In [126…
```
import pandas as pd
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sb
from scipy.stats import ttest_ind
from math import ceil
import math
activity=pd.read_csv('public_activity_export_2023-12-21_111702.csv')
#print (activity)
```

In [127…
```
groups=pd.read_csv('public_groups_export_2023-12-21_111722.csv')
#print (groups)
```

In [128…
```
users=pd.read_csv('public_users_export_2023-12-21_111733.csv')
#print (users)
```

In [93]:
```
#What is the average amount spent per user for the control and treatment groups? This
merged_df = pd.merge(groups, activity, on='uid',how='outer')
```

In [129…
```
merged_df['spent'].fillna(0, inplace=True)
#merged_df.info()
```

In [95]:
```
duplicate_uid = merged_df[merged_df.duplicated(subset='uid')]

# Count the number of duplicate 'uid' values
num_duplicate_uid = len(duplicate_uid)

print(f"There are {num_duplicate_uid} users that appear mutliple times in the dataset.
```

```
There are 139 users that appear mutliple times in the dataset.
```

In [96]:
```
clean_df = merged_df = merged_df.drop(merged_df[merged_df.duplicated(subset='uid')].ir
print(clean_df.info())
```

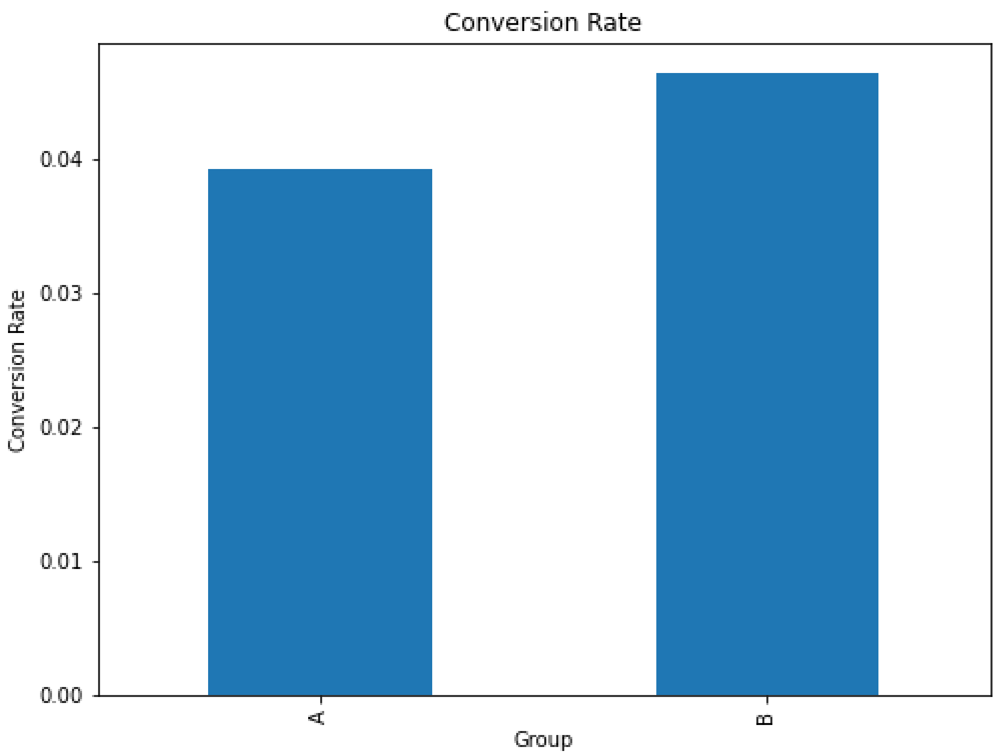
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48943 entries, 0 to 49081
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   uid       48943 non-null  int64  
 1   group     48943 non-null  object 
 2   join_dt   48943 non-null  object 
 3   device_x  48649 non-null  object 
 4   dt        2094 non-null   object 
 5   device_y  2085 non-null   object 
 6   spent     48943 non-null  float64
dtypes: float64(1), int64(1), object(5)
memory usage: 3.0+ MB
None
```

In [97]:
```
conversion_rates = clean_df.groupby('group')['spent'].apply(lambda x: (x > 0).mean())
print(conversion_rates)

# Calculating standard deviation.
std_deviation = clean_df.groupby('group')['spent'].std()
print(std_deviation)
```

```
# Calculating standard of error.
std_error = clean_df.groupby('group')['spent'].sem()
print(std_error)
```

```
group
A    0.039231
B    0.046301
Name: spent, dtype: float64
group
A    25.252823
B    24.633209
Name: spent, dtype: float64
group
A    0.161854
B    0.157056
Name: spent, dtype: float64
```

In [98]:
```
plt.figure(figsize=(8,6))
conversion_rates.plot(kind='bar')
plt.title('Conversion Rate')
plt.xlabel('Group')
plt.ylabel('Conversion Rate')
plt.savefig("conversionrate.png")
plt.show()
```



In [99]:
```
#Identifying how many users convert in each group

converted = clean_df[clean_df['spent'] > 0]
num_converted = converted.groupby('group')['uid'].count()
print(num_converted)

# Number of sample in each group
clean_df.groupby('group')['uid'].count()
```

```
group
A     955
B    1139
Name: uid, dtype: int64
```

Out[99]:
```
group
A    24343
B    24600
Name: uid, dtype: int64
```

In [100…

```python
# assigning each value to a variable to make the calculations easier.
x_control = 955 # number of users that made a purchase
N_control = 24343 # sample size for control group
p_hat_control = x_control / N_control # This is the proportion or convertion rate for
z_score_A = 1.96

#Standard error
SE_A = math.sqrt(p_hat_control * (1- p_hat_control) / N_control)

#Confidence level for control group

ci_A = (p_hat_control - z_score_A * SE_A, p_hat_control + z_score_A *  SE_A)

print(f"Conversion rate for control group:", np.round(p_hat_control,4))
print("Control group confidence Interval: ({:.4f}, {:.4f})".format(ci_A[0], ci_A[1]))
```

```
Conversion rate for control group: 0.0392
Control group confidence Interval: (0.0368, 0.0417)
```

In [101…

```python
x_treat = 1139 # number of users that made a purchase in the treatment group
N_treat = 24600 # sample size for treatment group
p_hat_treatment = x_treat / N_treat  # This is the proportion or convertion rate for t
z_score_B = 1.96

#Standard error
SE_B = math.sqrt(p_hat_treatment * (1- p_hat_treatment) / N_treat)

#Confidence level for Treatment group

ci_B = (p_hat_treatment - z_score_B * SE_B, p_hat_treatment + z_score_B *  SE_B)

print(f"Conversion rate treatment group:", np.round(p_hat_treatment,4))
print("Treatment group confidence Interval: ({:.4f}, {:.4f})".format(ci_B[0], ci_B[1])
```

```
Conversion rate treatment group: 0.0463
Treatment group confidence Interval: (0.0437, 0.0489)
```

In [107…

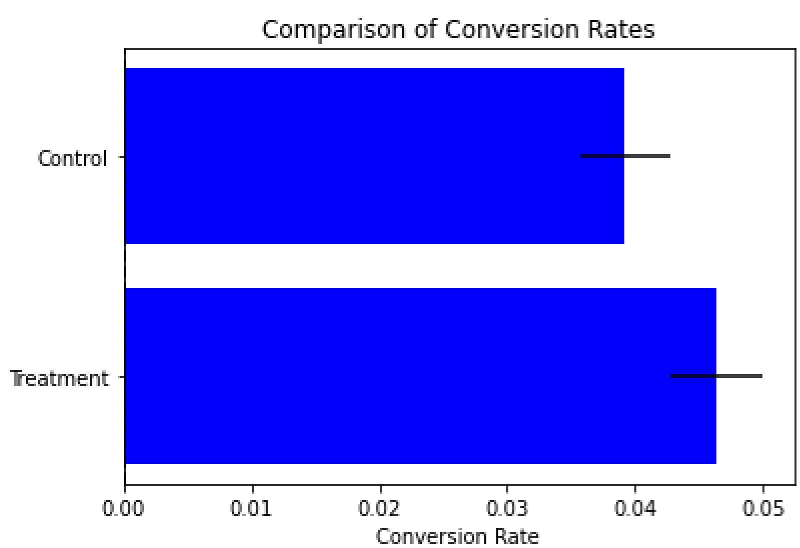
```python
from scipy.stats import norm

#Significance level
alpha = 0.05

# Calculate the proportion of the difference.
p_pooled = (x_control + x_treat) / (N_control + N_treat)
pooled_variance = p_pooled * (1-p_pooled) * (1/N_control + 1/N_treat) # Variance of th

#Standard of error
SE = np.sqrt(pooled_variance)

#Test statistists
test_stat = (p_hat_treatment - p_hat_control)/ SE
```

```
# Critical value or z-score using the normal distribution
z_score = norm.ppf(1-alpha /2)

# Calculating the margin of error
ME = SE * z_score

#Calculating p-value
p_value = norm.sf(test_stat)*2 # We multiply here for 2 because we are using two sidec

# Calculating confidence intervals
CI = [(p_hat_treatment - p_hat_control) - SE * z_score, (p_hat_treatment - p_hat_contr

if np.abs(test_stat) >= z_score:
    print("reject the null hypothesis")
    print("p-value:",np.round(p_value,4))

print("Test statistics stat: ", np.round(test_stat,4))
print("Z-Critical score: ", np.round(z_score,2))
print("P_value: ", np.round(p_value,4))
print("Confidence Interval of 2 sample proportion: " , np.round(CI,4))
```

```
reject the null hypothesis
p-value: 0.0001
Test statistics stat:  3.8643
Z-Critical score:  1.96
P_value:  0.0001
Confidence Interval of 2 sample proportion:  [0.0035 0.0107]
```

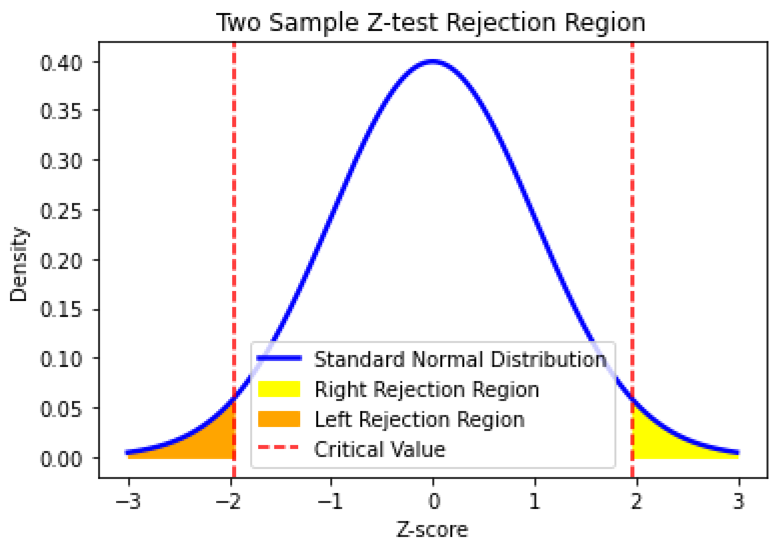
In [108…

```
groups = ['Control', 'Treatment']
conversion_rates = [p_hat_control, p_hat_treatment]
errors = [ME, ME]
fig, ax = plt.subplots()
y_pos = np.arange(len(groups))
ax.barh(y_pos, conversion_rates, xerr=errors, align='center', color='blue')
ax.set_yticks(y_pos)
ax.set_yticklabels(groups)
ax.invert_yaxis()
ax.set_xlabel('Conversion Rate')
ax.set_title('Comparison of Conversion Rates')
plt.axvline(x=0, color='black', linestyle='--')
plt.show()
```



In [111…

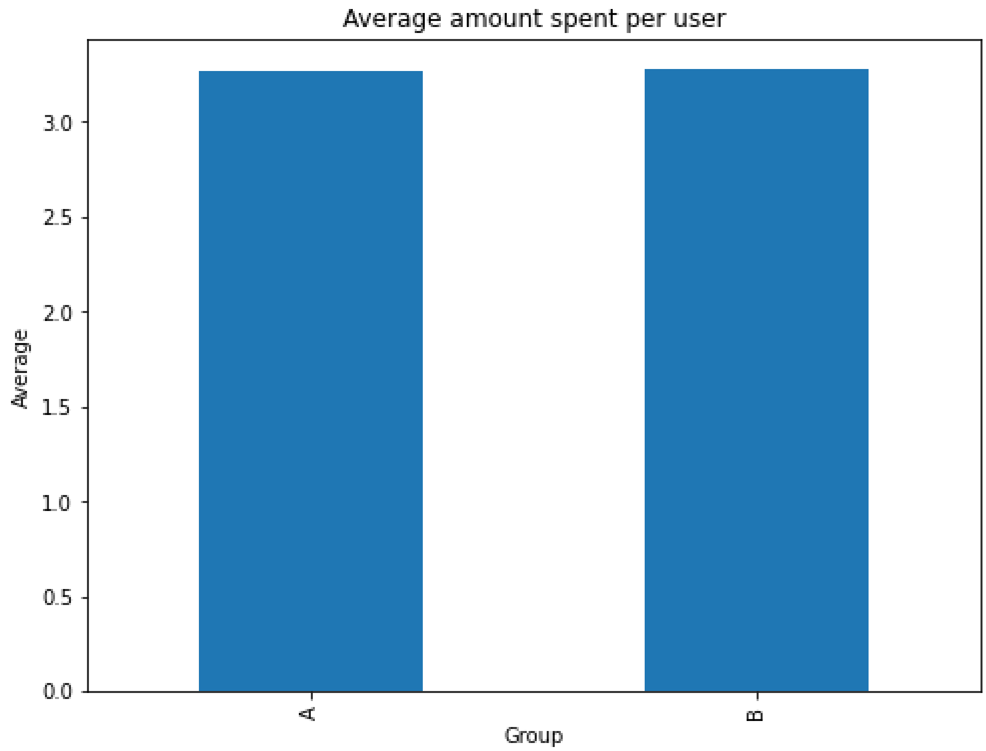
```
z = np.arange(-3, 3, 0.01)
plt.plot(z, norm.pdf(z), label='Standard Normal Distribution', color='blue', linewidth
plt.fill_between(z[z > z_score], norm.pdf(z[z > z_score]), label='Right Rejection Regi
plt.fill_between(z[z < -z_score], norm.pdf(z[z < -z_score]), label='Left Rejection Reg
plt.axvline(x=z_score, color='red', linestyle='--', label='Critical Value')
plt.axvline(x=-z_score, color='red', linestyle='--')
plt.title("Two Sample Z-test Rejection Region")
plt.xlabel('Z-score')
plt.ylabel('Density')
plt.legend()
plt.show()
```



In [118…

```
# calculating the average amount spent per user in each group. In this calculation I u
# In the proportion since we want to analyze which users actually converte, it was nec
amount_spent_user = clean_df.groupby(['group','uid'])['spent'].sum().groupby('group').
print(amount_spent_user)

plt.figure(figsize=(8,6))
amount_spent_user.plot(kind='bar')
plt.title('Average amount spent per user')
plt.xlabel('Group')
plt.ylabel('Average')
plt.savefig("average amount spent per user.png")
plt.show()
```

```
group
A    3.271324
B    3.275629
Name: spent, dtype: float64
```



In [120…

```
from scipy.stats import t

control = clean_df[clean_df['group']=='A']

#Calculating some statistics over control group

NC= control.shape[0]
sample_mean = control['spent'].mean()
sample_std = control['spent'].std()

#Calculating the standard error of the mean
SOE = sample_std / np.sqrt(NC)

#Claculating the t-critical value, in this case 95% confidence level
t_critical = t.ppf(0.975, NC-1)

#Margin of error
MOE = t_critical * SOE

#Confidence interval

CIC = (sample_mean - MOE, sample_mean + MOE)
print(f"Sample mean; {sample_mean:.2f}")
print(f"Sample standard deviation : {sample_std:.2f}")
print(f"Standard of error: {SOE:.2f}")
print(f"t-critical value: {t_critical:.2f}")
print(f"Margin of error: {MOE:.2f}")
print(f"95% Confidence Interval: {np.round(CIC, 3)}")
```

```
Sample mean; 3.27
Sample standard deviation : 25.25
Standard of error: 0.16
t-critical value: 1.96
Margin of error: 0.32
95% Confidence Interval: [2.954 3.589]
```

In [123…

```
#Subsetting the data to incluide only the control group
treatment = clean_df[clean_df['group']=='B']

#Calculating some statistics over control group

NT= treatment.shape[0]
sample_mean_t = treatment['spent'].mean()
sample_std_t = treatment['spent'].std()

#Calculating the standard error of the mean
SET = sample_std_t / np.sqrt(NT)

#Claculating the t-critical value, in this case 95% confidence level
t_critical_t = t.ppf(0.975, NT-1)

#Margin of error
MET = t_critical * SET

#Confidence interval

CIT = (sample_mean_t - MET, sample_mean_t + MET)

print(f"Sample mean: {sample_mean_t:.3f}")
print(f"Sample standard deviation : {sample_std_t:.2f}")
print(f"Standard of error: {SET:.2f}")
print(f"t-critical value: {t_critical_t:.2f}")
print(f"Margin of error: {MET:.2f}")
print(f"95% Confidence Interval: {np.round(CIT, 3)}")
```

```
Sample mean: 3.276
Sample standard deviation : 24.63
Standard of error: 0.16
t-critical value: 1.96
Margin of error: 0.31
95% Confidence Interval: [2.968 3.583]
```

In [125…

```
# Calculating the confidence interval for the difference of mean using unequal varianc

diff = sample_mean_t - sample_mean

#standard error of the difference
se_diff = np.sqrt((sample_std**2/len(control)) + (sample_std_t**2/len(treatment)))

#t-statistics and p-value
t_stat, p_value = ttest_ind(treatment['spent'], control['spent'], equal_var =False)

#Confidence interal for the difference
ci_low,ci_high = diff - 1.96 * se_diff,diff + 1.96 * se_diff

print(f"Difference in mean: {diff:.3f}")
print(F"95% Confidence Interval: {np.round(ci_low,3), np.round(ci_high, 3)}")
print(f"p-value:  {p_value: .3f}")
```

```
Difference in mean: 0.004
95% Confidence Interval: (-0.438, 0.446)
p-value:   0.985
```

In [ ]: