Marketing_AB_Testing

October 7, 2024

1 Marketing A/B testing dataset

Marketing companies want to run successful campaigns, but the market is complex and several options can work. So normally they run A/B tests, that is a randomized experimentation process wherein two or more versions of a variable (web page, page element, banner, etc.) are shown to different segments of people at the same time to determine which version leaves the maximum impact and drive business metrics.

-The companies are interested in answering two questions:

- Would the campaign be successful?
- If the campaign was successful, how much of that success could be attributed to the ads?

With the second question in mind, we normally do an A/B test. The majority of the people will be exposed to ads (the experimental group). And a small portion of people (the control group) would instead see a Public Service Announcement (PSA) (or nothing) in the exact size and place the ad would normally be.

The idea of the dataset is to analyze the groups, find if the ads were successful, how much the company can make from the ads, and if the difference between the groups is statistically significant.

-Data dictionary:

- Index: Row index
- user id: User ID (unique)
- test group: If "ad" the person saw the advertisement, if "psa" they only saw the public service announcement
- converted: If a person bought the product then True, else is False
- total ads: Amount of ads seen by person
- most ads day: Day that the person saw the biggest amount of ads
- most ads hour: Hour of day that the person saw the biggest amount of ads

2 EDA

[1]:
```
import pandas as pd
import numpy as np
import scipy.stats as stats
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
```

```
from scipy.stats import chi2_contingency
import seaborn as sns
from scipy.stats import ttest_ind
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import kruskal
import statsmodels.api as sm
import pylab as py
from scipy.stats import shapiro
from statsmodels.stats.multicomp import pairwise_tukeyhsd
from scipy.stats import kruskal
from scipy.stats import levene
from statsmodels.stats.power import NormalIndPower
from scipy.stats import norm
```

[2]:
```
df = pd.read_csv('marketing_AB.csv')
df.head(3)
```

[2]:
```
   Unnamed: 0  user id test group  converted  total ads most ads day  \
0           0  1069124         ad      False        130       Monday
1           1  1119715         ad      False         93      Tuesday
2           2  1144181         ad      False         21      Tuesday

   most ads hour
0             20
1             22
2             18
```

[3]:
```
df.shape
```

[3]:
```
(588101, 7)
```

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 588101 entries, 0 to 588100
Data columns (total 7 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Unnamed: 0     588101 non-null  int64
 1   user id        588101 non-null  int64
 2   test group     588101 non-null  object
 3   converted      588101 non-null  bool
 4   total ads      588101 non-null  int64
 5   most ads day   588101 non-null  object
 6   most ads hour  588101 non-null  int64
dtypes: bool(1), int64(4), object(2)
memory usage: 27.5+ MB
```

[5]:
```
# check null values
df.isna().sum()
```

[5]:
```
Unnamed: 0       0
user id          0
test group       0
converted        0
total ads        0
most ads day     0
most ads hour    0
dtype: int64
```

[50]:
```
duplicates = df[df.duplicated('user id',keep=False)]
duplicates
```

[50]:
```
Empty DataFrame
Columns: [Unnamed: 0, user id, test group, converted, total ads, most ads day,
most ads hour]
Index: []
```

[51]:
```
# calculate Q1 (25th percentile) and (75th percentile)
numeric_cols = df.select_dtypes(include=['number'])
Q1 = numeric_cols.quantile(0.25)
Q3 = numeric_cols.quantile(0.75)
IQR = Q3-Q1

# define outliers as points outside 1.5*IQR range
outliers=(numeric_cols < (Q1-1.5*IQR)) | (numeric_cols > (Q3+1.5*IQR))

# print outliers for each column
outliers_count = outliers.sum()
print('no of outliers in each column')
print(outliers_count)

# optionally fiter out the outliers
df_outliers = numeric_cols[outliers.any(axis=1)]
df_outliers
```

```
no of outliers in each column
Unnamed: 0           0
user id              0
total ads        52057
most ads hour     5536
dtype: int64
```

[51]:

	Unnamed: 0	user id	total ads	most ads hour
0	0	1069124	130	20
1	1	1119715	93	22

```
3                    3  1435133          355               10
4                    4  1015700          276               14
5                    5  1137664          734               10
…               …       …           …             …
584639       584639  1004190            1                0
584640       584640  1028589            1                0
584641       584641  1536866            1                0
584642       584642  1089798            1                0
584643       584643  1096523            1                0

[57135 rows x 4 columns]
```

[7]:

```
cont_var = ['total ads','most ads hour']
cat_var = ['test group','converted','most ads day']
```

Frequency Table and Mode for Categorical Variables

Frequency tables organize and display the counts and percentages of different categories in a dataset, offering a snapshot of categorical data distribution. Meanwhile, the mode pinpoints the most frequently occurring category, serving as a quick reference for the dominant aspect of the data. Together, they provide a concise yet insightful analysis, aiding in identifying patterns and making informed interpretations of categorical variables.

[8]:

```
def frequency_table(variable):

    # Get unique elements and their counts
    unique_elements, counts = np.unique(variable.dropna(), return_counts=True)

    # Calculate percentages
    percentages = (counts / len(variable)) * 100

    # Create a dictionary to store the value counts and percentages
    value_counts_and_percentages = zip(unique_elements, counts, percentages)

    # Print the value counts and percentages
    for i, j, k in value_counts_and_percentages:
        print(f"{i}: Count: {j}, Percentage: {k:.2f}%")
    return


# Calculate frequency table and mode for each categorical variable
for var in cat_var:
    print (f"frequency table for {var}")
    frequency_table(df[var])
    print("Mode =", df[var].mode()[0])
    print ("#"*50)
```

```
frequency table for test group
ad: Count: 564577, Percentage: 96.00%
```

```
psa: Count: 23524, Percentage: 4.00%
Mode = ad
##################################################
frequency table for converted
False: Count: 573258, Percentage: 97.48%
True: Count: 14843, Percentage: 2.52%
Mode = False
##################################################
frequency table for most ads day
Friday: Count: 92608, Percentage: 15.75%
Monday: Count: 87073, Percentage: 14.81%
Saturday: Count: 81660, Percentage: 13.89%
Sunday: Count: 85391, Percentage: 14.52%
Thursday: Count: 82982, Percentage: 14.11%
Tuesday: Count: 77479, Percentage: 13.17%
Wednesday: Count: 80908, Percentage: 13.76%
Mode = Friday
##################################################
```

[54]:

```
data.head(2)
```

[54]:

```
   Unnamed: 0  user id test group  converted  total ads most ads day  \
0           0  1069124         ad      False        130       Monday
1           1  1119715         ad      False         93      Tuesday

   most ads hour
0             20
1             22
```

Advanced Statistics for Continuous Variables

Basic statistics like mean and standard deviation offer a fundamental grasp of continuous variables. Going beyond, skewness and kurtosis provide advanced insights. Skewness indicates the distribution's asymmetry, with positive values suggesting a longer right tail and negative values indicating a longer left tail. Kurtosis measures the distribution's peakedness, with higher values indicating a more peaked shape. These advanced statistics add depth to understanding the nuances of continuous variables, offering a comprehensive view of distributional characteristics beyond basic measures.

[55]:

```
concatenated_series = pd.concat([
    df[cont_var].describe().T,
    df[cont_var].skew().rename('skewness'),
    df[cont_var].kurtosis().rename('kurtosis')
], axis=1)


# Adding lower and upper confidence intervals
confidence_level = 0.95  # 95% confidence interval
```

```
for var in cont_var:
    values = df[var].dropna()
    mean = values.mean()
    std_error = stats.sem(values)
    if std_error != 0:
        lower, upper = stats.t.interval(confidence_level, len(values) - 1,␣
↪loc=mean, scale=std_error)
    else:
        lower, upper = mean, mean

    # Adding lower and upper confidence intervals to the concatenated series
    concatenated_series.loc[var, 'lower_ci'] = lower
    concatenated_series.loc[var, 'upper_ci'] = upper
concatenated_series
```

[55]:

```
                   count       mean        std  min   25%   50%   75%     max  \
total ads       588101.0  24.820876  43.715181  1.0   4.0  13.0  27.0  2065.0
most ads hour   588101.0  14.469061   4.834634  0.0  11.0  14.0  18.0    23.0

                skewness   kurtosis   lower_ci   upper_ci
total ads       7.433113 109.917983  24.709150  24.932602
most ads hour  -0.336972   0.103237  14.456704  14.481417
```

[56]:

```
import pandas as pd

# Function to calculate summary statistics and other metrics for a given column
def calculate_summary_statistics(data, column_name):
    desc = data[column_name].describe()

    # Skewness and kurtosis
    skewness = data[column_name].skew()
    kurtosis = data[column_name].kurt()

    # Confidence interval (95%)
    mean = desc['mean']
    std = desc['std']
    count = desc['count']
    lower_ci = mean - 1.96 * (std / (count ** 0.5))
    upper_ci = mean + 1.96 * (std / (count ** 0.5))

    # Creating a DataFrame to store the results
    summary_df = pd.DataFrame({
        'count': [count],
        'mean': [mean],
        'std': [std],
        'min': [desc['min']],
        '25%': [desc['25%']],
```

```
        '50%': [desc['50%']],
        '75%': [desc['75%']],
        'max': [desc['max']],
        'skewness': [skewness],
        'kurtosis': [kurtosis],
        'lower_ci': [lower_ci],
        'upper_ci': [upper_ci]
    })

    return summary_df

# Example usage with 'total ads' and 'most ads hour' columns
total_ads_summary = calculate_summary_statistics(df, 'total ads')
most_ads_hour_summary = calculate_summary_statistics(df, 'most ads hour')
```

[57]:
```
# Display the results
print("Total Ads Summary:")
total_ads_summary
```

```
Total Ads Summary:
```

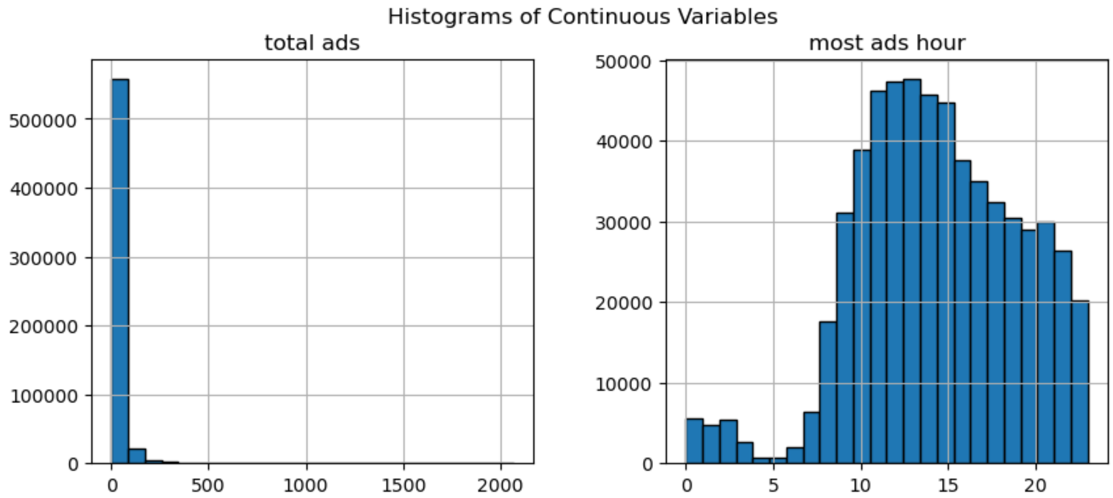
[57]:
```
      count       mean        std  min  25%   50%   75%     max  skewness  \
0  588101.0  24.820876  43.715181  1.0  4.0  13.0  27.0  2065.0  7.433113

     kurtosis   lower_ci   upper_ci
0  109.917983  24.709148  24.932604
```

[58]:
```
print("\nMost Ads Hour Summary:")
most_ads_hour_summary
```

```
Most Ads Hour Summary:
```

[58]:
```
      count       mean       std  min   25%   50%   75%   max  skewness  \
0  588101.0  14.469061  4.834634  0.0  11.0  14.0  18.0  23.0 -0.336972

   kurtosis   lower_ci   upper_ci
0  0.103237  14.456704  14.481417
```

[59]:
```
cont_var = ['total ads', 'most ads hour']

# Plot histograms for each continuous variable
df[cont_var].hist(bins=24, figsize=(10, 4), edgecolor='black')

plt.suptitle('Histograms of Continuous Variables')
plt.show()
```



[60]:
```
df.head(2)
```

[60]:
```
   Unnamed: 0  user id test group  converted  total ads most ads day  \
0           0  1069124         ad      False        130       Monday
1           1  1119715         ad      False         93      Tuesday

   most ads hour
0             20
1             22
```

3 Lets check is there relationship between converted and most ads day

Chi-Square Test for Independence - coverted and most ads day

[37]:
```
from scipy.stats import norm

def OR_CIs(contingency_table):

    # Calculate odds ratio
    odds_ratio = (contingency_table.iloc[0, 0] / contingency_table.iloc[0, 1]) /
 ↪ (contingency_table.iloc[1, 0] / contingency_table.iloc[1, 1])

    # Calculate standard error of log(odds ratio)
    log_odds_std_error = np.sqrt(contingency_table.applymap(lambda x: 1/x).
 ↪sum().sum())

    # Set confidence level
    confidence_level = 0.95
```

```
    # Calculate z-score for the confidence interval
    z_score = norm.ppf(1-(1 - confidence_level) / 2)

    # Calculate confidence intervals
    ci_low = np.exp(np.log(odds_ratio) - z_score * log_odds_std_error)
    ci_high = np.exp(np.log(odds_ratio) + z_score * log_odds_std_error)

    # Print the results
    print(f"Odds Ratio: {odds_ratio:.2f}")
    print(f"95% Confidence Interval: {ci_low:.2f}, {ci_high:.2f}")

    return
```

[69]:
```
# Create a contingency table
contingency_table = pd.crosstab(df['converted'], df['most ads day'])

print("Contingency Table with Frequencies:")
display(contingency_table)
print("#"*60)

# Calculate row percentages
row_percentages = contingency_table.div(contingency_table.sum(axis=1), axis=0)␣
 ↪* 100

print("\nRow Percentages:")
display(row_percentages)
print("#"*60)

# Perform chi-square test
chi2, p, dof, expected = chi2_contingency(contingency_table)

print(f"\nChi-squared value: {chi2}")
print(f"P-value: {p}")
print(f"Degrees of freedom: {dof}")
print("#"*60)

# Calculate the percentage of cells with expected counts less than 5
percentage_low_expected = (expected < 5).sum().sum() / (expected.shape[0] *␣
 ↪expected.shape[1]) * 100

print(f"Percentage of cells with expected counts less than 5:␣
 ↪{percentage_low_expected:.2f}%")
print("#"*60)

# Calculate residuals (observed minus expected values)
residuals = contingency_table - expected
```

```python
print("\nResiduals (Observed - Expected):")
display(residuals)
print("#"*60)

# Calculate odds ratio
OR_CIs(contingency_table)
```

```
Contingency Table with Frequencies:

most ads day  Friday  Monday  Saturday  Sunday  Thursday  Tuesday  Wednesday
converted
False          90551   84216     79941   83301     81192    75167      78890
True            2057    2857      1719    2090      1790     2312       2018

############################################################

Row Percentages:

most ads day     Friday     Monday   Saturday     Sunday   Thursday  \
converted
False         15.795855  14.690768  13.945030  14.531154  14.163256
True          13.858384  19.248130  11.581217  14.080711  12.059557

most ads day    Tuesday  Wednesday
converted
False         13.112246  13.761692
True          15.576366  13.595634

############################################################

Chi-squared value: 410.0478857936585
P-value: 1.932184379244731e-85
Degrees of freedom: 6
############################################################
Percentage of cells with expected counts less than 5: 0.00%
############################################################

Residuals (Observed - Expected):

most ads day      Friday      Monday    Saturday     Sunday    Thursday  \
converted
False         280.320535 -659.376566  342.005474  65.171668  304.371249
True         -280.320535  659.376566 -342.005474 -65.171668 -304.371249

most ads day     Tuesday  Wednesday
converted
False        -356.518209  24.025849
True          356.518209 -24.025849

############################################################
```

```
Odds Ratio: 1.49
95% Confidence Interval: 1.33, 1.68
```

[39]:
```
import pandas as pd

# Create a crosstab for 'converted' status and 'most ads day'
contingency_table = pd.crosstab(df['converted'], df['most ads day'])

# Display the contingency table with custom styling (for cleaner output)
styled_table = contingency_table.style.set_caption("Contingency Table with␣
↪Frequencies").background_gradient(cmap="Blues", axis=None)

# Display the styled table
styled_table
```

[39]:
```
<pandas.io.formats.style.Styler at 0x1cc54467850>
```

[ ]:
```
# darker color shows the friday has most not converted users or customers␣
↪compared to other days
```

4 Interpretation:

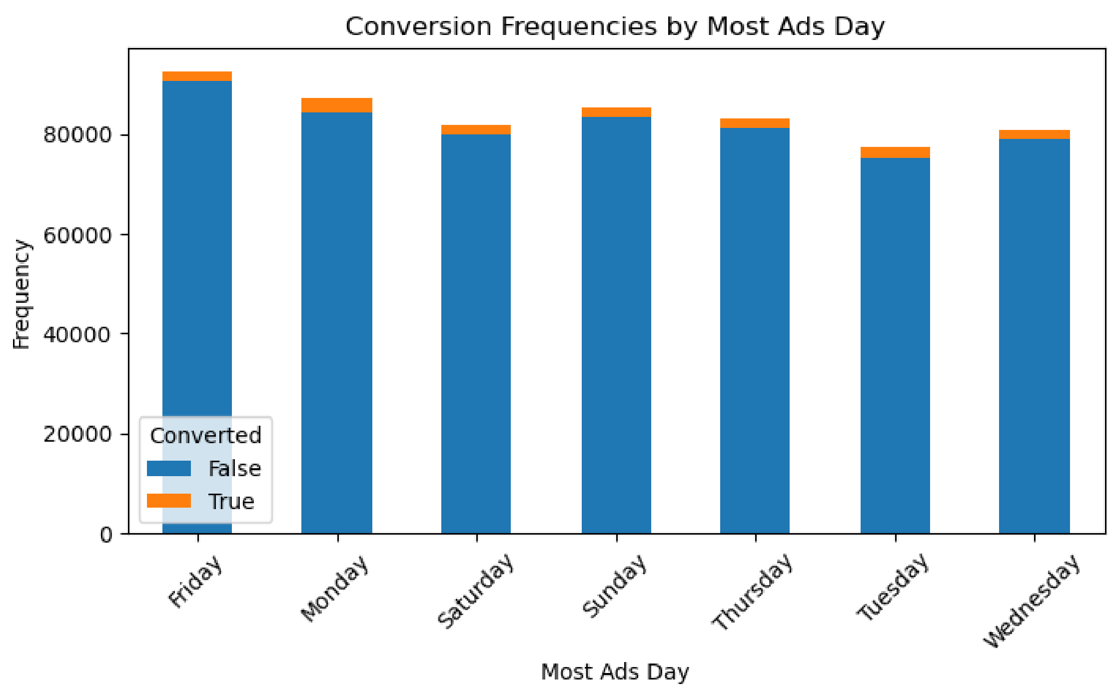
1. Contingency Table: This table shows the count of conversions and non-conversions (True/False) based on the most frequent day users saw ads. For example, on Friday, 90,551 users did not convert, and 2,057 users converted.
2. Row Percentages: These percentages show how conversions and non-conversions are distributed across different days. Among the users who did not convert, 15.80% saw ads mostly on Friday, while 13.11% saw ads mostly on Tuesday. Among users who converted, the largest percentage (19.25%) saw ads mostly on Monday, while the lowest percentage (11.58%) saw ads on Saturday.
3. Chi-Squared Test: Chi-squared value: 410.05 P-value: (1.93 ×10^{-85}) Degrees of freedom: 6 The very low p-value indicates a statistically significant association between the day users saw the most ads and their conversion status. This suggests that the day on which users saw the most ads had a meaningful impact on whether they converted or not.
4. Residuals: Positive residuals mean that the actual count is higher than the expected count, and negative residuals mean the actual count is lower than expected. For example: Monday (True): The positive residual (659.38) means more people converted on Monday than expected. Saturday (True): The negative residual (-342.01) indicates fewer people converted on Saturday than expected.
5. Odds Ratio: Odds ratio: 1.49 (Confidence Interval: 1.33 to 1.68) This means that users who saw ads most frequently on one day are 1.49 times more likely to convert compared to other users. The confidence interval indicates that this effect is likely to be true and not due to random chance.

Conclusion:

There is a strong association between the day users see the most ads and their conversion likelihood. Users seeing ads most frequently on Monday tend to convert more, while Saturday seems to have the least conversions. The overall odds of converting are increased for users based on the day they view ads the most frequently.

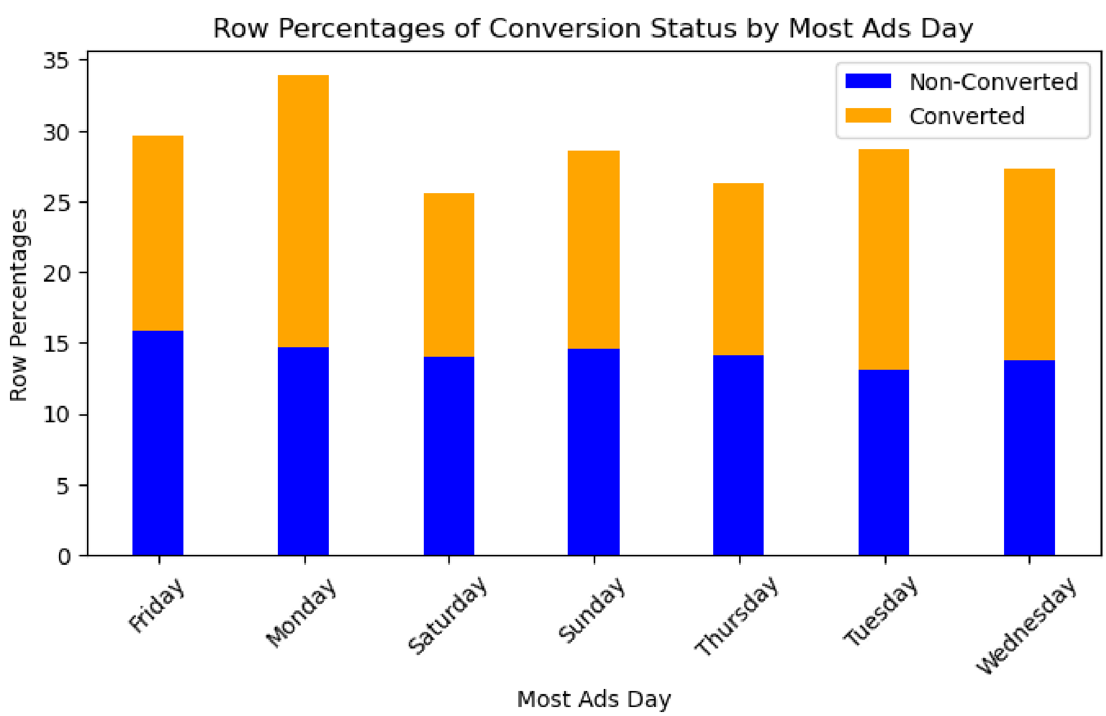
visulize Converted and most ads day

[48]:
```
# Data
categories = ['Friday', 'Monday', 'Saturday', 'Sunday', 'Thursday', 'Tuesday',␣
↪'Wednesday']
false_values = [90551, 84216, 79941, 83301, 81192, 75167, 78890]
true_values = [2057, 2857, 1719, 2090, 1790, 2312, 2018]

# Create a DataFrame
df2 = pd.DataFrame({
    'False': false_values,
    'True': true_values
}, index=categories)

# Plot
df2.plot(kind='bar', stacked=True, figsize=(8, 4))
plt.title('Conversion Frequencies by Most Ads Day')
plt.xlabel('Most Ads Day')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.legend(title='Converted')
plt.show()
```



[45]:

```
false_percentages = [15.795855, 14.690768, 13.945030, 14.531154, 14.163256, 13.
↪112246, 13.761692]
true_percentages = [13.858384, 19.248130, 11.581217, 14.080711, 12.059557, 15.
↪576366, 13.595634]

# Plot
fig, ax = plt.subplots(figsize=(8, 4))
width = 0.35  # Width of the bars
ax.bar(categories, false_percentages, width, label='Non-Converted',␣
↪color='blue')
ax.bar(categories, true_percentages, width, bottom=false_percentages,␣
↪label='Converted', color='orange')

ax.set_xlabel('Most Ads Day')
ax.set_ylabel('Row Percentages')
ax.set_title('Row Percentages of Conversion Status by Most Ads Day')
ax.legend()

plt.xticks(rotation=45)
plt.show()
```



5 Lets check is there relationship between converted and most ads hour

Chi-Square Test for Independence

- coverted and most ads hour

[61]:
```
df.head(2)
```

[61]:
```
   Unnamed: 0  user id test group  converted  total ads most ads day  \
0           0  1069124         ad      False        130       Monday
1           1  1119715         ad      False         93      Tuesday

   most ads hour
0             20
1             22
```

[65]:
```
df['most ads hour'].value_counts()
```

[65]:
```
13    47655
12    47298
11    46210
14    45648
15    44683
10    38939
16    37567
17    34988
18    32323
9     31004
19    30352
21    29976
20    28923
22    26432
23    20166
8     17627
7      6405
0      5536
2      5333
1      4802
3      2679
6      2068
5       765
4       722
Name: most ads hour, dtype: int64
```

[3]:
```
# Define a function to categorize hours into time slots
def categorize_hour(hour):
    if 5 <= hour < 12:
```

```
        return 'Morning'
    elif 12 <= hour < 17:
        return 'Afternoon'
    elif 17 <= hour < 21:
        return 'Evening'
    else:
        return 'Night'

# Apply the function to create a new column in the DataFrame
df['time_slot'] = df['most ads hour'].apply(categorize_hour)

# Display the updated DataFrame
df.head(3)
```

[3]:

```
   Unnamed: 0  user id test group  converted  total ads most ads day  \
0           0  1069124         ad      False        130       Monday
1           1  1119715         ad      False         93      Tuesday
2           2  1144181         ad      False         21      Tuesday

   most ads hour time_slot
0             20   Evening
1             22     Night
2             18   Evening
```

[82]:

```
# Create a contingency table
contingency_table_T = pd.crosstab(df['converted'], df['time_slot'])

print("Contingency Table with Frequencies:")
display(contingency_table_T)
print("#"*60)

# Calculate row percentages
row_percentages_T = contingency_table_T.div(contingency_table_T.sum(axis=1),␣
 ↪axis=0) * 100

print("\nRow Percentages:")
display(row_percentages_T)
print("#"*60)

# Perform chi-square test
chi2, p, dof, expected_T = chi2_contingency(contingency_table_T)

print(f"\nChi-squared value: {chi2}")
print(f"P-value: {p}")
print(f"Degrees of freedom: {dof}")
print("#"*60)
```

```python
# Calculate the percentage of cells with expected counts less than 5
percentage_low_expected_T = (expected < 5).sum().sum() / (expected.shape[0] *
    expected.shape[1]) * 100

print(f"Percentage of cells with expected counts less than 5:
    {percentage_low_expected_T:.2f}%")
print("#"*60)

# Calculate residuals (observed minus expected values)
residuals_T = contingency_table_T - expected_T

print("\nResiduals (Observed - Expected):")
display(residuals_T)
print("#"*60)

# Calculate odds ratio
OR_CIs(contingency_table_T)
```

```
Contingency Table with Frequencies:

time_slot  Afternoon  Evening  Morning  Night
converted
False         216786   123041   140041  93390
True            6065     3545     2977   2256

############################################################

Row Percentages:

time_slot  Afternoon    Evening    Morning      Night
converted
False      37.816481  21.463460  24.428966  16.291094
True       40.861012  23.883312  20.056592  15.199084

############################################################

Chi-squared value: 199.10145345082745
P-value: 6.596461961915453e-43
Degrees of freedom: 3
############################################################
Percentage of cells with expected counts less than 5: 0.00%
############################################################

Residuals (Observed - Expected):

time_slot   Afternoon     Evening    Morning       Night
converted
False     -440.494357 -350.113411  632.61157  157.996198
True       440.494357  350.113411 -632.61157 -157.996198
```

```
############################################################
Odds Ratio: 1.03
95% Confidence Interval: 0.96, 1.10
```

6 Interpretation of Insights:

Contingency Table with Frequencies:

1. False (Not Converted):

- The highest number of users who did not convert saw ads in the Afternoon (216,786 users).
- The Night time slot has the lowest number of users who did not convert (93,390 users).

2. True (Converted):

- The Afternoon time slot also has the highest number of conversions (6,065 users), followed by the Evening (3,545 users).
- The Night time slot has the lowest number of conversions (2,256 users).

Row Percentages:

1. For Non-conversions (False):

- The largest percentage of non-conversions occurred in the Afternoon (37.82%), followed by the Morning (24.43%).-
- The Night has the lowest percentage of non-conversions (16.29%).

2. For Conversions (True):

- Interestingly, the largest percentage of conversions also occurred in the Afternoon (40.86%), followed by the Evening (23.88%).
- The Night time slot again has the lowest percentage of conversions (15.20%).

Chi-square Test:

- Chi-squared value: 199.10 and p-value: 6.60e-43:
- The very low p-value (much lower than 0.05) indicates that the relationship between time slot and conversion status is statistically significant. In other words, the time slot does affect whether users convert or not.
- The degrees of freedom (3) refers to the number of categories (time slots) minus one.

Percentage of cells with expected counts less than 5: - 0%: This means that all the expected values are sufficiently large for the chi-square test to be reliable. There's no issue with low expected counts in this case.

Residuals (Observed - Expected): - Positive residuals mean more users converted in that time slot than expected, and negative residuals mean fewer users converted than expected. - Afternoon: More users converted than expected (440.49 for conversions, -440.49 for non-conversions). - Morning: Fewer users converted than expected (-632.61 for conversions, 632.61 for non-conversions), meaning it underperforms in terms of conversion compared to expectations. - Night: Slightly fewer users converted than expected (-157.99 for conversions).

The largest positive residuals for conversions were in the Afternoon, meaning that the Afternoon time slot outperformed expectations for conversions. Conversely, the Morning significantly underperformed in terms of conversions.

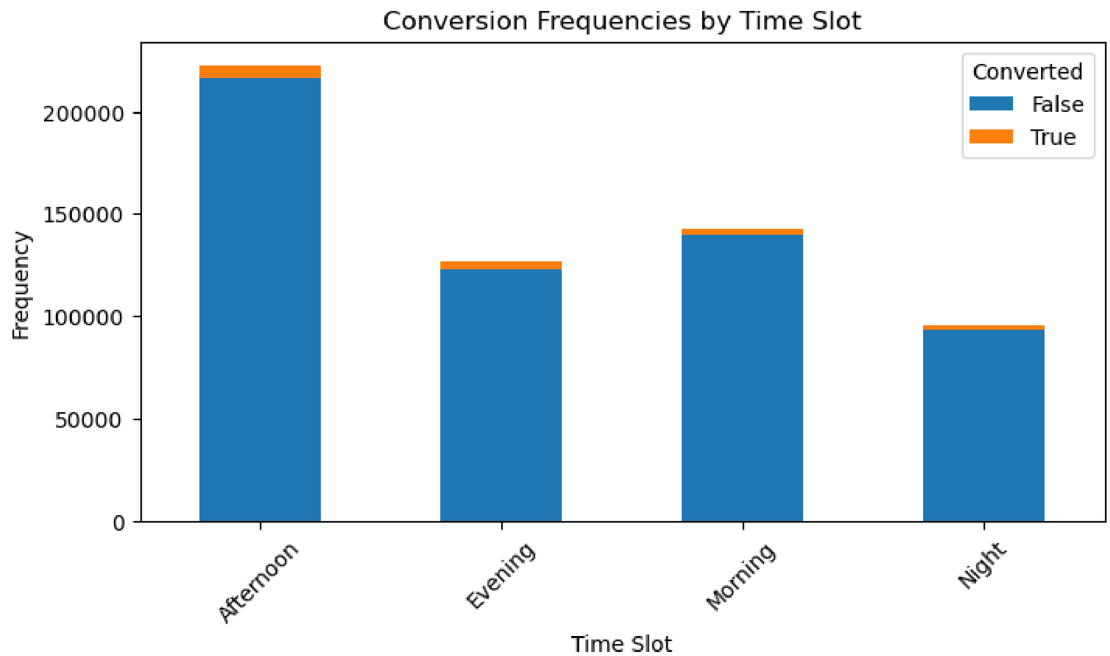
Odds Ratio: - Odds Ratio = 1.03 with a confidence interval of (0.96, 1.10): - The odds ratio is close to 1, which means that the difference between the time slots in terms of conversion rates is very small. This indicates that although the difference between time slots is statistically significant (because of the low p-value), the practical difference (effect size) is quite small. A 1.03 odds ratio suggests that one group is only 3% more likely to convert compared to another, which might not be a meaningful difference in practice.

Summary of Insights:

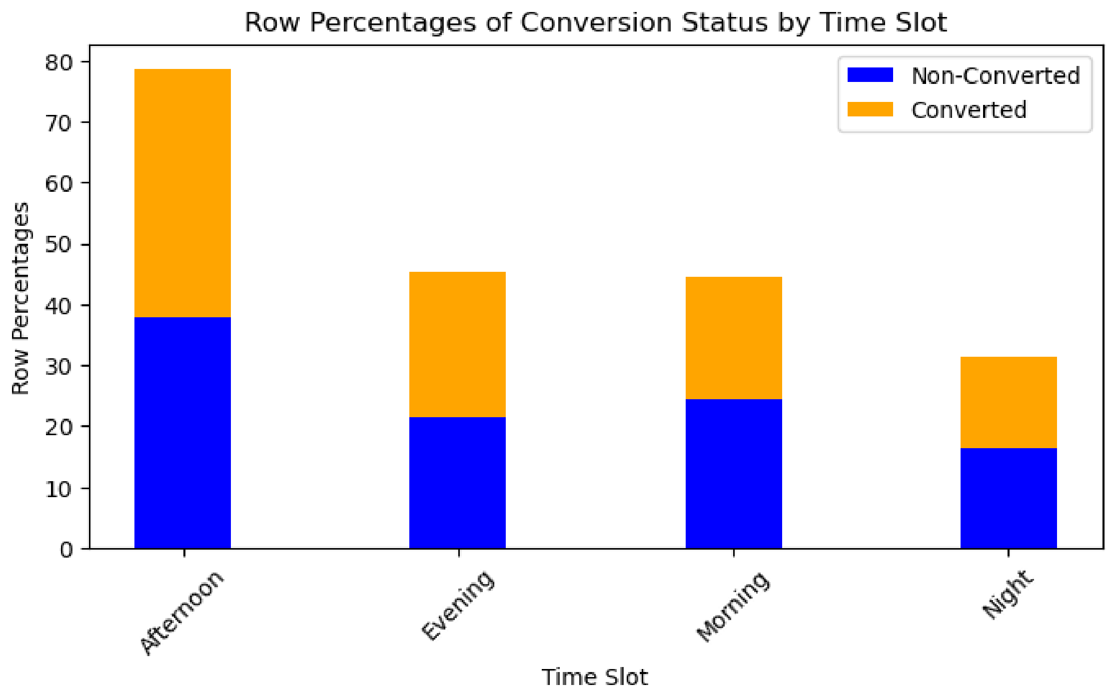
- Afternoon is the best-performing time slot both in terms of total conversions and exceeding expected conversions. It also has the highest percentage of users converting.
- Morning underperforms in terms of conversions, with significantly fewer conversions than expected based on the residuals.
- Evening performs decently, being the second-best time slot for conversions.
- Night has the lowest performance in both conversions and non-conversions.

The chi-square test indicates that time slot and conversion status are related, but the odds ratio (1.03) shows that the effect size is small, meaning there isn't a large difference in conversion rates between time slots. Thus, while Afternoon seems like the best time slot to show ads, the difference in conversion rates across time slots may not be substantial in practical terms, as indicated by the small odds ratio.

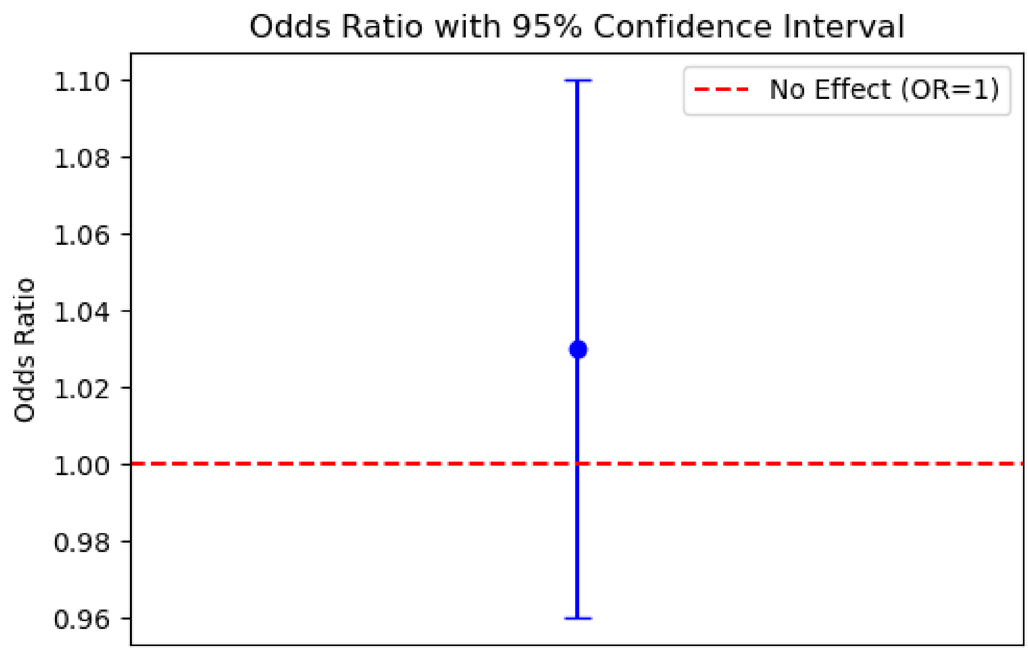
[78]:
```
categories = ['Afternoon', 'Evening', 'Morning', 'Night']
false_values = [216786, 123041, 140041, 93390]
true_values = [6065, 3545, 2977, 2256]

# Create a DataFrame
df3 = pd.DataFrame({
    'False': false_values,
    'True': true_values
}, index=categories)

# Plot
df3.plot(kind='bar', stacked=True, figsize=(8, 4))
plt.title('Conversion Frequencies by Time Slot')
plt.xlabel('Time Slot')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.legend(title='Converted')
plt.show()
```



[74]:
```
false_percentages = [37.82, 21.46, 24.43, 16.29]
true_percentages = [40.86, 23.88, 20.06, 15.20]

# Plot
fig, ax = plt.subplots(figsize=(8, 4))
width = 0.35  # Width of the bars
ax.bar(categories, false_percentages, width, label='Non-Converted',␣
 ↪color='blue')
ax.bar(categories, true_percentages, width, bottom=false_percentages,␣
 ↪label='Converted', color='orange')

ax.set_xlabel('Time Slot')
ax.set_ylabel('Row Percentages')
ax.set_title('Row Percentages of Conversion Status by Time Slot')
ax.legend()

plt.xticks(rotation=45)
plt.show()
```



[75]:
```
odds_ratio = 1.03
conf_int = [0.96, 1.10]

plt.figure(figsize=(6, 4))
plt.errorbar(1, odds_ratio, yerr=[[odds_ratio - conf_int[0]], [conf_int[1] -
    odds_ratio]], fmt='o', capsize=5, color='blue')
plt.axhline(1, linestyle='--', color='red', label='No Effect (OR=1)')
plt.title('Odds Ratio with 95% Confidence Interval')
plt.ylabel('Odds Ratio')
plt.xlim(0.5, 1.5)
plt.xticks([])
plt.legend()
plt.show()
```



comparison of highly conversion time of day. (Morning and Evening (second highest time of day))

[86]:
```python
# Create a new column to categorize users into 'Morning' or 'Evening' time slots
df['morning_evening_group'] = df['time_slot'].apply(lambda x: 'Morning' if x ==
↪'Morning' else ('Evening' if x == 'Evening' else 'Other'))

# Filter out the rows that are either 'Morning' or 'Evening'
df_filtered = df[df['morning_evening_group'].isin(['Morning', 'Evening'])]

# Create a contingency table for conversion in 'Morning' vs 'Evening' time slots
contingency_table_morning_evening = pd.crosstab(df_filtered['converted'],
↪df_filtered['morning_evening_group'])

# Calculate odds ratio and confidence interval for Morning vs Evening time slots
OR_CIs(contingency_table_morning_evening)
```

```
Odds Ratio: 0.74
95% Confidence Interval: 0.70, 0.78
```

[86]:
```
(0.7378315874832102, 0.7023231390612695, 0.7751352920190514)
```

7 Conclusion:

- The Evening time slot is a significantly more effective time for driving conversions compared to the Morning.
- Users exposed to ads in the Morning are 26% less likely to convert than users in the Evening, and this result is statistically significant given the low p-value and confidence interval entirely below 1.

8 Lets check is there relationship between converted and Test Group

- Chi-Square Test for Independence
- Converted and Test group

[96]:
```
df.drop(columns=['afternoon_group', 'morning_evening_group'], inplace=True)
```

[101]:
```
df
```

[101]:
```
        Unnamed: 0  user id test group  converted  total ads most ads day  \
0                0  1069124         ad      False        130       Monday
1                1  1119715         ad      False         93      Tuesday
2                2  1144181         ad      False         21      Tuesday
3                3  1435133         ad      False        355      Tuesday
4                4  1015700         ad      False        276       Friday
…              …        …          …          …          …
588096      588096  1278437         ad      False          1      Tuesday
588097      588097  1327975         ad      False          1      Tuesday
588098      588098  1038442         ad      False          3      Tuesday
588099      588099  1496395         ad      False          1      Tuesday
588100      588100  1237779         ad      False          1      Tuesday

        most ads hour  time_slot
0                  20    Evening
1                  22      Night
2                  18    Evening
3                  10    Morning
4                  14  Afternoon
…                 …         …
588096             23      Night
588097             23      Night
588098             23      Night
588099             23      Night
588100             23      Night

[588101 rows x 8 columns]
```

[102]:
```
# Create a contingency table
contingency_table_CT = pd.crosstab(df['converted'], df['test group'])

print("Contingency Table with Frequencies:")
display(contingency_table_CT)
print("#"*60)

# Calculate row percentages
```

```
row_percentages_CT = contingency_table_CT.div(contingency_table_CT.sum(axis=1),␣
 ↪axis=0) * 100

print("\nRow Percentages:")
display(row_percentages_CT)
print("#"*60)

# Perform chi-square test
chi2, p, dof, expected_CT = chi2_contingency(contingency_table_CT)

print(f"\nChi-squared value: {chi2}")
print(f"P-value: {p}")
print(f"Degrees of freedom: {dof}")
print("#"*60)

# Calculate the percentage of cells with expected counts less than 5
percentage_low_expected_CT = (expected < 5).sum().sum() / (expected.shape[0] *␣
 ↪expected.shape[1]) * 100

print(f"Percentage of cells with expected counts less than 5:␣
 ↪{percentage_low_expected_CT:.2f}%")
print("#"*60)

# Calculate residuals (observed minus expected values)
residuals_CT = contingency_table_CT - expected_CT

print("\nResiduals (Observed - Expected):")
display(residuals_CT)
print("#"*60)

# Calculate odds ratio
OR_CIs(contingency_table_CT)
```

```
Contingency Table with Frequencies:

test group      ad    psa
converted
False       550154  23104
True         14423    420

############################################################

Row Percentages:

test group         ad       psa
converted
False       95.969703  4.030297
True        97.170383  2.829617
```

```
###########################################################

Chi-squared value: 54.005823883685245
P-value: 1.9989623063390075e-13
Degrees of freedom: 1
###########################################################
Percentage of cells with expected counts less than 5: 0.00%
###########################################################

Residuals (Observed - Expected):

test group         ad        psa
converted
False      -173.71899  173.71899
True        173.71899 -173.71899

###########################################################
Odds Ratio: 0.69
95% Confidence Interval: 0.63, 0.76
```

[102]:
```
(0.6934110943143867, 0.6287384190783388, 0.7647360669053799)
```

[103]:
```python
df['test group'].value_counts()  # ad = ads  psa = no ads
```

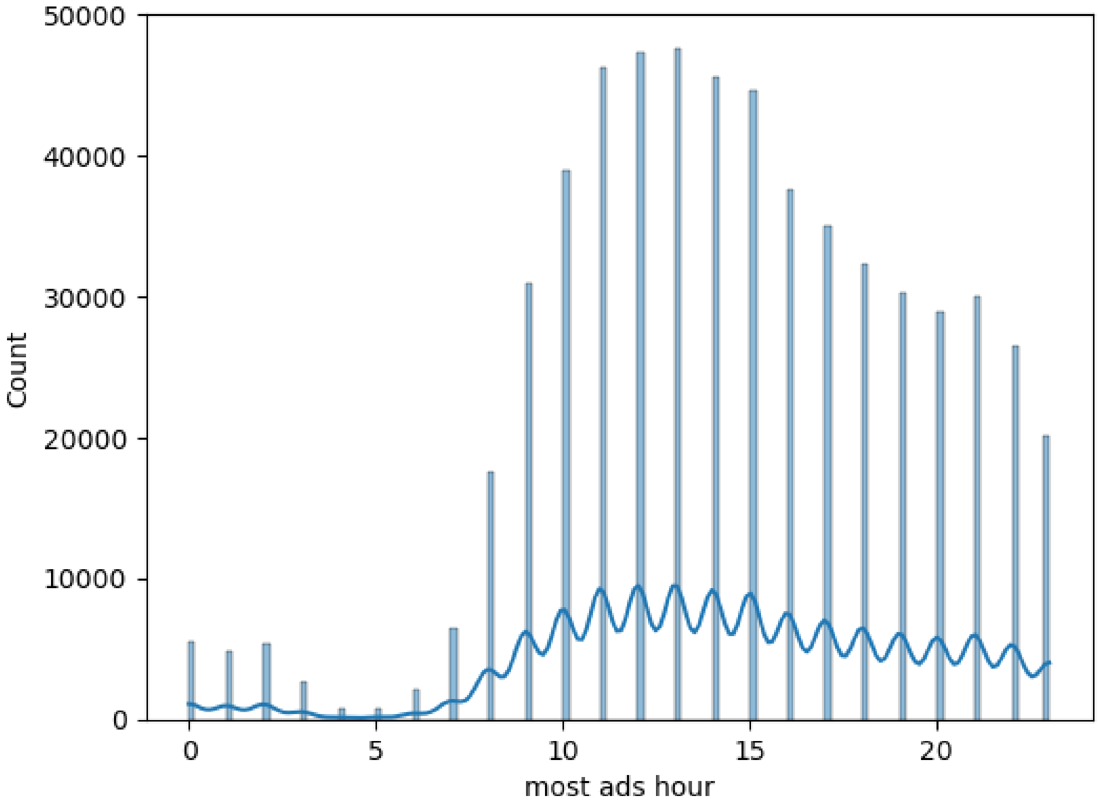
[103]:
```
ad     564577
psa     23524
Name: test group, dtype: int64
```

[104]:
```python
df.head(3)
```

[104]:
```
   Unnamed: 0  user id test group  converted  total ads most ads day  \
0           0  1069124         ad      False        130       Monday
1           1  1119715         ad      False         93      Tuesday
2           2  1144181         ad      False         21      Tuesday

   most ads hour time_slot
0             20   Evening
1             22     Night
2             18   Evening
```

[106]:
```python
sns.histplot(df['most ads hour'],kde=True)
```

[106]:
```
<Axes: xlabel='most ads hour', ylabel='Count'>
```




```
[108]: # between 10 - 15 people see the most ads which is in afternoon
```

- ads = experimental group (in which ads shown to user)
- psa = control group (in which users are unaware of ads)

Contingency Table with Frequencies:

1. Ad group:

- 550,154 users in the ad group did not convert.
- 14,423 users in the ad group converted.

2. PSA group:

- 23,104 users in the PSA group did not convert.
- 420 users in the PSA group converted.
- This shows that the majority of users, whether exposed to the ad or the PSA, did not convert. However, the number of conversions in the ad group is significantly higher than in the PSA group.

Row Percentages: 1. For Non-conversions (False): - 95.97% of non-converting users were from the ad group, while only 4.03% were from the PSA group.

2. For Conversions (True):

- 97.17% of converting users were from the ad group, while only 2.83% were from the PSA group.
- This shows that the ad group not only had a higher percentage of total conversions but also a higher percentage of users who converted compared to the PSA group.

Chi-square Test: - Chi-squared value: 54.01 and p-value: 1.99e-13: - The very low p-value (much less than 0.05) indicates that there is a statistically significant relationship between being exposed to the ad or PSA and whether users converted. - This means that the difference in conversion rates between the ad group and the PSA group is unlikely due to chance and is instead a real effect.

Residuals (Observed - Expected): - Positive residuals in the ad group for conversions (173.72) indicate that more users in the ad group converted than expected. - Negative residuals in the PSA group for conversions (-173.72) indicate that fewer users in the PSA group converted than expected. - The residuals tell us that the ad group significantly outperformed the PSA group in terms of conversions.

Odds Ratio: - Odds Ratio = 0.69: - This means that users in the PSA group are 31% less likely to convert compared to users in the ad group (since 1 - 0.69 = 0.31 or 31%).

This indicates that the ad group is more effective at driving conversions compared to the PSA group. 95% Confidence Interval = (0.63, 0.76): We are 95% confident that the true odds ratio lies between 0.63 and 0.76. Since the confidence interval is below 1, it confirms that the PSA group has a lower likelihood of conversion compared to the ad group.

Summary of Insights:

- Ad effectiveness: The ad group significantly outperforms the PSA group in driving conversions, as evidenced by the higher number of conversions in the ad group and the statistically significant chi-squared test result.

- Odds Ratio: Users in the PSA group are 31% less likely to convert than those in the ad group, which suggests that the ads have a positive impact on conversions.

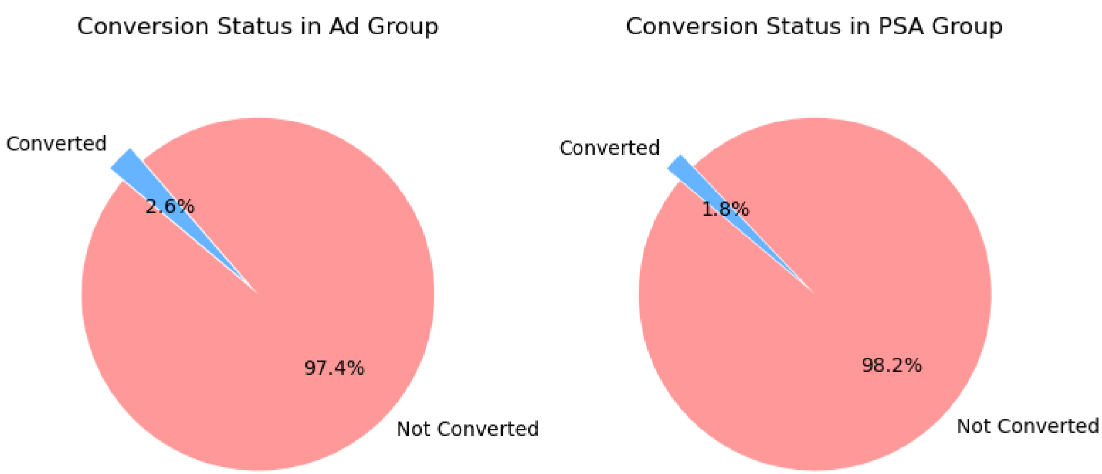
- Significance: The low p-value shows that the relationship between exposure to the ad and conversions is statistically significant and not due to random chance.

- Residuals: The positive residuals in the ad group for conversions show that more users converted than expected, reinforcing the idea that the ads are effective at driving purchases.

Actionable Insight: - Ads are effective: Based on these results, the company can confidently conclude that the ad campaign is effective in increasing conversions compared to the PSA. - It is recommended to continue or scale the ad campaign, as it is clearly having a positive impact on user conversions.

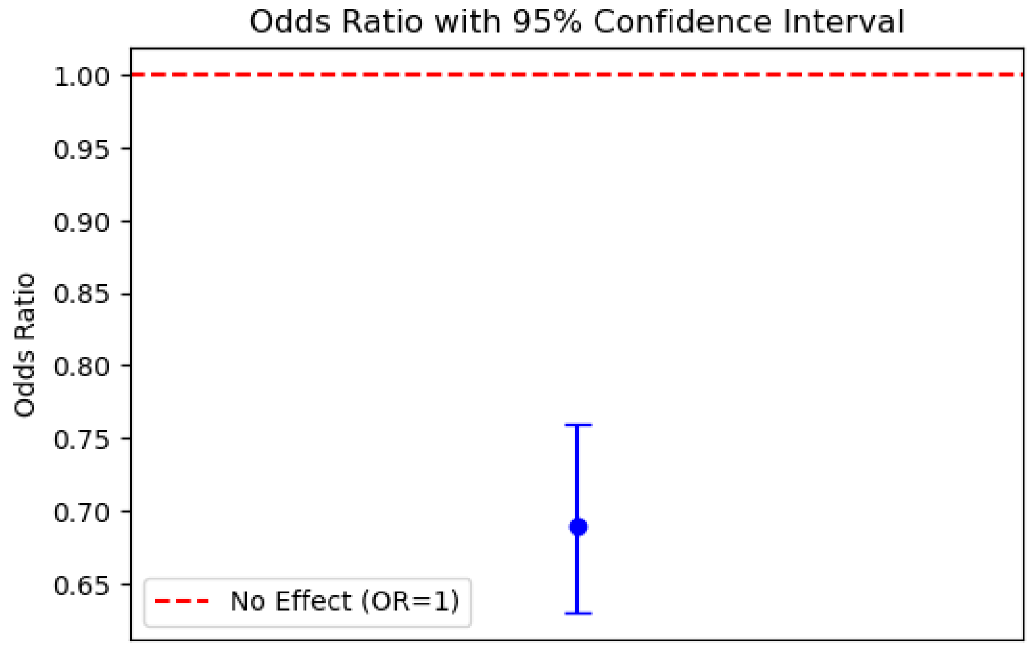
[111]:
```
# Data for 'ad' and 'psa' groups
labels = ['Not Converted', 'Converted']
sizes_ad = [550154, 14423]  # False, True counts for 'ad'
sizes_psa = [23104, 420]    # False, True counts for 'psa']
colors = ['#ff9999', '#66b3ff']
explode = (0.1, 0)  # explode 1st slice (Not Converted)

# Create subplots for side-by-side pie charts
```

```
fig, axes = plt.subplots(1, 2, figsize=(8, 4))

# Pie chart for 'ad' group
axes[0].pie(sizes_ad, explode=explode, labels=labels, colors=colors,␣
 ↪autopct='%1.1f%%', startangle=140)
axes[0].set_title('Conversion Status in Ad Group')
axes[0].axis('equal')  # Equal aspect ratio ensures the pie is drawn as a␣
 ↪circle.

# Pie chart for 'psa' group
axes[1].pie(sizes_psa, explode=explode, labels=labels, colors=colors,␣
 ↪autopct='%1.1f%%', startangle=140)
axes[1].set_title('Conversion Status in PSA Group')
axes[1].axis('equal')  # Equal aspect ratio ensures the pie is drawn as a␣
 ↪circle.

# Display both pie charts
plt.tight_layout()
plt.show()
```




```
[112]: odds_ratio = 0.69  # This is a placeholder value
conf_int = [0.63, 0.76]  # Confidence interval for the odds ratio

plt.figure(figsize=(6, 4))
plt.errorbar(1, odds_ratio, yerr=[[odds_ratio - conf_int[0]], [conf_int[1] -␣
 ↪odds_ratio]], fmt='o', capsize=5, color='blue')
plt.axhline(1, linestyle='--', color='red', label='No Effect (OR=1)')
plt.title('Odds Ratio with 95% Confidence Interval')
```

```
plt.ylabel('Odds Ratio')
plt.xlim(0.5, 1.5)
plt.xticks([])
plt.legend()
plt.show()
```



Summary:

Interpretation: The PSA group has a lower conversion rate than the ad group, and the odds ratio is significantly different from 1 (as indicated by the fact that the confidence interval does not cross the red dashed line). This confirms that the ads are effective in driving conversions compared to the PSAs.

9 Independent Samples t-Test

Converted and Total ads

Assumption of Homogeneity of Variance in Independent Samples t-Test

- converted: If a person bought the product then True, else is False
- total ads: Amount of ads seen by person

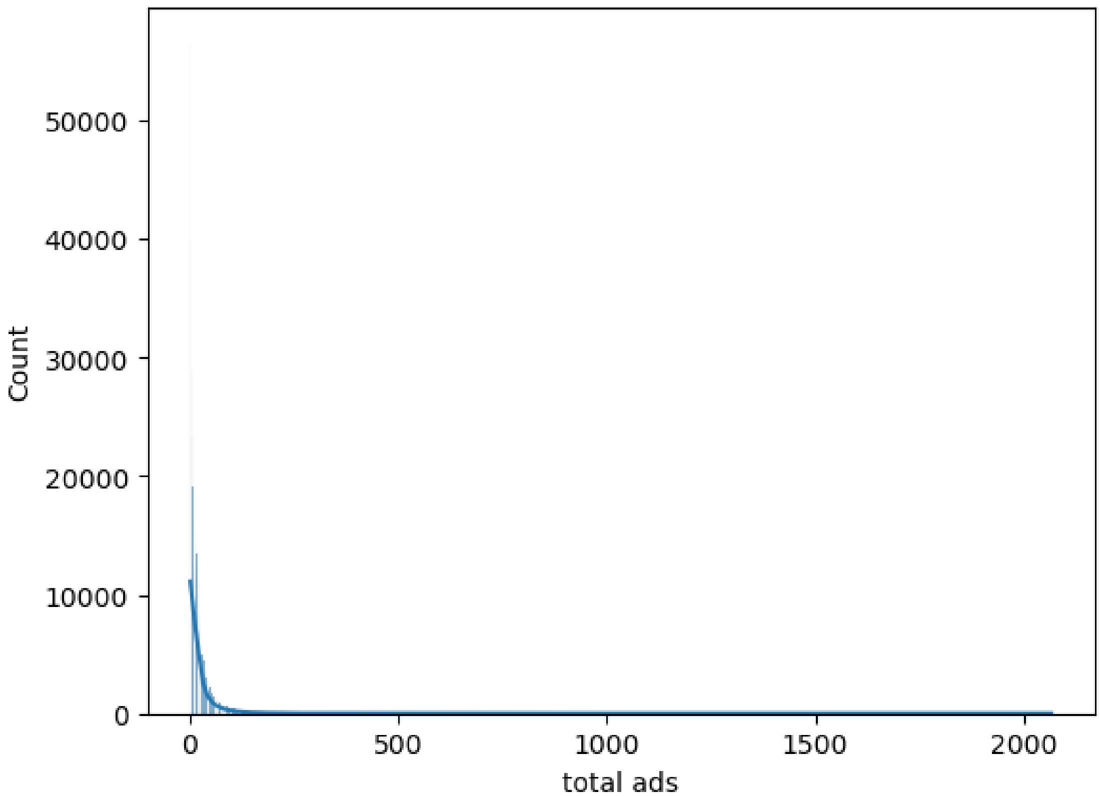
[9]:
```
df.head()
```

[9]:
```
   Unnamed: 0  user id test group  converted  total ads most ads day  \
0           0  1069124         ad      False        130       Monday
1           1  1119715         ad      False         93      Tuesday
```

```
2            2  1144181          ad       False          21       Tuesday
3            3  1435133          ad       False         355       Tuesday
4            4  1015700          ad       False         276        Friday

   most ads hour
0             20
1             22
2             18
3             10
4             14
```

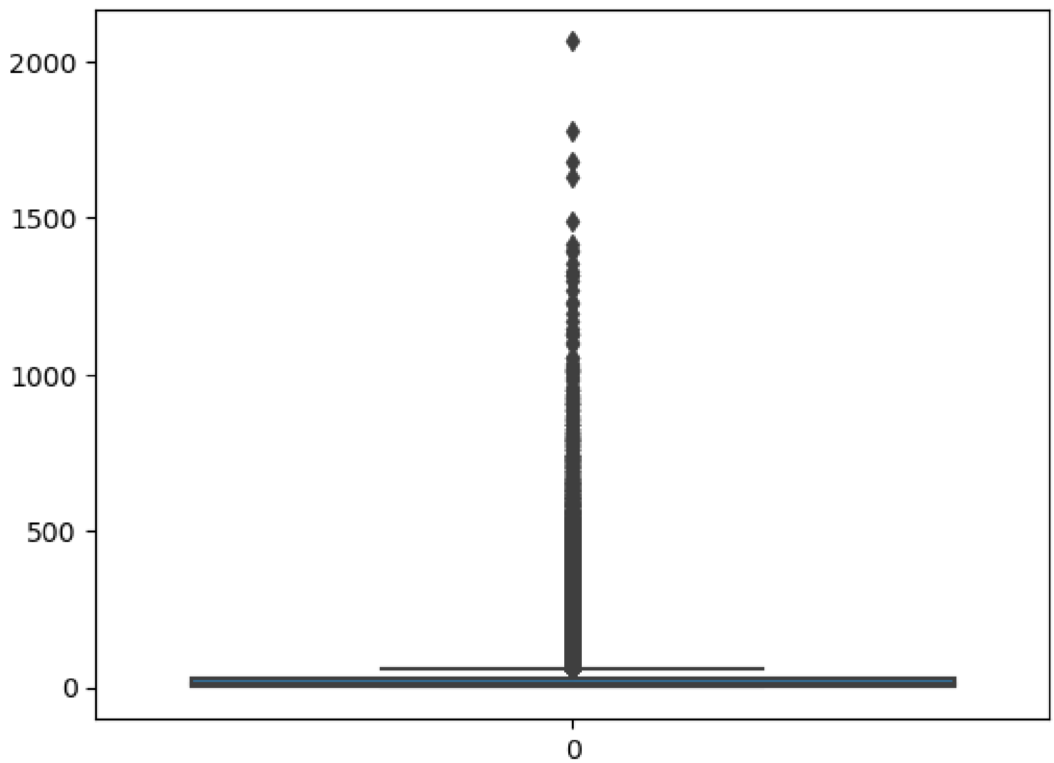
[11]:
```
sns.histplot(df['total ads'],kde=True)
```

[11]: <Axes: xlabel='total ads', ylabel='Count'>



[12]:
```
sns.boxplot(df['total ads'])
```

[12]: <Axes: >



[17]:
```
# Assuming df is your DataFrame and it contains 'converted' and 'total ads'␣
↪columns

# Step 1: Group the data
# Extract the total ads seen by converted (True) and not converted (False) users
ads_converted = df[df['converted'] == True]['total ads']
ads_not_converted = df[df['converted'] == False]['total ads']

# Step 2: Perform an independent two-sample t-test
# Null hypothesis: There is no significant difference in the mean number of ads␣
↪shown between converters and non-converters
# Alternative hypothesis: There is a significant difference in the mean number␣
↪of ads shown
t_stat, p_value = ttest_ind(ads_converted, ads_not_converted, equal_var=False) ␣
↪# Welch's t-test in case variances are unequal

# Print the results
print(f"T-statistic: {t_stat}")
print(f"P-value: {p_value}")
```

```
# Interpretation: If the p-value is less than 0.05, we reject the null␣
↪hypothesis and conclude that there is a significant difference.
```

```
T-statistic: 84.17740664633055
P-value: 0.0
```

[16]:
```
ads_converted
```

[16]:
```
15          9
44        265
107      1328
121       323
135       246
         …
586343     14
586818     11
586990      8
587069      4
587665      3
Name: total ads, Length: 14843, dtype: int64
```

[18]:
```
import pandas as pd
from scipy.stats import ttest_ind

# Assuming df is your DataFrame and it contains 'converted' and 'total ads'␣
↪columns

# Step 1: Group the data
# Extract the total ads seen by converted (True) and not converted (False) users
ads_converted = df[df['converted'] == True]['total ads']
ads_not_converted = df[df['converted'] == False]['total ads']

# Step 2: Perform an independent two-sample t-test
# Null hypothesis: There is no significant difference in the mean number of ads␣
↪shown between converters and non-converters
# Alternative hypothesis: There is a significant difference in the mean number␣
↪of ads shown
t_stat, p_value = ttest_ind(ads_converted, ads_not_converted, equal_var=False) ␣
↪# Welch's t-test in case variances are unequal

# Print the results
print(f"T-statistic: {t_stat}")
print(f"P-value: {p_value}")

# Interpretation: If the p-value is less than 0.05, we reject the null␣
↪hypothesis and conclude that there is a significant difference.
```

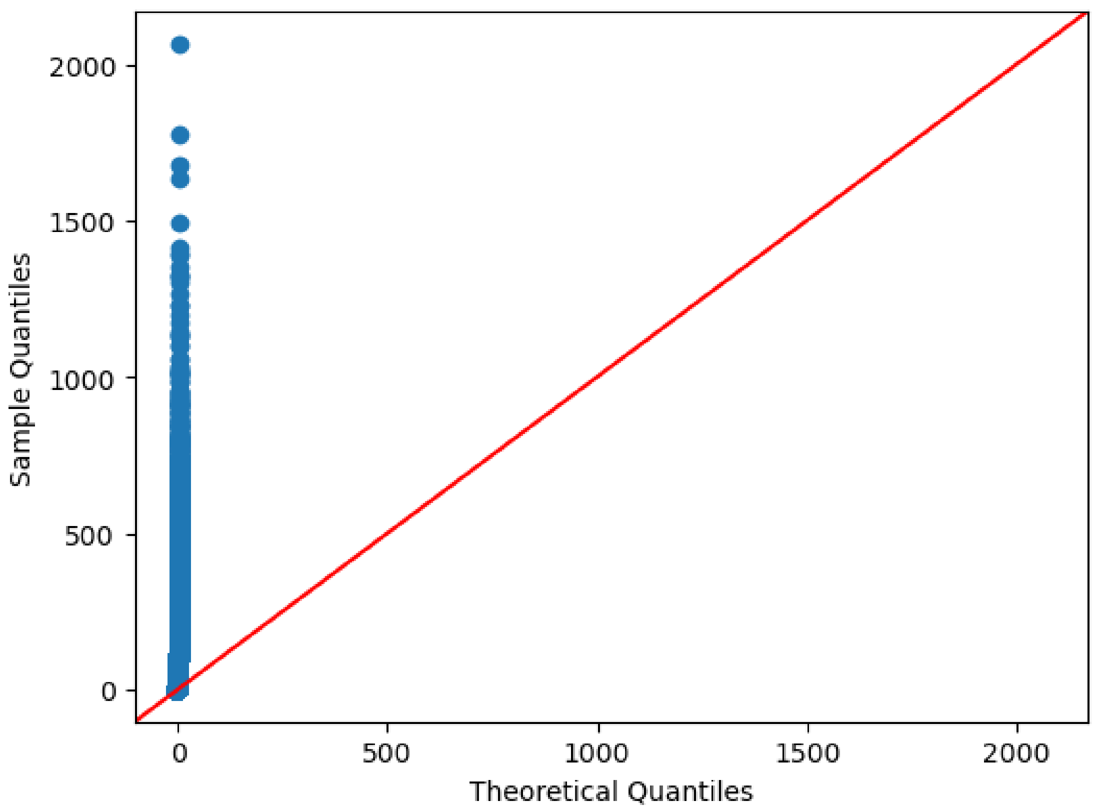
```
T-statistic: 84.17740664633055
```

P-value: 0.0

[19]:
```
# Mean of total ads for users who converted (True) and did not convert (False)
mean_ads_converted = df[df['converted'] == True]['total ads'].mean()
mean_ads_not_converted = df[df['converted'] == False]['total ads'].mean()

# Print the results
print(f"Mean total ads for converted users: {mean_ads_converted}")
print(f"Mean total ads for not converted users: {mean_ads_not_converted}")
```

Mean total ads for converted users: 83.88775853937884
Mean total ads for not converted users: 23.291495277867906

[21]:
```
sm.qqplot(df['total ads'], line ='45')
py.show()
```



[22]:
```
# H0: Data is Gaussian
# Ha: Data is not Gaussian
test_stat, p_value = shapiro(df['total ads'])
print(p_value)

if p_value < 0.05:
```

```
    print("Reject H0")
    print("Data is not Gaussian")
else:
    print("Fail to reject H0")
    print("Data is Gaussian")
```

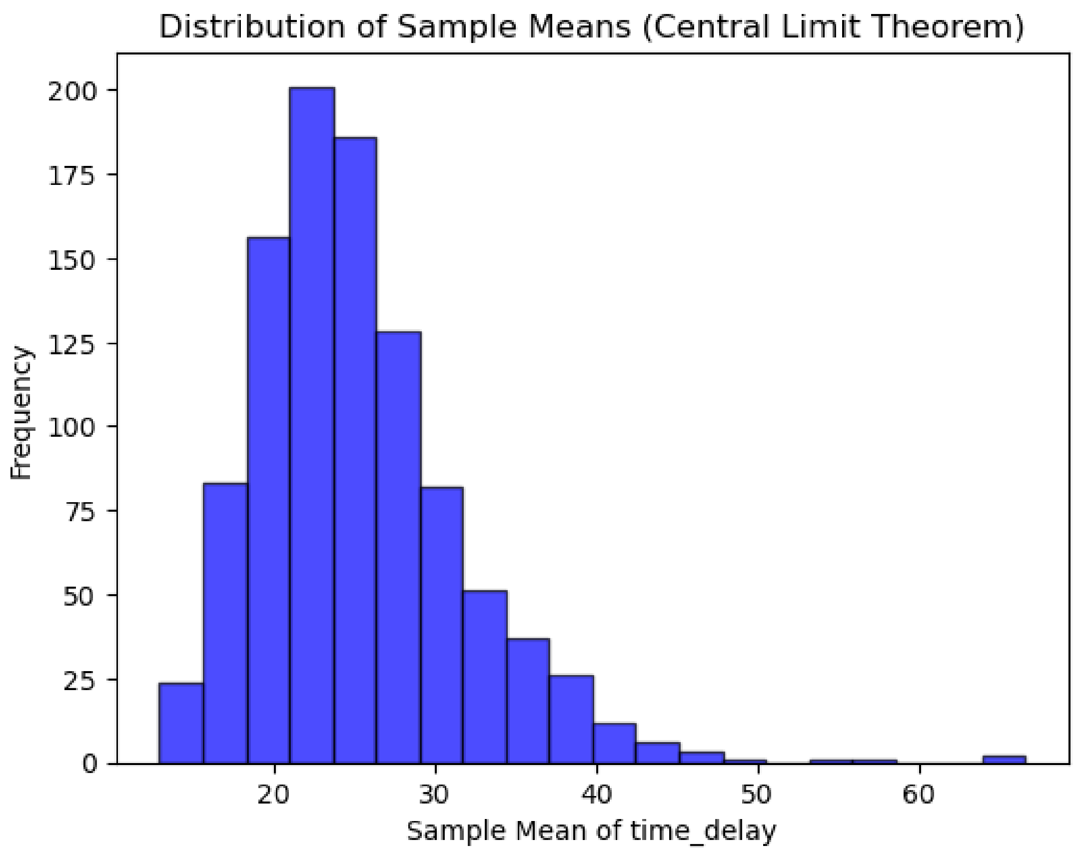
```
0.0
Reject H0
Data is not Gaussian
```

[41]:

```
# So now we will use CLT method by taking sample

num_samples = 1000  # Ideal number of samples
sample_size = 50   # Each sample should have more than 30 rows

# Function to generate samples
samples = []
for i in range(num_samples):
    sample = df.sample(n=sample_size, random_state=i)  # Taking random sample␣
 ↪of size 30
    samples.append(sample)

# Convert list of samples into a DataFrame for analysis
samples_df = pd.concat(samples, keys=range(num_samples))  # 'keys' label each␣
 ↪sample
```

[42]:

```
# Calculating the mean of time_delay2 for each sample
sample_means = [sample['total ads'].mean() for sample in samples]

# Plot the distribution of the sample means
plt.hist(sample_means, bins=20, color='blue', edgecolor='black', alpha=0.7)
plt.title('Distribution of Sample Means (Central Limit Theorem)')
plt.xlabel('Sample Mean of time_delay')
plt.ylabel('Frequency')
plt.show()
```




```
[43]: # H0: Data is Gaussian
      # Ha: Data is not Gaussian
      test_stat, p_value = shapiro(sample['total ads'])
      print(p_value)

      if p_value < 0.05:
          print("Reject H0")
          print("Data is not Gaussian")
      else:
          print("Fail to reject H0")
          print("Data is Gaussian")
```

```
1.2165937528230142e-11
Reject H0
Data is not Gaussian
```

```
[45]: # Group the data based on conversion status
      ads_converted = df[df['converted'] == True]['total ads']
      ads_not_converted = df[df['converted'] == False]['total ads']
```

```python
# Perform the Kruskal-Wallis test
stat, p_value = kruskal(ads_converted, ads_not_converted)

# Output the result
print(f"Kruskal-Wallis H-statistic: {stat}")
print(f"P-value: {p_value}")

# Interpretation:
if p_value < 0.05:
    print("There is a statistically significant difference between the groups␣
 ↪(reject H ).")
else:
    print("There is no statistically significant difference between the groups␣
 ↪(fail to reject H ).")
```

```
Kruskal-Wallis H-statistic: 21832.90065407058
P-value: 0.0
There is a statistically significant difference between the groups (reject H ).
```

[48]:
```python
from scipy.stats import ttest_ind


# Separate the data into two groups based on the 'converted' column
groupN = df[df['converted'] == False]['total ads']
groupY = df[df['converted'] == True]['total ads']
# Perform two-sample t-test
t_statistic, p_value = ttest_ind(groupN, groupY, equal_var=False)

# Print the results
print("t-statistic:", t_statistic)
print("p-value:", p_value)

# Determine significance
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant difference␣
 ↪between the means.")
else:
    print("Fail to reject the null hypothesis: There is no significant␣
 ↪difference between the means.")
```

```
t-statistic: -84.17740664633055
p-value: 0.0
Reject the null hypothesis: There is a significant difference between the means.
```

[49]:
```python
# Descriptive statistics for the 'total ads' seen by converted and␣
 ↪non-converted groups
print(groupN.describe())  # Non-converted group
```

```
print(groupY.describe())  # Converted group
```

```
count    573258.000000
mean         23.291495
std          40.863176
min           1.000000
25%           4.000000
50%          13.000000
75%          26.000000
max        2065.000000
Name: total ads, dtype: float64
count    14843.000000
mean        83.887759
std         87.455498
min          1.000000
25%         35.000000
50%         64.000000
75%        103.000000
max       1778.000000
Name: total ads, dtype: float64
```

[52]:

```
from scipy.stats import levene

# Example data for ads seen by converted and not converted users
converted_ads_seen = [10, 7, 9, 12]  # Example data
not_converted_ads_seen = [15, 20, 18, 22]  # Example data

# Perform Levene's test
stat, p_value = levene(groupN, groupY)

print(f"Levene's Test statistic: {stat}, P-value: {p_value}")
```

```
Levene's Test statistic: 9121.196956737573, P-value: 0.0
```

[53]:

```
t_stat, p_value = stats.ttest_ind(groupY, groupN, equal_var=False)
print(f"T-statistic: {t_stat}, P-value: {p_value}")
```

```
T-statistic: 84.17740664633055, P-value: 0.0
```

[54]:

```
mean_ads_converted = ads_converted.mean()
mean_ads_not_converted = ads_not_converted.mean()

print(f"Mean ads for converted users: {mean_ads_converted}")
print(f"Mean ads for not converted users: {mean_ads_not_converted}")
```

```
Mean ads for converted users: 83.88775853937884
Mean ads for not converted users: 23.291495277867906
```

[62]:
```python
import numpy as np

# Calculate means and standard deviations for both groups
mean_not_converted = np.mean(ads_not_converted)
mean_converted = np.mean(ads_converted)
std_converted = np.std(ads_converted, ddof=1)  # Use ddof=1 to get sample␣
↪standard deviation
std_not_converted = np.std(ads_not_converted, ddof=1)

# Get the number of observations in each group
n_converted = len(ads_converted)
n_not_converted = len(ads_not_converted)

# Calculate the pooled standard deviation
pooled_std = np.sqrt(((n_converted - 1) * std_converted**2 + (n_not_converted -␣
↪1) * std_not_converted**2) / (n_converted + n_not_converted - 2))

# Calculate Cohen's d
cohens_d = (mean_converted - mean_not_converted) / pooled_std

print(f"Cohen's d: {cohens_d}")
```

```
Cohen's d: 1.4201314076149827
```

A Cohen's d of 1.42 indicates a very large effect size. Here's what this means in the context of your data:

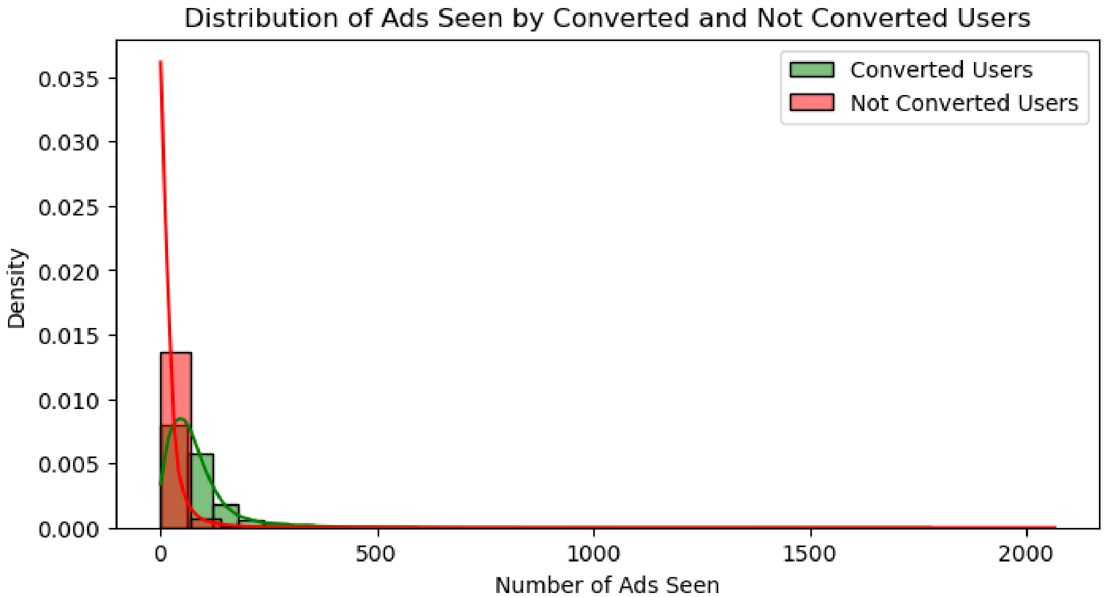
Interpretation:

- 1.42 is significantly greater than the thresholds for small (0.2), medium (0.5), and large (0.8) effect sizes.
- This large effect size suggests that the difference in the number of ads seen between users who converted and those who did not is substantial. In other words, the number of ads seen by converted users is much higher than those who did not convert, and this difference is not just statistically significant (as shown by your previous tests), but also practically meaningful.

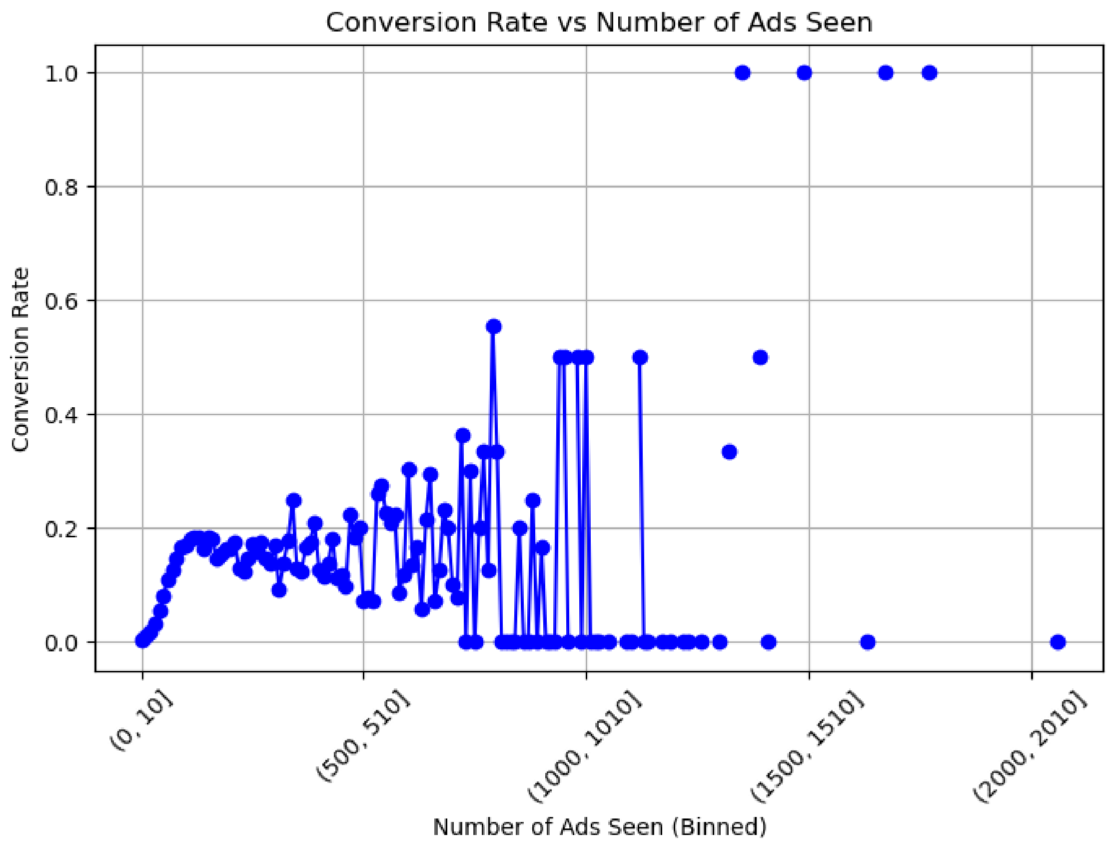
10 Optimal Number of Ads for Conversion

[64]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Plot histogram of ads for converted and not converted users
plt.figure(figsize=(8, 4))
sns.histplot(ads_converted, color="green", label="Converted Users", kde=True,␣
↪stat="density", bins=30)
sns.histplot(ads_not_converted, color="red", label="Not Converted Users",␣
↪kde=True, stat="density", bins=30)
plt.title("Distribution of Ads Seen by Converted and Not Converted Users")
```

```
plt.xlabel("Number of Ads Seen")
plt.ylabel("Density")
plt.legend()
plt.show()
```



[66]:
```
# Group users by number of ads seen and calculate conversion rate for each group
df['ads_seen_bins'] = pd.cut(df['total ads'], bins=range(0, int(df['total ads'].
    ↪max()) + 10, 10))

conversion_rates = df.groupby('ads_seen_bins')['converted'].mean()

# Plot conversion rate vs ads seen
plt.figure(figsize=(8, 5))
conversion_rates.plot(kind='line', marker='o', color='blue')
plt.title("Conversion Rate vs Number of Ads Seen")
plt.xlabel("Number of Ads Seen (Binned)")
plt.ylabel("Conversion Rate")
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```



Key Observations:

Initial Growth (0-500 ads):

- The conversion rate starts off low for users who saw between 0-10 ads.
- As the number of ads seen increases (up to around 500 ads), the conversion rate rises gradually, with some fluctuations. This indicates that users who see more ads within this range are slightly more likely to convert.

Significant Fluctuations (500-1000 ads):

- Between 500 and 1000 ads, the conversion rate fluctuates wildly. There are several sharp increases and decreases in conversion rates, but the overall trend doesn't seem stable.
- This might be due to fewer users seeing such a large number of ads, which leads to more variability in conversion rates.
- Very High Ad Exposure (1000+ ads):
- After around 1000 ads, conversion rates drop to nearly 0 or show very erratic behavior.
- There are some outliers where users who saw a very high number of ads (closer to 2000) had high conversion rates, but these seem to be outliers rather than a consistent trend.

Interpretation:

- Optimal Range for Conversion: It seems that showing users between 100 and 500 ads is where conversion rates are most stable and relatively higher.
- Too Many Ads May Hurt: After 500 ads, the conversion rate becomes highly variable, and after 1000 ads, it drops close to zero for most users. This could indicate that showing too many ads might not be effective or could lead to ad fatigue.
- Outliers: Some users who saw a large number of ads (around 2000) converted, but these outliers do not reflect the general trend and could represent special cases.

Actions to Consider: - You might want to limit the number of ads shown to users, as conversion rates tend to plateau or become erratic after about 500 ads. - the general trend suggests that too many ads might not be beneficial.

11 A/B Test the equality of proportions hypothesis

[93]:
```
cross_tab = pd.crosstab(df['test group'], df['converted'],normalize='index')
cross_tab
```

[93]:
```
converted      False     True
test group
ad          0.974453  0.025547
psa         0.982146  0.017854
```

Define the null and alternative hypothsis - Null Hypothesis (H): there is no difference in the conversion rates between the test group - Alternative Hypothesis : there is a difference in the conversion rates between the test group and the control group

Set the probability of type I and type II errors - we set : =0.05 and =0.2.

Calculate the sample size

Based on the provided conversion data, we have the conversion rates for two groups in an A/B test: one group that saw the advertisement ("ad") and another that saw a public service announcement ("psa"). The conversion rates are calculated as follows:

Ad Group:

- Conversion Rate: 2.55% (0.025547)

PSA Group: - Conversion Rate: 1.79% (0.017854) To determine the sample size needed for such an A/B test, you can use these conversion rates as your baseline rates (p1 and p2). Here's how you can proceed: Steps to Determine Sample Size

Define Parameters:

- Baseline Conversion Rate (p1): Use the conversion rate of the PSA group, which is 1.79%.
- Expected Conversion Rate (p2): Use the conversion rate of the Ad group, which is 2.55%.
- Significance Level ( ): Typically set at 0.05.
- Power (1-): Typically set at 0.8 (80%).

[69]:
```
alpha = 0.05  # Significance level
power = 0.8   # Power
p1 = 0.017854  # Conversion rate for control group (psa)
```

```
p2 = 0.025547  # Conversion rate for test group (ad)

# Calculate the average conversion rate
p = (p1 + p2) / 2

# Calculate the effect size
effect_size = (p2 - p1) / ((p * (1 - p)) ** 0.5)

# Calculate the sample size per group
power_analysis = NormalIndPower()
sample_size = power_analysis.solve_power(effect_size=effect_size, power=power,␣
 ↪alpha=alpha, ratio=1)

print(f"Required sample size per group: {int(sample_size)}")
```

```
Required sample size per group: 5631
```

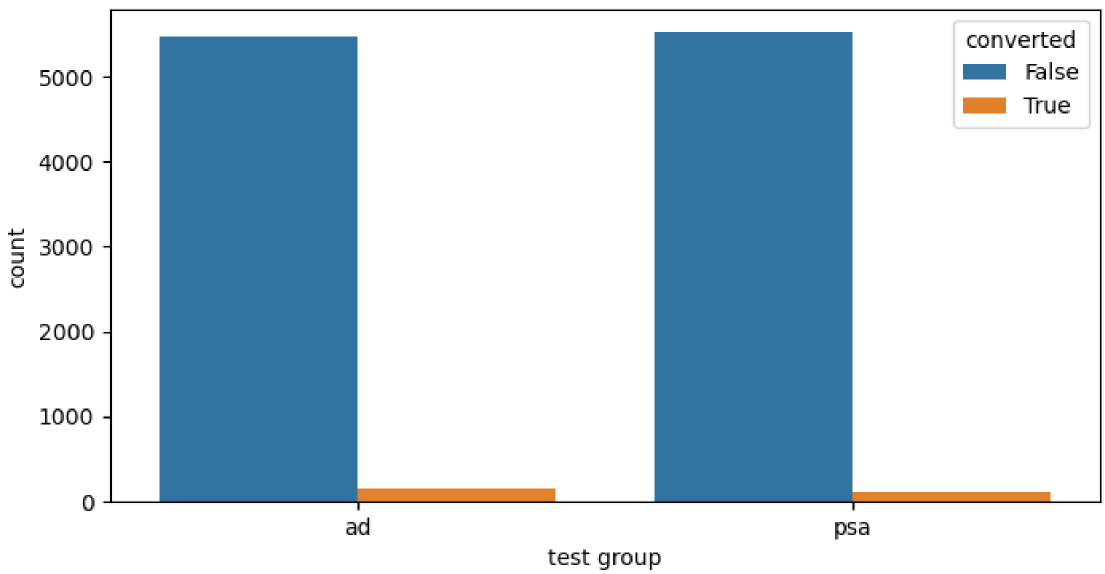
[70]:
```
def choose_random_sample(data,random_state=42):
    return data.sample(n=int(sample_size), random_state=random_state)

# Apply the function to each group using groupby
df_smpl = df.groupby('test group', group_keys=False).apply(choose_random_sample)
df_smpl
```

[70]:
```
        Unnamed: 0  user id test group  converted  total ads most ads day  \
529666      529666  1300427         ad      False         21       Friday
385537      385537  1197483         ad      False          2     Thursday
120467      120467  1234257         ad      False         20       Sunday
186608      186608  1384841         ad       True         47       Friday
141292      141292  1646962         ad      False         13      Tuesday
…              …        …          …          …          …
158646      158646   909937        psa      False          7     Thursday
521629      521629   903824        psa      False          6     Thursday
376798      376798   920768        psa      False          7    Wednesday
282973      282973   908849        psa      False         12       Monday
547431      547431   902688        psa      False         24       Monday

        most ads hour ads_seen_bins
529666             20      (20, 30]
385537             20       (0, 10]
120467             10      (10, 20]
186608             14      (40, 50]
141292             13      (10, 20]
…                 …            …
158646             12       (0, 10]
521629             20       (0, 10]
376798             19       (0, 10]
```

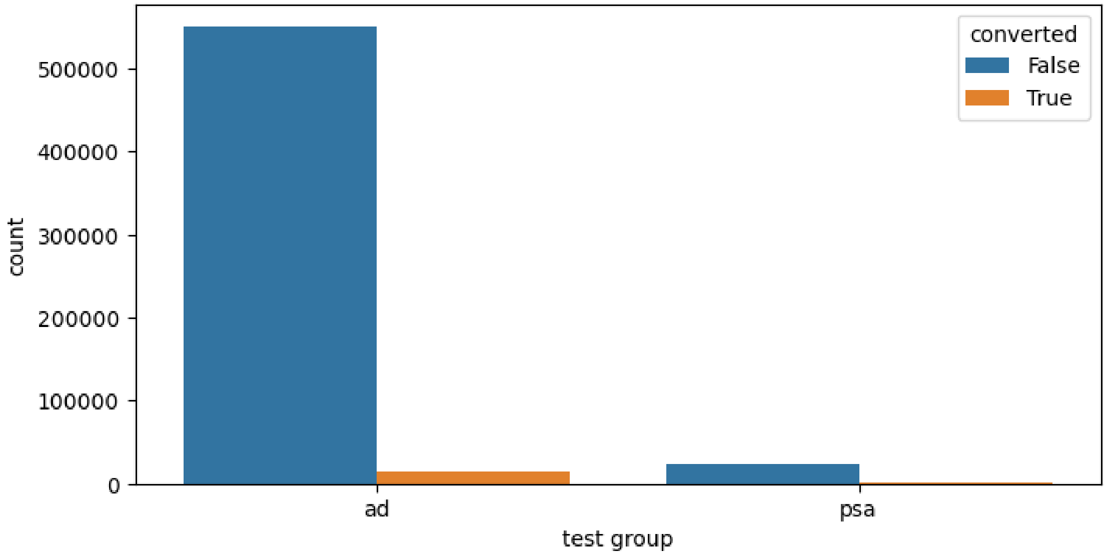
```
282973              13      (10, 20]
547431              19      (20, 30]

[11262 rows x 8 columns]
```

12 visualize samples

[80]:
```
fig, ax = plt.subplots(figsize=(8, 4))
sns.countplot(df_smpl, x='test group', hue='converted', ax=ax)
plt.show()
```



[81]:
```
fig, ax = plt.subplots(figsize=(8, 4))
sns.countplot(df, x='test group', hue='converted', ax=ax)
plt.show()
```



13 Test the equality of proportions hypothesis

[82]:
```
# Create the contingency table
contingency_table_Z = pd.crosstab(df_smpl['test group'], df_smpl['converted'])


print("Contingency Table with Frequencies:")
print(contingency_table_Z)
print("#"*60)

# Calculate row percentages
row_percentages = contingency_table_Z.div(contingency_table_Z.sum(axis=1),␣
↪axis=0) * 100

print("\nRow Percentages:")
print(row_percentages)
print("#"*60)

# Count the number of successes and trials in each group
success_ad = contingency_table_Z.loc['ad', True]
trials_ad = contingency_table_Z.loc['ad', False] + contingency_table_Z.
↪loc['ad', True]

success_psa = contingency_table_Z.loc['psa', True]
trials_psa = contingency_table_Z.loc['psa', False] + contingency_table_Z.
↪loc['psa', True]
```

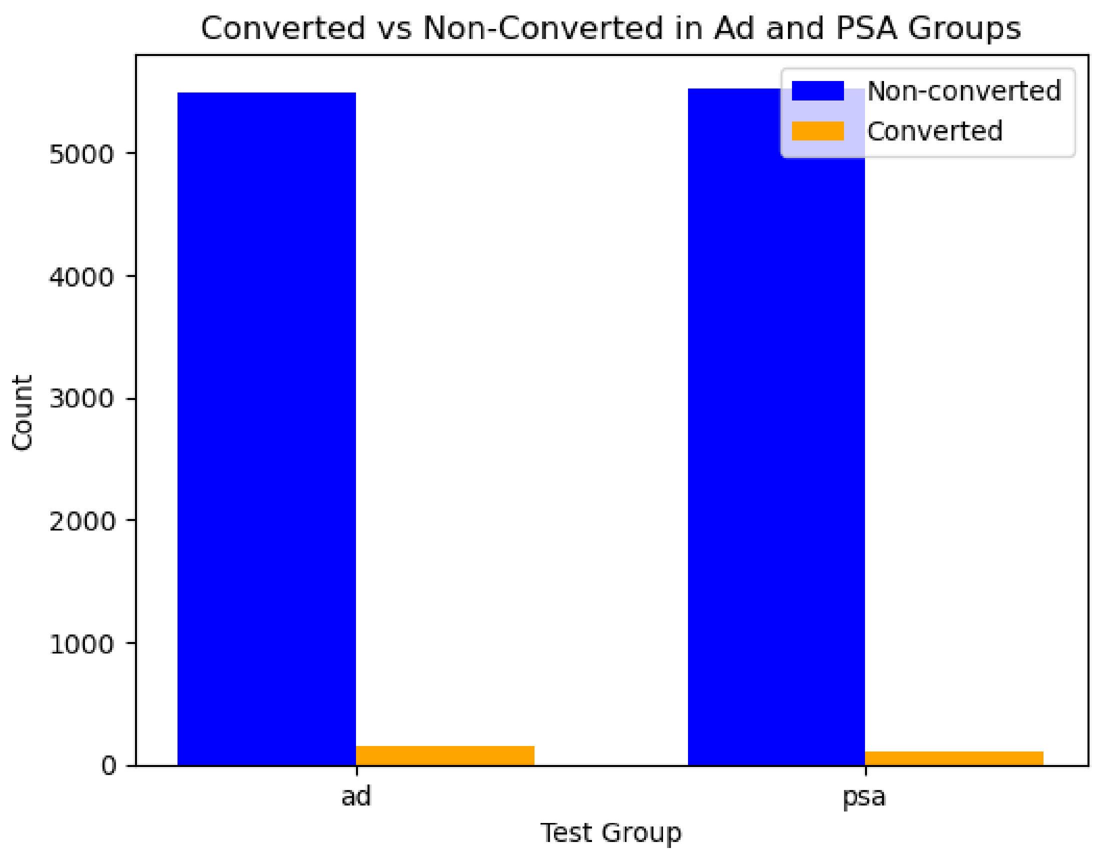
```python
# Perform the z-test for proportions
z_stat, p_value = sm.stats.proportions_ztest(
    [success_ad, success_psa],
    [trials_ad, trials_psa],
    alternative='larger'
)

# Print the results
print(f"Z-statistic: {z_stat}")
print(f"P-value: {p_value}")

# Interpret the results
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis. There is a significant difference in␣
 ↪proportions.")
else:
    print("Fail to reject the null hypothesis. Proportions are not␣
 ↪significantly different.")
```

```
Contingency Table with Frequencies:
converted   False  True
test group
ad           5481    150
psa          5524    107
############################################################

Row Percentages:
converted       False     True
test group
ad          97.336175  2.663825
psa         98.099805  1.900195
############################################################
Z-statistic: 2.7134050689178086
P-value: 0.003329782308620703
Reject the null hypothesis. There is a significant difference in proportions.
```

- A p-value of 0.0033 is less than the significance level (typically 0.05), meaning we reject the null hypothesis.
- This indicates that there is a statistically significant difference in the proportions of conversions between the ad group and the PSA group. The difference in the proportions (2.66% vs. 1.90%) is not due to random chance.

[83]:
```python
# Data from the contingency table
groups = ['ad', 'psa']
false_values = [5481, 5524]
true_values = [150, 107]
```

```
# Plot a grouped bar chart
bar_width = 0.35
index = range(len(groups))

plt.bar(index, false_values, bar_width, label='Non-converted', color='blue')
plt.bar([i + bar_width for i in index], true_values, bar_width,␣
 ↪label='Converted', color='orange')

plt.xlabel('Test Group')
plt.ylabel('Count')
plt.title('Converted vs Non-Converted in Ad and PSA Groups')
plt.xticks([i + bar_width / 2 for i in index], groups)
plt.legend()
plt.show()
```



14 Calculate the confidence interval for odds ratio

[86]:
```
def OR_CIs(contingency_table):

    # Calculate odds ratio
    odds_ratio = (contingency_table.iloc[0, 0] / contingency_table.iloc[0, 1]) /
 ↪ (contingency_table.iloc[1, 0] / contingency_table.iloc[1, 1])

    # Calculate standard error of log(odds ratio)
    log_odds_std_error = np.sqrt(contingency_table.applymap(lambda x: 1/x).
 ↪sum().sum())

    # Set confidence level
    confidence_level = 0.95

    # Calculate z-score for the confidence interval
    z_score = norm.ppf(1-(1 - confidence_level) / 2)

    # Calculate confidence intervals
    ci_low = np.exp(np.log(odds_ratio) - z_score * log_odds_std_error)
    ci_high = np.exp(np.log(odds_ratio) + z_score * log_odds_std_error)

    # Print the results
    print(f"Odds Ratio: {odds_ratio:.2f}")
    print(f"95% Confidence Interval: {ci_low:.2f}, {ci_high:.2f}")

    return
# Calculate the confidence interval for odds ratio
OR_CIs(contingency_table_Z)
```

```
Odds Ratio: 0.71
95% Confidence Interval: 0.55, 0.91
```

- The odds ratio of 0.71 means that the odds of converting in the ad group are 29% lower (1 - 0.71 = 0.29) than in the PSA group.
- The confidence interval (0.55 to 0.91) does not include 1, which indicates that the result is statistically significant. In other words, we are 95% confident that the true odds ratio lies within this range, confirming that there is a difference in conversion odds between the groups.

15 Overall Interpretation & Insights:

- There is a statistically significant difference in conversion rates between the ad and PSA groups.
- Although the conversion rate in the ad group (2.66%) is slightly higher than in the PSA group (1.90%), the odds ratio of 0.71 suggests that users exposed to the ad have lower odds of converting than those exposed to the PSA.
- While the ad group had a higher conversion percentage, the odds ratio indicates that users

in the PSA group had a better relative chance of converting compared to the ad group, when accounting for odds rather than raw proportions.

- Optimal Range for Conversion: It seems that showing users between 100 and 500 ads is where conversion rates are most stable and relatively higher.
- Too Many Ads May Hurt: After 500 ads, the conversion rate becomes highly variable, and after 1000 ads, it drops close to zero for most users. This could indicate that showing too many ads might not be effective or could lead to ad fatigue.
- You might want to limit the number of ads shown to users, as conversion rates tend to plateau or become erratic after about 500 ads.