In [1]:
```python
import pandas as pd
import numpy as np
import plotly.express as px
from scipy import stats
import plotly.graph_objects as go
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```python
df = pd.read_csv("cookie_cats.csv")
```

In [3]:
```python
df
```

Out[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
...	...	...	...	...	...
90184	9999441	gate_40	97	True	False
90185	9999479	gate_40	30	False	False
90186	9999710	gate_30	28	True	False
90187	9999768	gate_40	51	True	False
90188	9999861	gate_40	16	False	False

90189 rows × 5 columns

In [4]:
```python
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 [5]:
```python
df.describe()
```

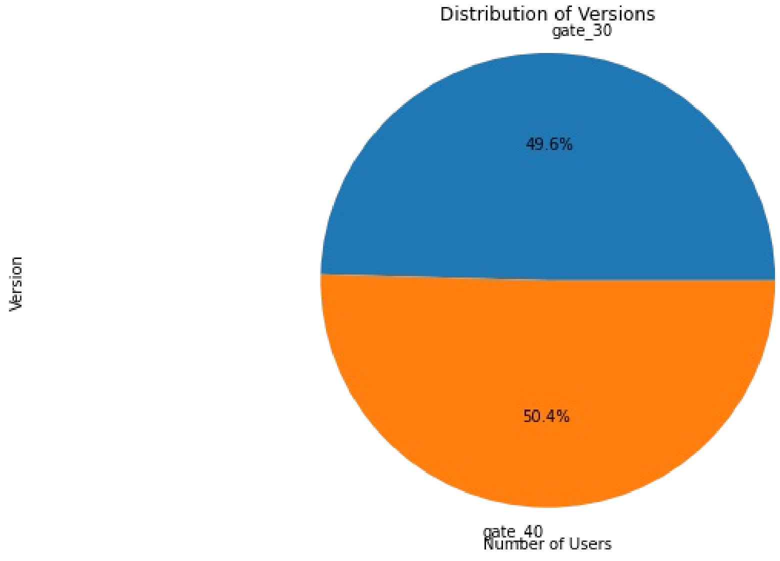
Out[5]:

	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 [6]:
```python
pie_data = df[['userid', 'version']].groupby('version').count().reset_index()
print(pie_data)

plt.figure(figsize=(12, 6))
plt.pie(pie_data['userid'], labels=pie_data['version'], autopct='%1.1f%%')
plt.title('Distribution of Versions')
plt.xlabel('Number of Users')
```

```
plt.ylabel('Version')
plt.axis('equal')
plt.show()
```

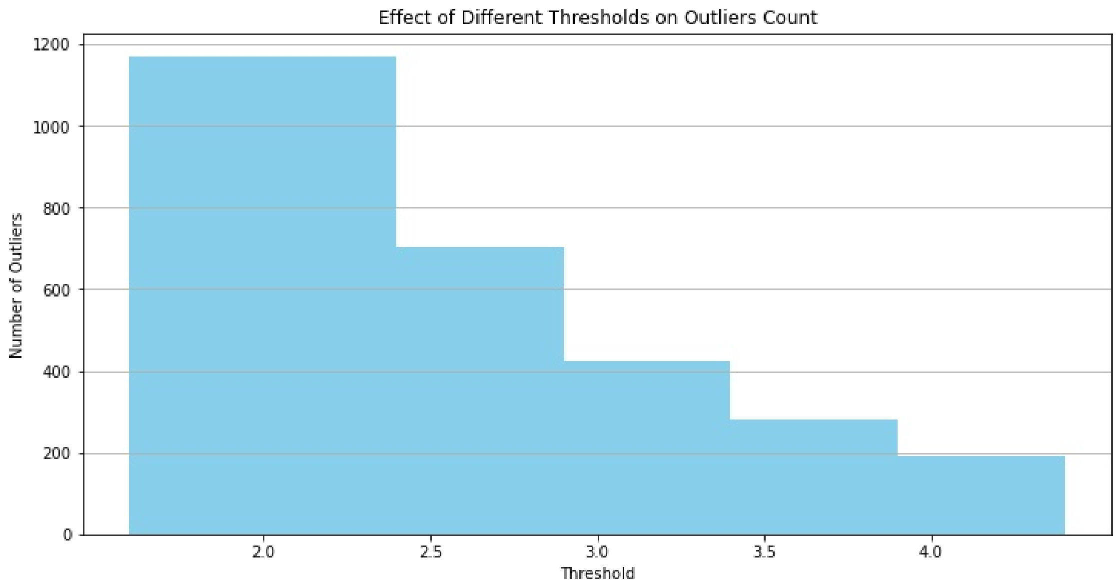
```
   version  userid
0  gate_30   44700
1  gate_40   45489
```



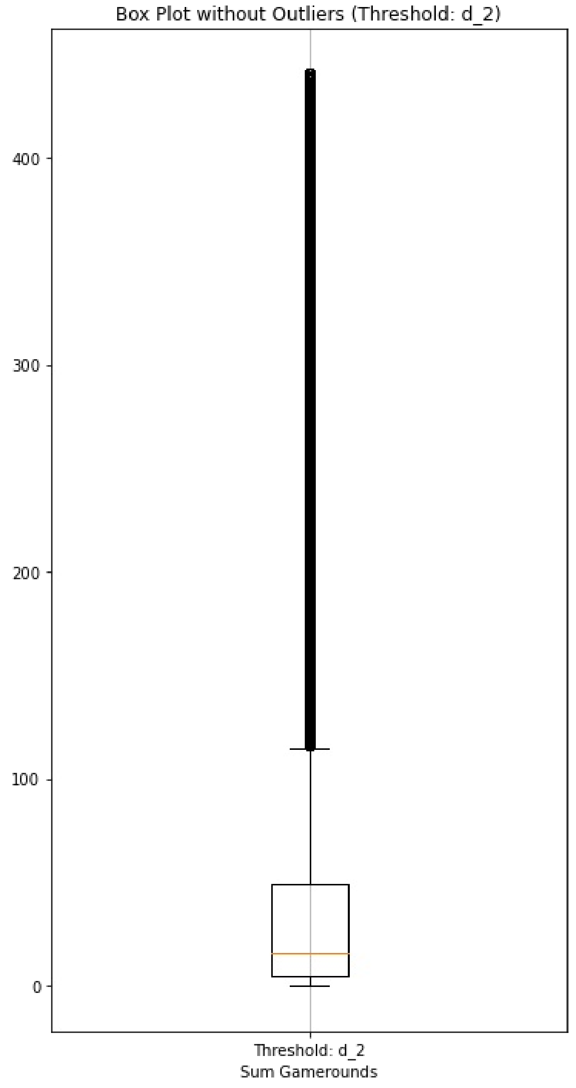
LET'S SEE IF WE SHOULD CLEAN OUR DATA OR NOT

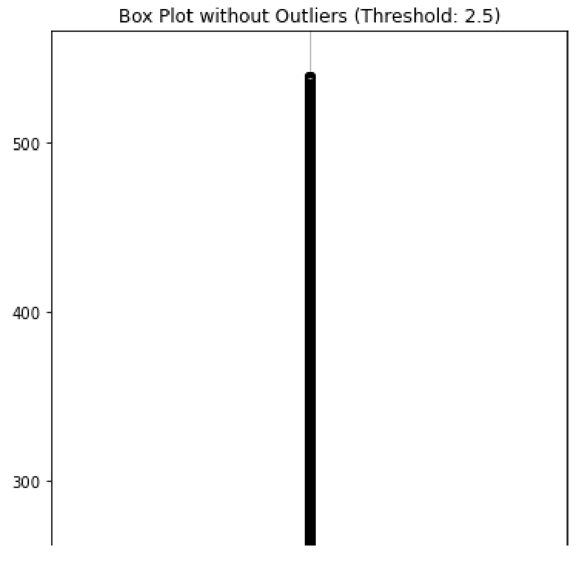
In [7]:

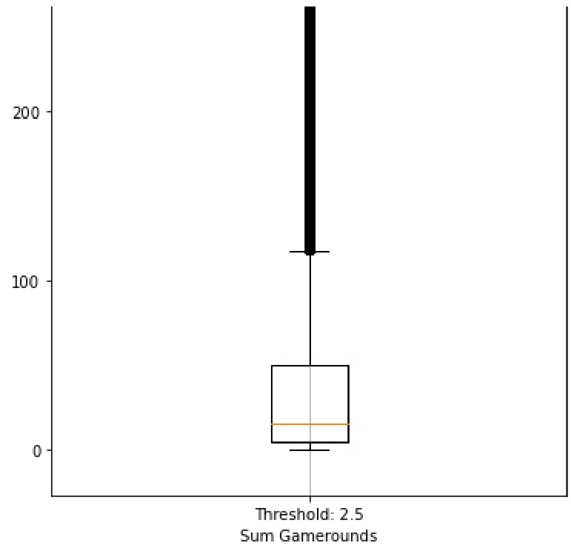
```
# Calculating Z-scores for 'sum_gamerounds'
z_scores = stats.zscore(df['sum_gamerounds'])

# Trying different threshold values and evaluate the number of outliers for each
thresholds_to_try = [2, 2.5, 3, 3.5, 4]
outliers_count = []

for threshold in thresholds_to_try:
    outliers = df[abs(z_scores) > threshold]
    outliers_count.append(len(outliers))

# Plotting the effect of different thresholds on the number of outliers
plt.figure(figsize=(12, 6))
plt.bar(thresholds_to_try, outliers_count, color='skyblue')
plt.title('Effect of Different Thresholds on Outliers Count')
plt.xlabel('Threshold')
plt.ylabel('Number of Outliers')
plt.xticks(thresholds_to_try)
plt.grid(axis='y')
plt.show()
```

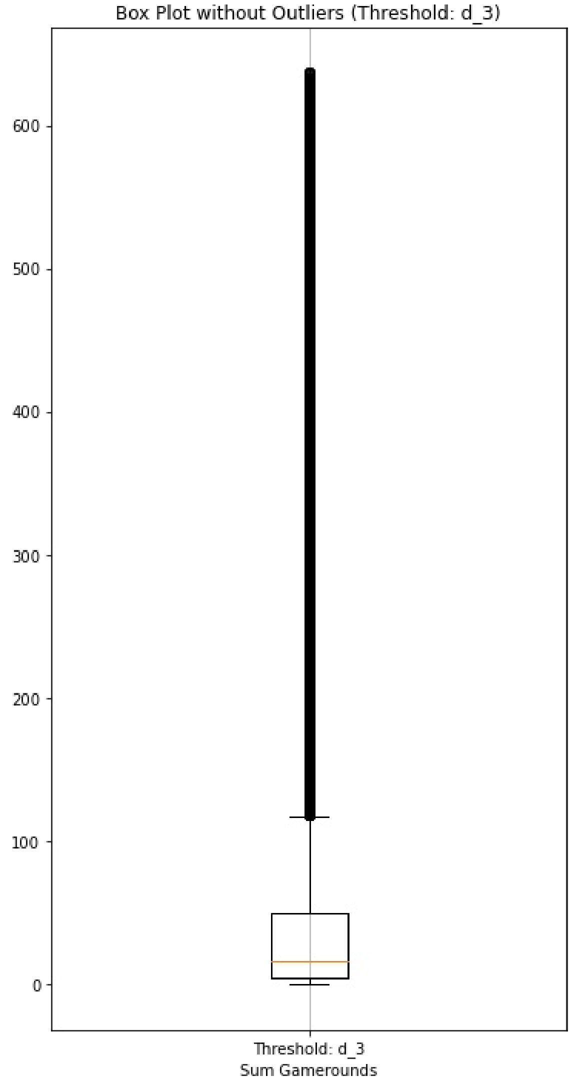


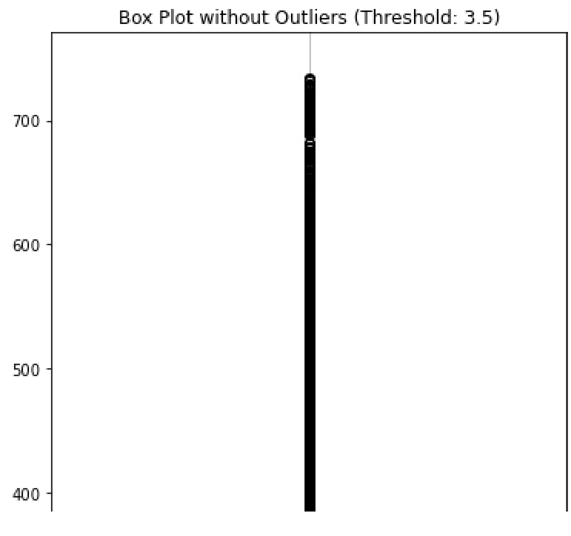
```python
# Threshold values to try
threshold_values = [2, 2.5, 3, 3.5, 4]

# Calculating Z-scores for different thresholds and create new dataframes without outliers
dfs_no_outliers = {}  # Dictionary to store dataframes without outliers for different thresholds

for threshold in threshold_values:
    z_scores = stats.zscore(df['sum_gamerounds'])
    outliers = df[abs(z_scores) > threshold]
    df_no_outlier = df[abs(z_scores) <= threshold]
    dfs_no_outliers[f"df_no_outlier_threshold_{threshold}"] = df_no_outlier

# Creating box plots for each dataframe without outliers (for each threshold)
for key, df_no_outlier in dfs_no_outliers.items():
    plt.figure(figsize=(6, 12))
    plt.boxplot(df_no_outlier['sum_gamerounds'], labels=[f'Threshold: {key[-3:]}'])
    plt.xlabel('Sum Gamerounds')
    plt.title(f'Box Plot without Outliers (Threshold: {key[-3:]})')
    plt.grid(axis='x')
    plt.show()
```

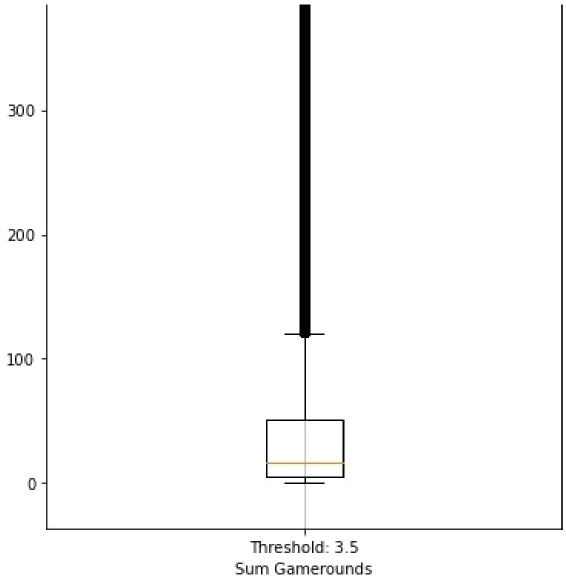


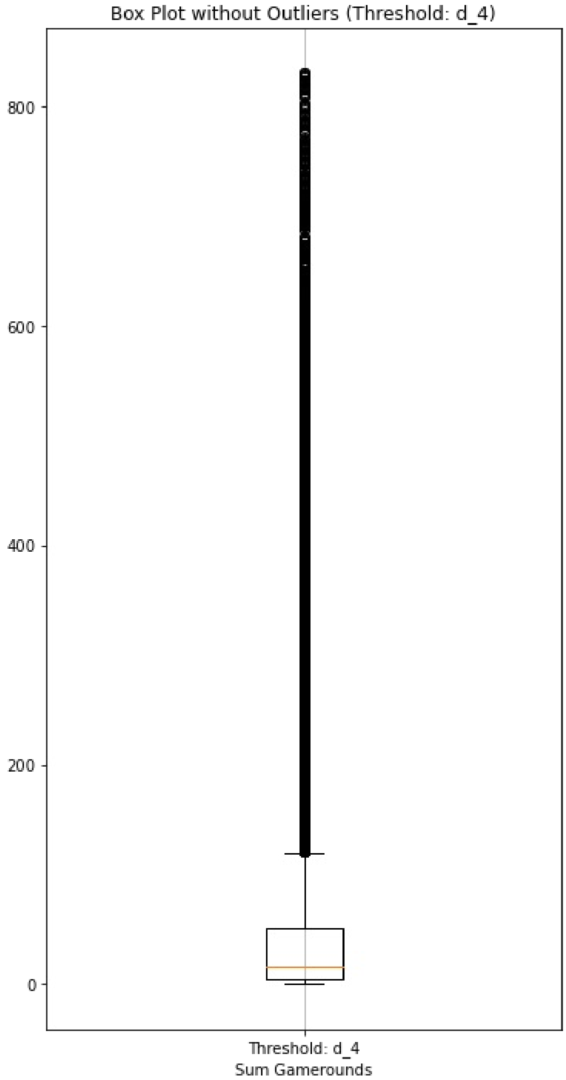



200
100
0
Threshold: 2.5
Sum Gamerounds


Box Plot without Outliers (Threshold: d_3)
600
500
400
300
200
100
0
Threshold: d_3
Sum Gamerounds


Box Plot without Outliers (Threshold: 3.5)
700
600
500
400

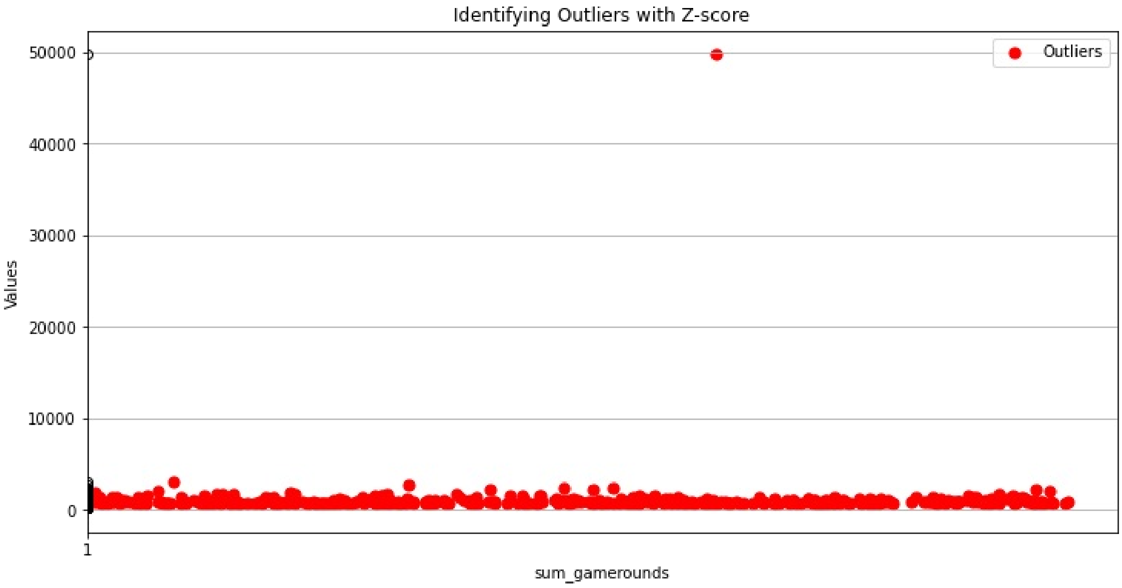




In [9]:

```
# Calculating Z-scores for 'sum_gamerounds'
z_scores = stats.zscore(df['sum_gamerounds'])

# Defining a threshold for outliers (Z-score greater than 3)
threshold = 3

# Identifying outliers
outliers = df[abs(z_scores) > threshold]

# Creating plot to see outliers
plt.figure(figsize=(12, 6))
plt.boxplot(df['sum_gamerounds'])
plt.title('Identifying Outliers with Z-score')
plt.xlabel('sum_gamerounds')
plt.ylabel('Values')
```

```
plt.grid()

# Highlighting outliers
plt.scatter(outliers.index, outliers['sum_gamerounds'], color='red', label='Outliers', s=50)
plt.legend()

plt.show()

# Outliers DataFrame
print("Outliers:")
print(outliers)
```




```
Outliers:
        userid  version  sum_gamerounds  retention_1  retention_7
601      63617  gate_30             902         True         True
655      69927  gate_30            1906         True         True
865      97308  gate_30             798         True         True
1097    121303  gate_30            1374         True         True
1264    139072  gate_40             681        False         True
...        ...      ...             ...          ...          ...
88328  9791599  gate_40            2063         True         True
88354  9794383  gate_40             846         True         True
88590  9822327  gate_40             768         True         True
89719  9949589  gate_40             708         True         True
89921  9971042  gate_30             892         True         True

[425 rows x 5 columns]
```

In [10]:

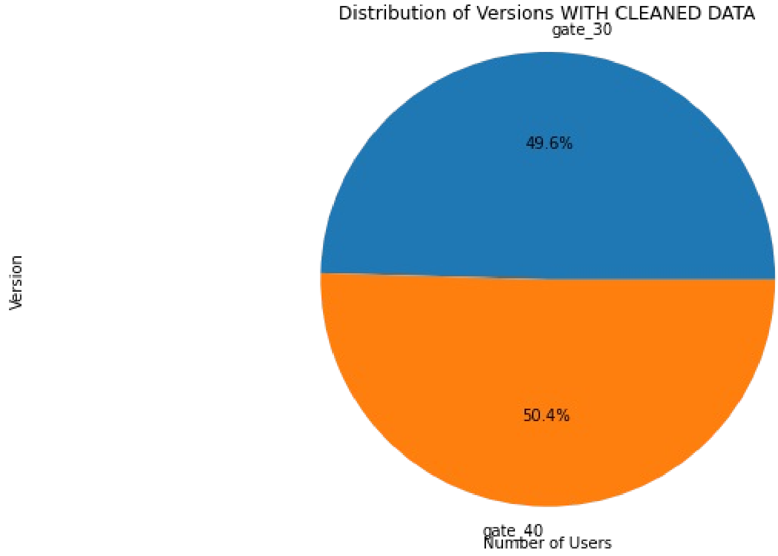
```
# Creating a new dataframe called cleaned_df

z_scores = stats.zscore(df['sum_gamerounds'])

# Defining a threshold for outliers (Z-score greater than 3)
threshold = 3

# Identifying outliers
outliers = df[abs(z_scores) > threshold]

# Creating a new DataFrame without outliers
cleaned_df = df[abs(z_scores) <= threshold]

print("Cleaned DataFrame without outliers (Threshold=3):")
print(cleaned_df)
```

```
Cleaned DataFrame without outliers (Threshold=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
...        ...      ...             ...          ...          ...
90184  9999441  gate_40              97         True        False
90185  9999479  gate_40              30        False        False
90186  9999710  gate_30              28         True        False
90187  9999768  gate_40              51         True        False
90188  9999861  gate_40              16        False        False

[89764 rows x 5 columns]
```

We deleted nearly 400 outliers and they didn't change almost anything in distribution:

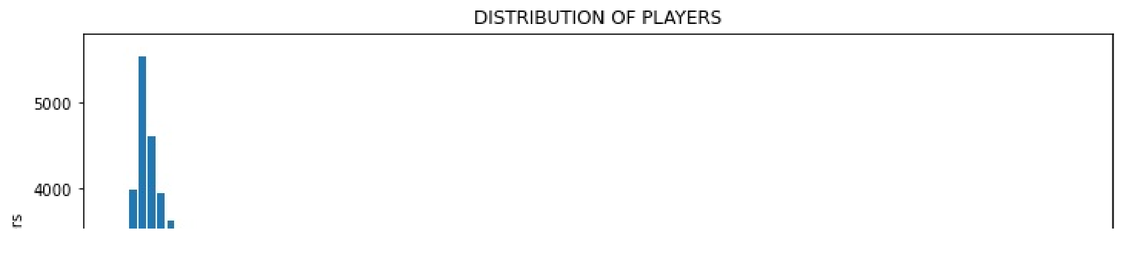
In [11]:
```
pie_data = cleaned_df[['userid', 'version']].groupby('version').count().reset_index()
print(pie_data)

# Creating a pie chart using Matplotlib
plt.figure(figsize=(12, 6))
plt.pie(pie_data['userid'], labels=pie_data['version'], autopct='%1.1f%%')
plt.title('Distribution of Versions WITH CLEANED DATA')
plt.xlabel('Number of Users')
plt.ylabel('Version')
plt.axis('equal')
plt.show()
```

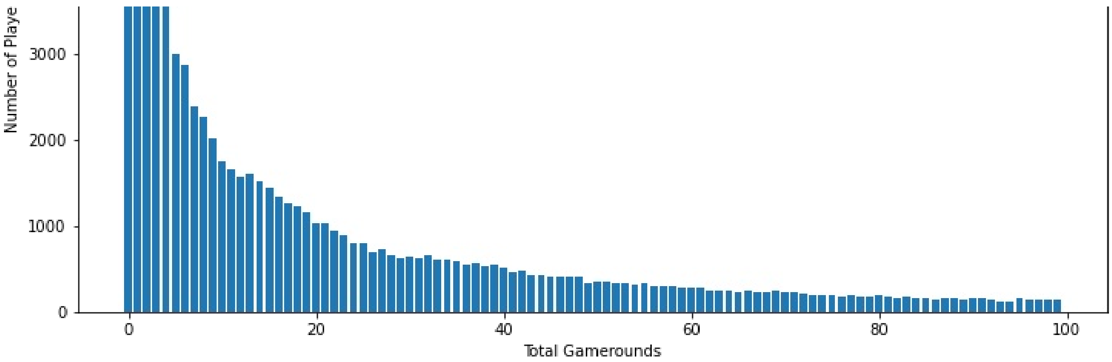
```
   version  userid
0  gate_30   44500
1  gate_40   45264
```



In [12]:
```
plot_df = cleaned_df.groupby("sum_gamerounds")["userid"].count().reset_index(name='count')

print(plot_df)
# Distribution of players that played 0 to 100 game rounds
plt.figure(figsize=(12, 6))
plt.bar(plot_df.head(100)['sum_gamerounds'], plot_df.head(100)['count'])
plt.xlabel('Total Gamerounds')
plt.ylabel('Number of Players')
plt.title('DISTRIBUTION OF PLAYERS')
plt.show()
```

```
     sum_gamerounds  count
0                 0   3994
1                 1   5538
2                 2   4606
3                 3   3958
4                 4   3629
..              ...    ...
629             633      3
630             634      1
631             635      1
632             636      2
633             637      3

[634 rows x 2 columns]
```





In [13]:

```
print("Players installed the game but then never played it:")
cleaned_df[cleaned_df["sum_gamerounds"]== 0]["userid"].count()
```

```
Players installed the game but then never played it:
```

Out[13]:

```
3994
```

In [ ]:

In [14]:

```
print("Day-1 Retention %")
retention_percentage = (cleaned_df['retention_1'].sum() / cleaned_df['retention_1'].count()) * 100
print(retention_percentage)


print("\nDay-7 Retention %")
retention_percentage = (cleaned_df['retention_7'].sum() / cleaned_df['retention_7'].count()) * 100
print(retention_percentage)
```

```
Day-1 Retention %
44.276101777995635

Day-7 Retention %
18.23670959404661
```

In [15]:

```
day_1_retention = cleaned_df.groupby('version')['retention_1'].mean() * 100
print("Day-1 Retention %")
print(day_1_retention)

day_7_retention = cleaned_df.groupby('version')['retention_7'].mean() * 100
print("\nDay-7 Retention %")
print(day_7_retention)
```

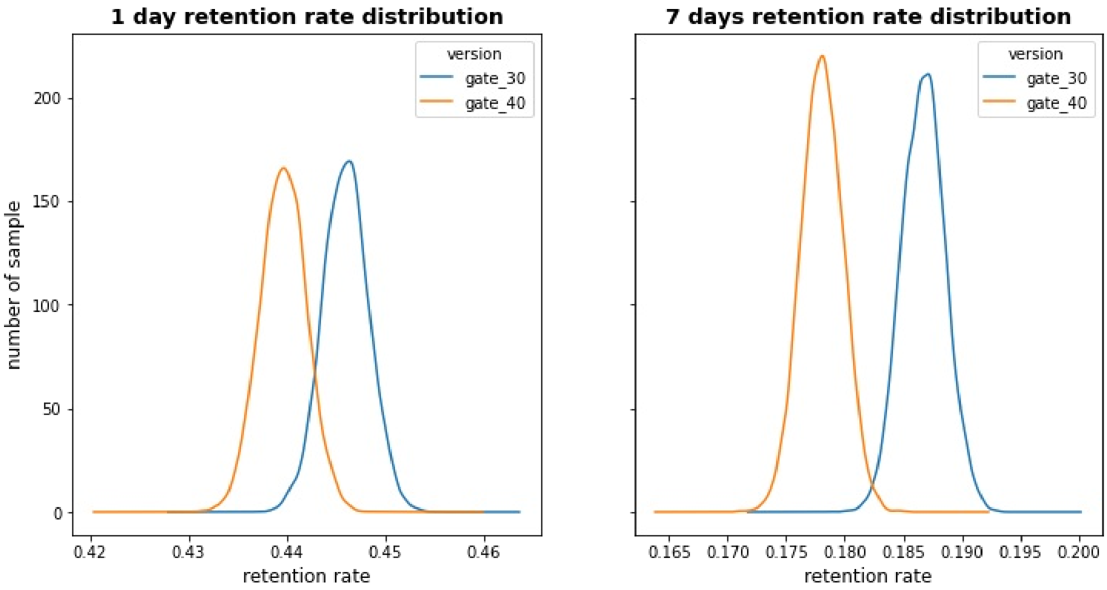
```
Day-1 Retention %
version
gate_30    44.593258
gate_40    43.964298
Name: retention_1, dtype: float64

Day-7 Retention %
version
gate_30    18.676404
gate_40    17.804436
Name: retention_7, dtype: float64
```

BOOTSTRAPING

In [16]:

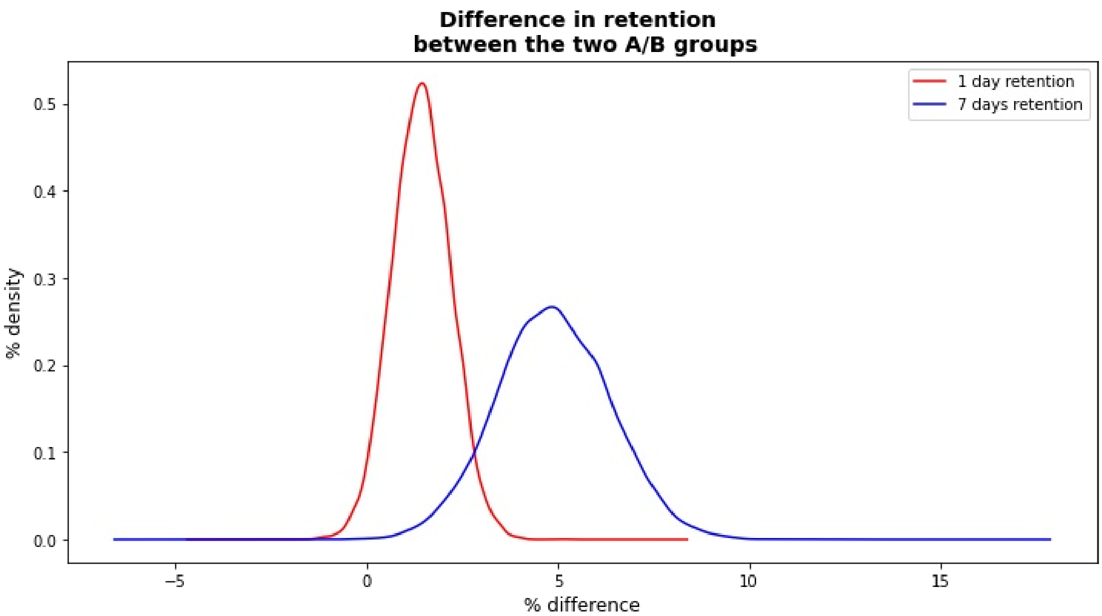
```
# Creating an list with bootstrapped means for each A/B group
bootstrap_1d = []
bootstrap_7d = []
for i in range(10000):
    bootstrap_mean_1 = cleaned_df.sample(frac=1, replace=True).groupby('version')['retention_1'].mean()
    bootstrap_mean_7 = cleaned_df.sample(frac=1, replace=True).groupby('version')['retention_7'].mean()
    bootstrap_1d.append(bootstrap_mean_1)
    bootstrap_7d.append(bootstrap_mean_7)

# Transforming the list to a DataFrame
```

```
bootstrap_1d = pd.DataFrame(bootstrap_1d)
bootstrap_7d = pd.DataFrame(bootstrap_7d)

# Kernel Density Estimate plot of the bootstrap distributions
fig, (ax1,ax2) = plt.subplots(1, 2, sharey=True, figsize=(12,6))

bootstrap_1d.plot.kde(ax=ax1)
ax1.set_xlabel("retention rate",size=12)
ax1.set_ylabel("number of sample",size=12)
ax1.set_title("1 day retention rate distribution", fontweight="bold",size=14)

bootstrap_7d.plot.kde(ax=ax2)
ax2.set_xlabel("retention rate",size=12)
ax2.set_title("7 days retention rate distribution", fontweight="bold",size=14)
plt.show()
```



In [17]:

```
# Adding a column with the % difference between the two A/B groups
bootstrap_1d['diff'] = ((bootstrap_1d['gate_30'] - bootstrap_1d['gate_40']) / bootstrap_1d['gate_40'] * 100)
bootstrap_7d['diff'] = ((bootstrap_7d['gate_30'] - bootstrap_7d['gate_40']) / bootstrap_7d['gate_40'] * 100)

# Ploting the bootstrap % difference
fig, (ax1) = plt.subplots(1, 1,figsize=(12,6))

bootstrap_1d['diff'].plot.kde(ax=ax1, c="#ff0000", label = "1 day retention")
bootstrap_7d['diff'].plot.kde(ax=ax1, c= "#0000ff", label = "7 days retention")
ax1.set_xlabel("% difference",size=12)
ax1.set_ylabel("% density",size=12)
ax1.set_title("Difference in retention \n between the two A/B groups", fontweight="bold", size=14)
plt.legend()
plt.show()
```



In [18]:

```
bootstrap_1d
```

Out[18]:

version	gate_30	gate_40	diff
retention_1	0.443179	0.439184	0.909608
retention_1	0.446777	0.442371	0.996085
retention_1	0.442900	0.439342	0.809806
retention_1	0.447738	0.441770	1.350916
retention_1	0.443079	0.437076	1.373519
...	...	...	...
retention_1	0.442849	0.438599	0.969133
retention_1	0.444350	0.438194	1.404773
retention_1	0.441758	0.441670	0.019797
retention_1	0.445581	0.441487	0.927201
retention_1	0.451293	0.439842	2.603357

10000 rows × 3 columns

In [19]:

```
bootstrap_1d.describe()
```

Out[19]:

version	gate_30	gate_40	diff
count	10000.000000	10000.000000	10000.000000
mean	0.445962	0.439651	1.438259
std	0.002322	0.002338	0.757838
min	0.436879	0.430264	-1.415694
25%	0.444385	0.438073	0.915234
50%	0.445968	0.439665	1.431624
75%	0.447495	0.441224	1.955120
max	0.454667	0.449987	5.116198

In [20]:

```
bootstrap_7d
```

Out[20]:

version	gate_30	gate_40	diff
retention_7	0.186949	0.179615	4.083073
retention_7	0.188025	0.176011	6.826267
retention_7	0.188397	0.175596	7.290346
retention_7	0.186262	0.174898	6.497665
retention_7	0.184315	0.177130	4.056256
...	...	...	...
retention_7	0.189575	0.173971	8.969294
retention_7	0.186234	0.177759	4.767948
retention_7	0.183792	0.178183	3.147623
retention_7	0.190032	0.180821	5.094224
retention_7	0.186643	0.173726	7.435178

10000 rows × 3 columns

In [21]:

```
bootstrap_7d.describe()
```

Out[21]:

version	gate_30	gate_40	diff
count	10000.000000	10000.000000	10000.000000
mean	0.186750	0.178074	4.882784
std	0.001844	0.001792	1.480396
min	0.178840	0.170932	-0.475870
25%	0.185476	0.176857	3.874697

50%	0.186752	0.178067	4.856484
75%	0.187982	0.179276	5.891354
max	0.193080	0.185167	11.753484

Conclusion

In [22]:

```
# Probability of 1-day retention is greater when the gate is at level 30
probability_1 = (bootstrap_1d['diff']>0).sum()/len(bootstrap_1d['diff'])

# Calculating of 7-days retention is greater when the gate is at level 30
probability_7 = (bootstrap_7d['diff']>0).sum()/len(bootstrap_7d['diff'])


print(f"The probability of 1-day retention is greater when the gate is at level 30: {(probability_1)*100}% \
      \nThe probability of 7-days retention is greater when the gate is at level 30: {(probability_7)*100}% ")
```

```
The probability of 1-day retention is greater when the gate is at level 30: 97.41%
The probability of 7-days retention is greater when the gate is at level 30: 99.96000000000001%
```

The likelihood of observing a higher 1-day retention when the gate is at level 30 stands at 96.88%. Correspondingly, the probability of a superior 7-day retention with the gate at level 30 is estimated at 99.97%.

As per the bootstrap analysis, there exists compelling evidence, accounting for 99.97% certainty, that 7-day retention tends to be higher when the gate is positioned at level 30 compared to level 40.

In summary, the inference drawn from these findings suggests that for maintaining high retention rates, both for 1-day and 7-day spans, it would be advisable to retain the gate at level 30 rather than shifting it to level 40. It's worth noting that while other metrics such as the quantity of game rounds played or the in-game purchase amounts of the two AB-groups may be considered, retention remains a pivotal metric.

In [ ]:

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js