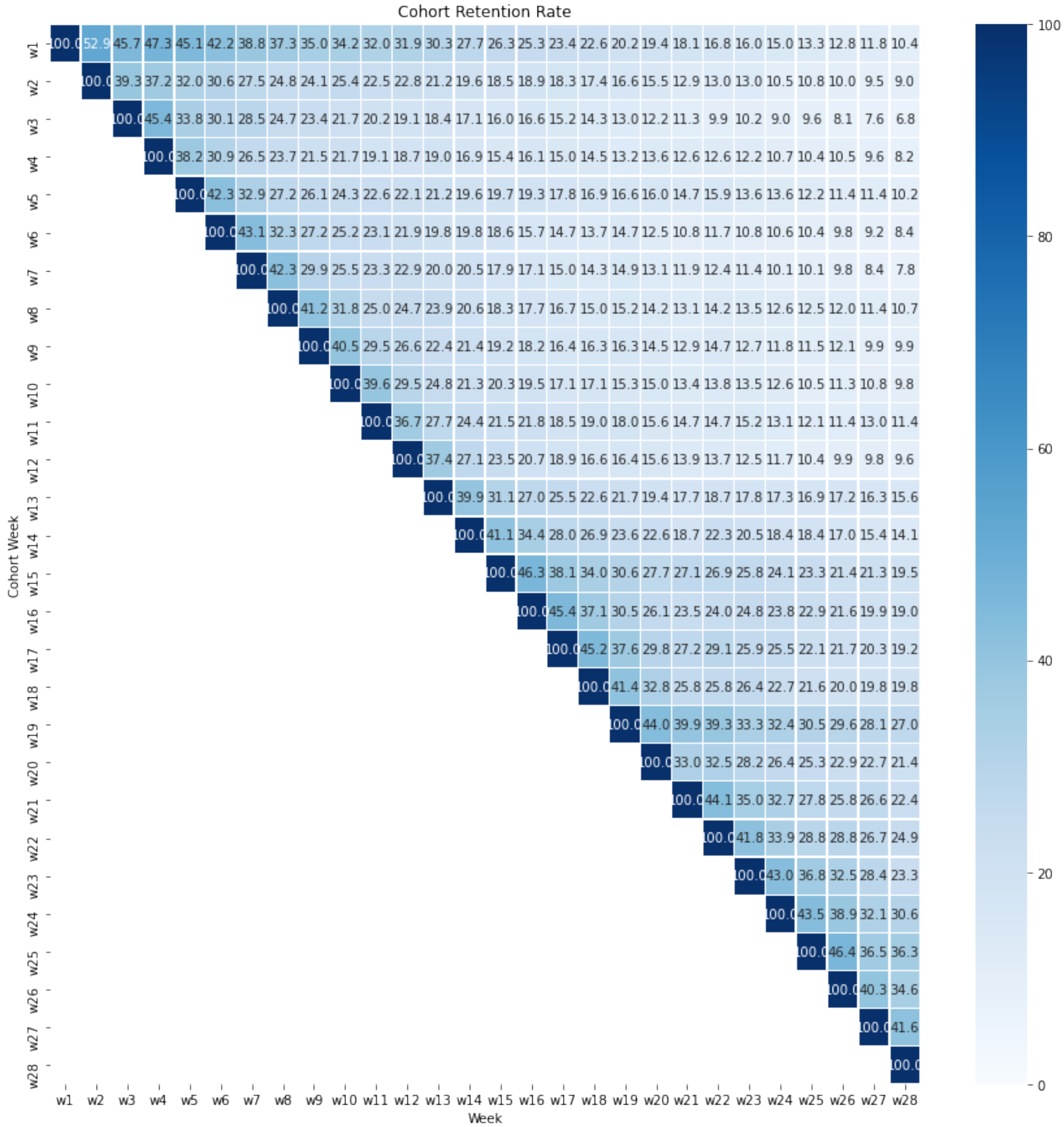
```
    # Convert retention rates to percentages
    cohort_retention = cohort_retention.astype(float) * 100

    # Visualize cohort retention heatmap
    plt.figure(figsize=(15, 15))
    #sns.heatmap(cohort_retention, annot=True, fmt=".1f", cmap="PiYG")
    sns.heatmap(data=cohort_retention, annot = True, cmap = "Blues", vmin = 0.
↪0, fmt = '.1f', linewidth = 0.3)
    plt.title('Cohort Retention Rate')
    plt.xlabel('Week')
    plt.ylabel('Cohort Week')
    plt.show()
```

[106]:

```
# Divide into three parts: part1 (weeks 1-28), part2 (weeks 18-48) and part3␣
↪(weeks 29-56)
part1 = df.loc[ :, 'w1':'w28']
part2 = df.loc[ :, 'w18':'w48']
part3 = df.loc[ :, 'w29':'w56']
```

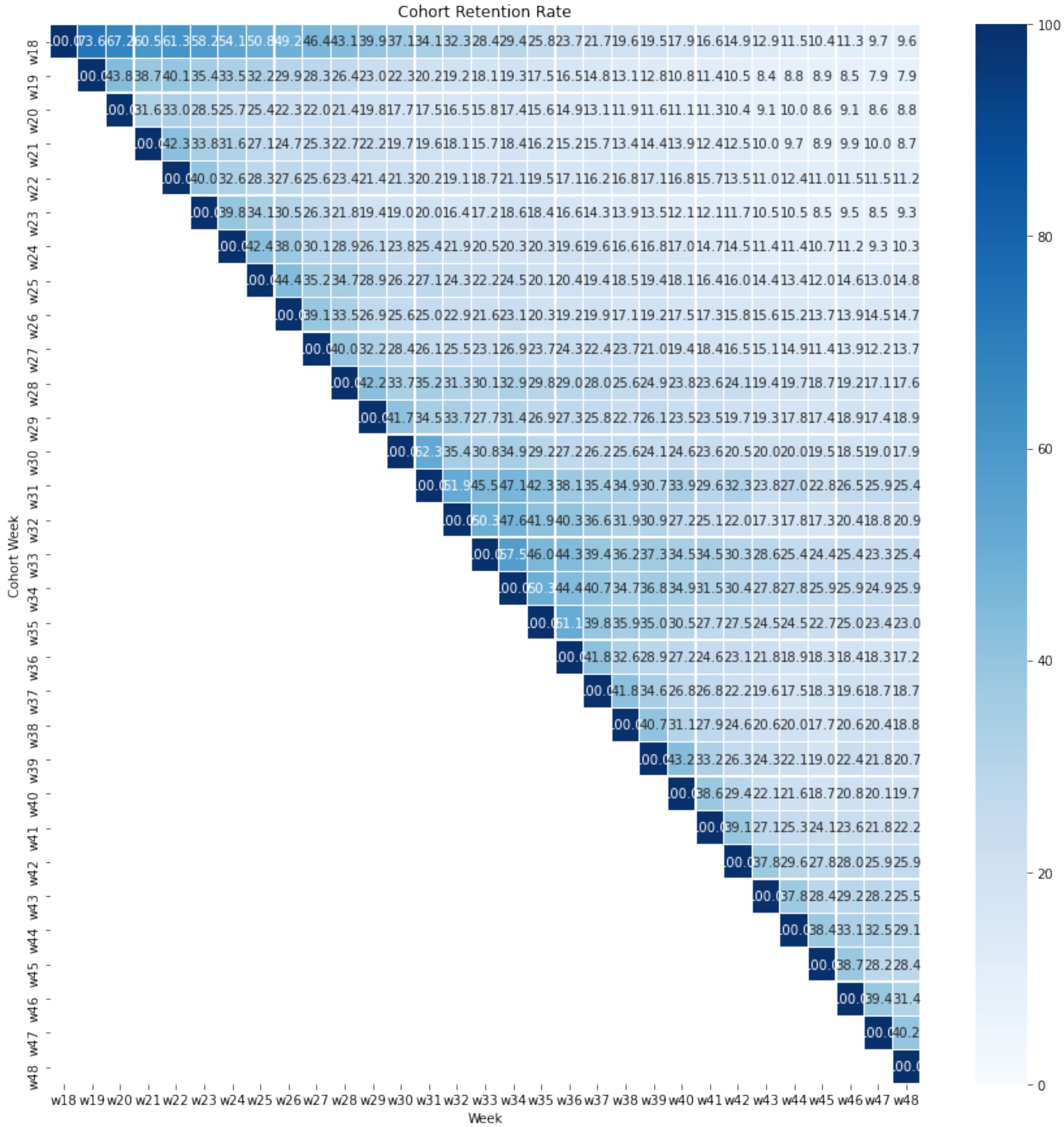
[104]:

```
analyze_cohort_retention(part1)
```

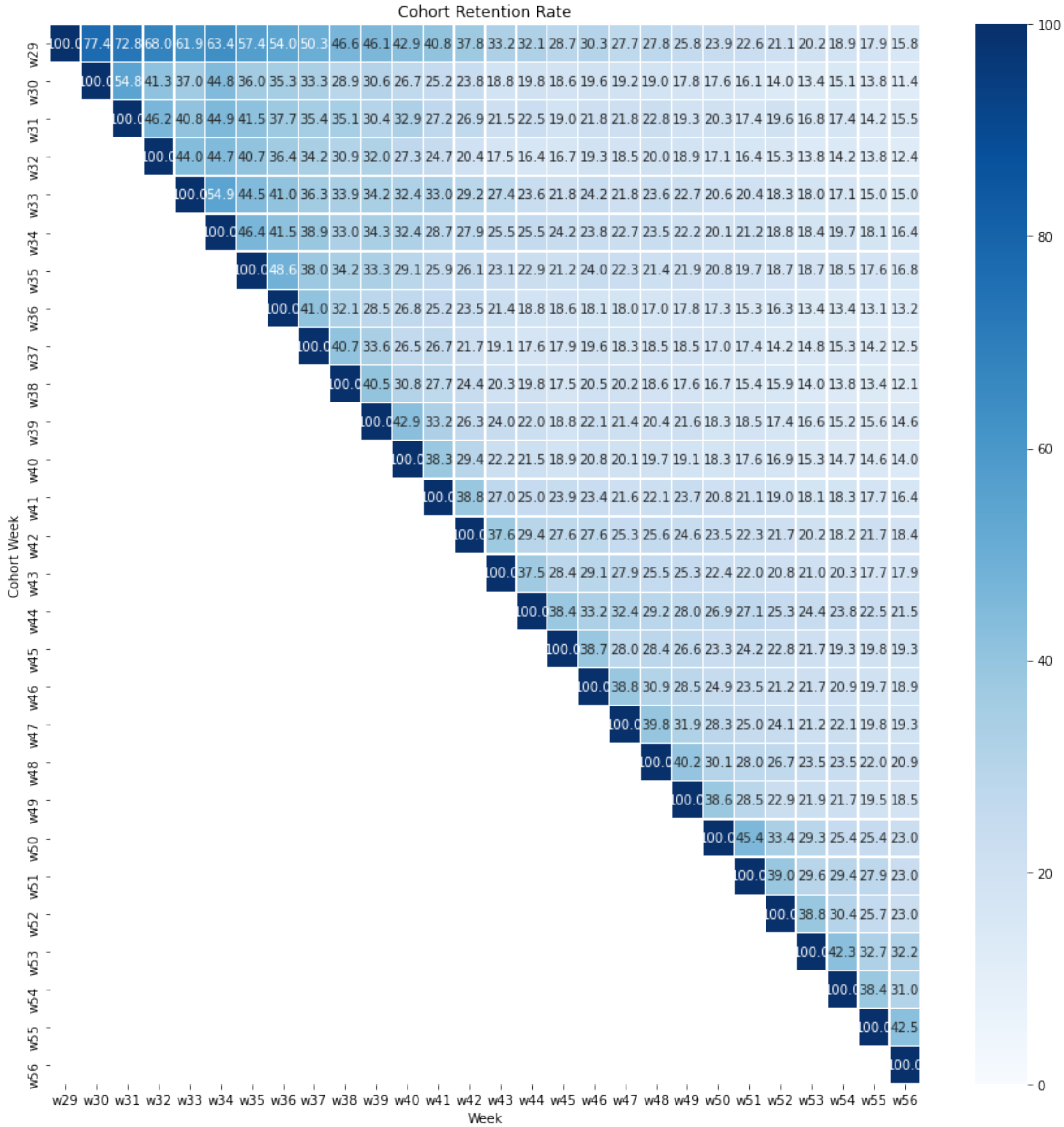


[107]:

```
analyze_cohort_retention(part2)
```



[108]:
```
analyze_cohort_retention(part3)
```



8 Report

Based on the interpretations, the business shows signs of both growth and areas for improvement. Here's a breakdown of the health indicators

Positive Signs:

- Retention Rate: Increasing retention across phases indicates an improving user experience or engagement strategies.
- User Growth: While the initial strong growth plateaued, it shows some recovery and remains positive in later phases.

- Churn Rate: Decreasing churn throughout most phases suggests successful efforts to retain users.
- Resurrection Rate: Steadily increasing reactivation signifies effectiveness in bringing back churned users.
- User Replacement Ratio: Generally positive ratios imply user acquisition and reactivation outpace churn.

Areas for Improvement:

- User Growth Rate: The initial drop in Phase 2 suggests a need for revised user acquisition strategies. The lower growth rate in Phase 4 compared to Phase 1 indicates room for further optimization.
- Churn Rate: Although decreasing, the slight increase in Phase 4 highlights the need for continued efforts to minimize user churn.

Overall Health:

- The business exhibits a mix of positive growth trends and areas needing improvement. It's not necessarily declining, but there's potential for further growth.

Here are some possibilities:

- Growing: The increasing retention rate and positive user replacement ratios suggest the business is on a growth trajectory. However, the plateauing user growth rate indicates potential for more substantial expansion.
- Attracting Users: The positive user replacement ratios imply user acquisition and reactivation efforts are attracting new users. However, there's room to improve the user growth rate for more significant user acquisition.
- Can Grow More: With continued refinement of user acquisition strategies and ongoing efforts to minimize churn, the business has the potential for more substantial and sustainable growth.

Recommendations:

- Analyze user acquisition channels and identify areas for improvement to revitalize user growth.
- Investigate reasons behind user churn and implement strategies to address them.
- Continuously monitor and refine user engagement strategies to keep retention rates high.
- Maintain focus on reactivation efforts to capitalize on the increasing resurrection rate.
- By addressing these areas, the business can solidify its growth trajectory and achieve more substantial user acquisition and retention.

9 What more can be done to grow application

9.0.1 Viral Coefficient:

Concept: * K factor estimates how many new users a single existing user can bring to the platform. It considers both referrals (direct invitations) and network effects (indirect growth through friends using the product).

Data Needed:

1) Active Users: This is the baseline, representing the existing user base that can potentially refer others. You likely already have this data in your weekly active user IDs.

2) Referrals: This tracks how many invitations existing users send to their friends. You might need an invite tracking system to capture this data.

3) Referral Conversion Rate: This tells you what percentage of invitations convert into new users. Ideally, you'd track how many sign-ups originated from referrals.

Calculation: * K factor is simply the number of referrals per user multiplied by the referral conversion rate.

Interpretation:

- K factor > 1: This indicates viral growth, where each user brings in more than one new user on average. Organic user acquisition is likely happening through referrals and network effects.
- K factor $= 1$: This suggests stable growth, where each user replaces themselves with one new user. No exponential growth, but the user base isn't shrinking either.
- K factor < 1: This implies non-viral growth. You might need to consider paid marketing efforts to acquire new users.