In [2]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
file_path = '/Users/lasyatummala/Downloads/data.csv'
data = pd.read_csv(file_path)

# Data Cleaning
# Convert date columns to datetime
data['reporting_start'] = pd.to_datetime(data['reporting_start'], errors='co
data['reporting_end'] = pd.to_datetime(data['reporting_end'], errors='coerce

# Fill missing numerical values with 0
num_columns = data.select_dtypes(include=['float64', 'int64']).columns
data[num_columns] = data[num_columns].fillna(0)

# Drop rows with missing or invalid date values
data = data.dropna(subset=['reporting_start', 'reporting_end'])

# Ensure numerical columns are of the appropriate type
data[num_columns] = data[num_columns].astype(float)

# Exploratory Analysis
# Summary statistics
print(data.describe())
```

```
              ad_id     interest1      interest2     interest3    impressions  \
count  1.143000e+03  1143.000000   1.143000e+03   1143.000000   1.143000e+03
mean   9.872611e+05    33.884514   1.180606e+05     42.474191   6.872500e+04
std    1.939928e+05    27.560263   2.670506e+05     48.987248   2.067023e+05
min    7.087460e+05     2.000000   3.000000e+00      0.000000   0.000000e+00
25%    7.776325e+05    16.000000   2.200000e+01     19.000000   1.442650e+02
50%    1.121185e+06    26.000000   3.300000e+01     27.000000   3.142000e+03
75%    1.121804e+06    32.000000   9.889400e+04     38.000000   2.786400e+04
max    1.314415e+06   120.000000   2.286228e+06    421.000000   3.052003e+06

             clicks         spent  total_conversion  approved_conversion
count   1143.000000   1143.000000       1143.000000          1143.000000
mean      11.629921     17.597760          1.439195             0.511811
std       27.347899     48.418711          3.467326             1.399146
min        0.000000      0.000000          0.000000             0.000000
25%        1.000000      0.000000          0.000000             0.000000
50%        2.000000      1.530000          1.000000             0.000000
75%        8.000000      8.540000          1.000000             1.000000
max      340.000000    639.949998         60.000000            21.000000
```

In [3]:
```
# Group data by age and gender to calculate performance metrics
demographics_performance = data.groupby(['age', 'gender']).agg({
    'impressions': 'sum',
    'clicks': 'sum',
    'spent': 'sum',
    'total_conversion': 'sum',
    'approved_conversion': 'sum'
}).reset_index()

# Calculate additional metrics
demographics_performance['CTR (%)'] = (demographics_performance['clicks'] /
demographics_performance['CPC ($)'] = demographics_performance['spent'] / (d
demographics_performance['Cost per Conversion ($)'] = demographics_performan

# Print or save the resulting dataframe
print(demographics_performance)

# Optional: Save the results to a CSV file
output_file = 'demographics_performance.csv'
demographics_performance.to_csv(output_file, index=False)
print(f"Demographics performance metrics saved to {output_file}")
```

```
    age gender  impressions  clicks  spent  total_conversion  \
0    10     11   465.079998     8.0    2.0               0.0
1    10     12    82.279999     1.0    0.0               0.0
2    10     13   577.699996    24.0   11.0               0.0
3    10     14   346.259998    15.0    3.0               0.0
4    10     15   424.770000    21.0    2.0               0.0
..   ..    ...          ...     ...    ...               ...
190  66     71    41.270000     2.0    0.0               0.0
191  66     72    10.550000     2.0    1.0               0.0
192   7     10    40.829999     3.0    1.0               0.0
193   7      8    73.260001     6.0    0.0               0.0
194   7      9    48.549999     2.0    0.0               0.0

     approved_conversion    CTR (%)   CPC ($)  Cost per Conversion ($)
0                    0.0   1.720134  0.250000               2000000.0
1                    0.0   1.215362  0.000000                     0.0
2                    0.0   4.154405  0.458333              11000000.0
3                    0.0   4.332005  0.200000               3000000.0
4                    0.0   4.943852  0.095238               2000000.0
..                   ...        ...       ...                     ...
190                  0.0   4.846135  0.000000                     0.0
191                  0.0  18.957346  0.500000               1000000.0
192                  0.0   7.347539  0.333333               1000000.0
193                  0.0   8.190008  0.000000                     0.0
194                  0.0   4.119465  0.000000                     0.0

[195 rows x 10 columns]
Demographics performance metrics saved to demographics_performance.csv
```

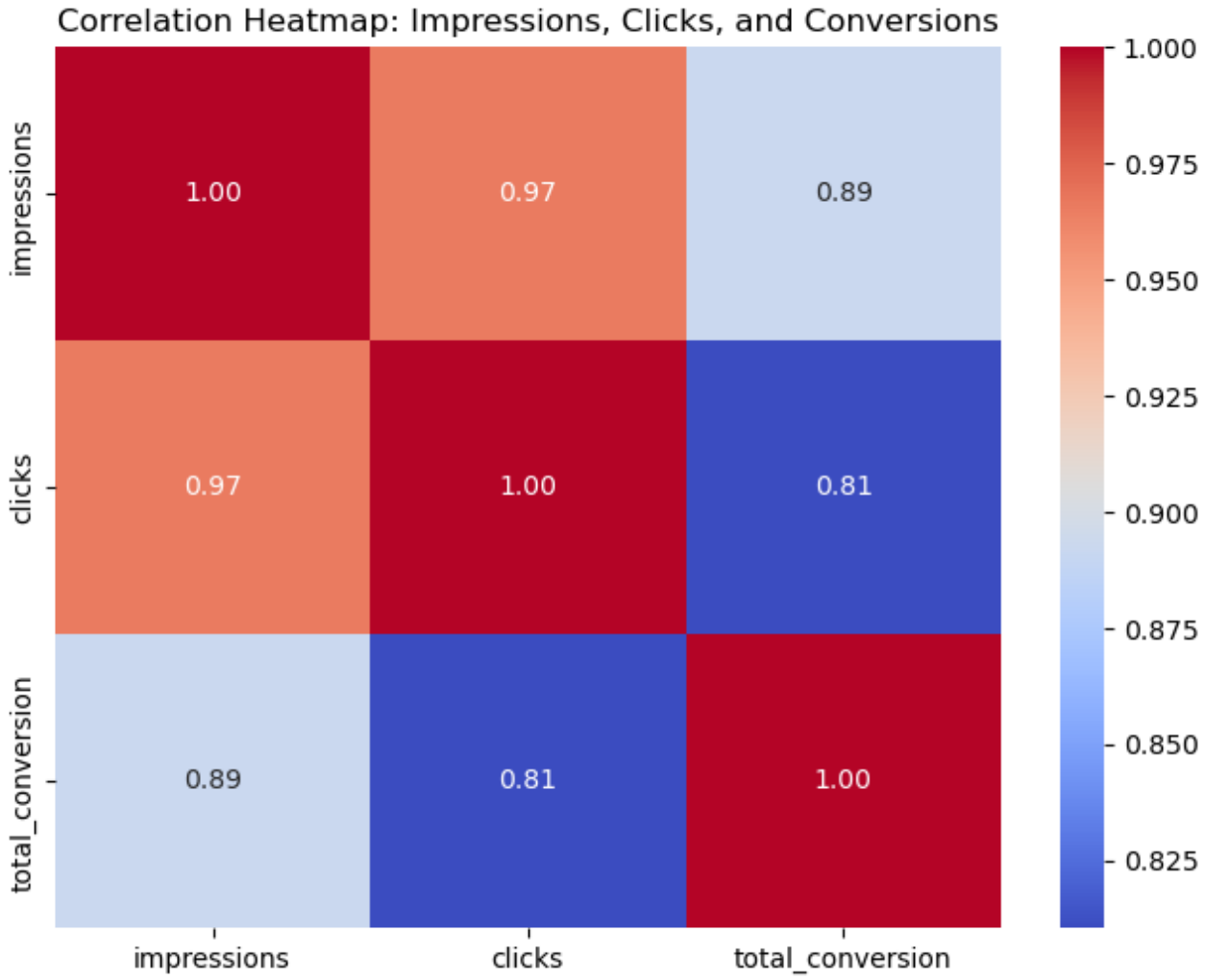
In [4]:

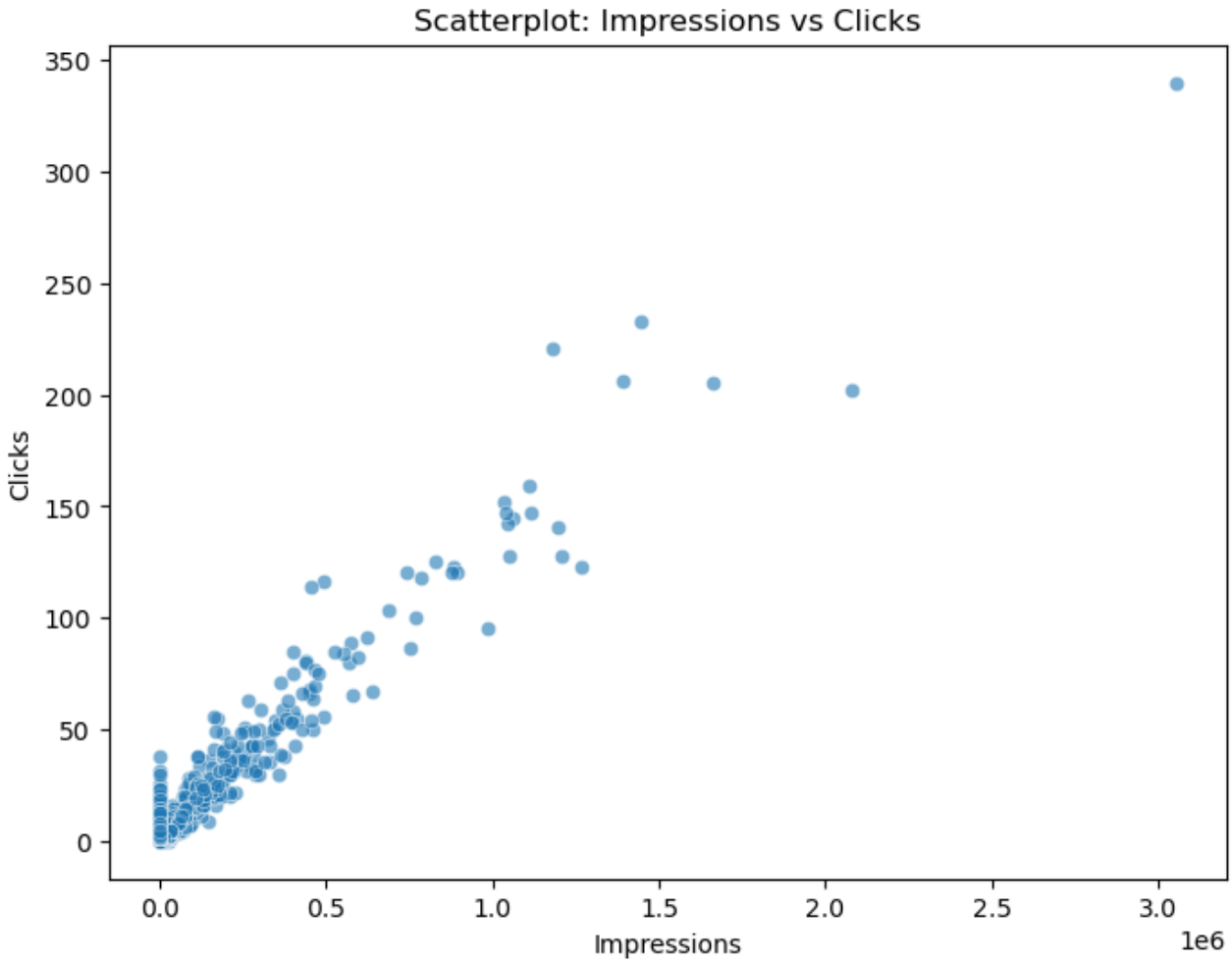
```
# Group data by age and gender to calculate cost-efficiency metrics
cost_efficiency = data.groupby(['age', 'gender']).agg({
    'spent': 'sum',
    'clicks': 'sum',
    'total_conversion': 'sum',
    'approved_conversion': 'sum'
}).reset_index()

# Calculate cost-efficiency metrics
cost_efficiency['CPC ($)'] = cost_efficiency['spent'] / (cost_efficiency['cl
cost_efficiency['Cost per Total Conversion ($)'] = cost_efficiency['spent']
cost_efficiency['Cost per Approved Conversion ($)'] = cost_efficiency['spent

# Print or save the results
print(cost_efficiency)

# Optional: Save the results to a CSV file
output_file = 'cost_efficiency_demographics.csv'
cost_efficiency.to_csv(output_file, index=False)
print(f"Cost efficiency metrics saved to {output_file}")
```

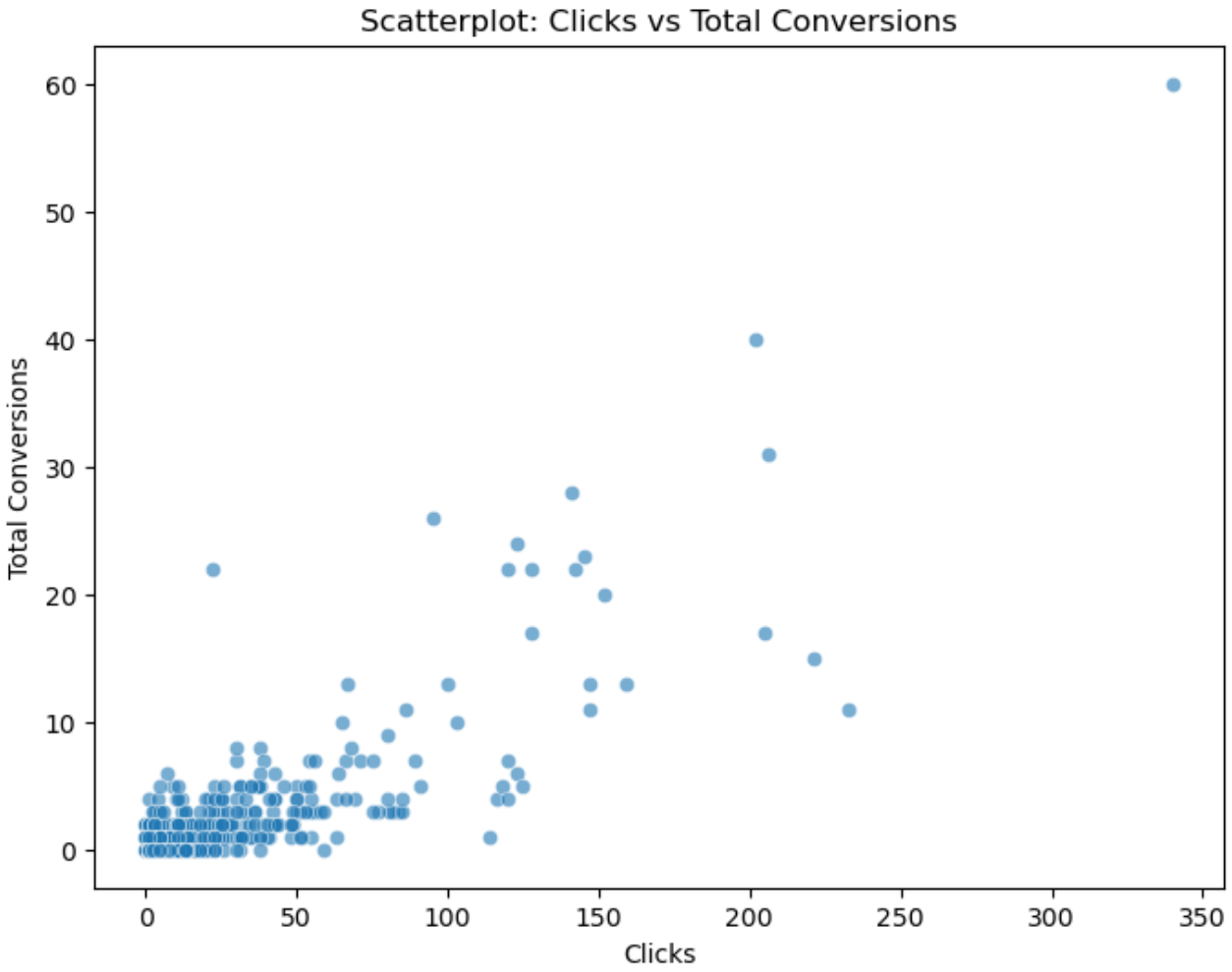
```
     age gender  spent  clicks  total_conversion  approved_conversion  \
0     10     11    2.0     8.0               0.0                  0.0
1     10     12    0.0     1.0               0.0                  0.0
2     10     13   11.0    24.0               0.0                  0.0
3     10     14    3.0    15.0               0.0                  0.0
4     10     15    2.0    21.0               0.0                  0.0
..    ..    ...    ...     ...               ...                  ...
190   66     71    0.0     2.0               0.0                  0.0
191   66     72    1.0     2.0               0.0                  0.0
192    7     10    1.0     3.0               0.0                  0.0
193    7      8    0.0     6.0               0.0                  0.0
194    7      9    0.0     2.0               0.0                  0.0

       CPC ($)  Cost per Total Conversion ($)  Cost per Approved Conversion (
$)
0    0.250000                      2000000.0                          200000
0.0
1    0.000000                            0.0
0.0
2    0.458333                     11000000.0                         1100000
0.0
3    0.200000                      3000000.0                          300000
0.0
4    0.095238                      2000000.0                          200000
0.0
..        ...                            ...
...
190  0.000000                            0.0
0.0
191  0.500000                      1000000.0                          100000
0.0
192  0.333333                      1000000.0                          100000
0.0
193  0.000000                            0.0
0.0
194  0.000000                            0.0
0.0

[195 rows x 9 columns]
Cost efficiency metrics saved to cost_efficiency_demographics.csv
```

In [5]:
```
# Convert date columns to datetime (if not already done)
data['reporting_start'] = pd.to_datetime(data['reporting_start'], errors='co
data['reporting_end'] = pd.to_datetime(data['reporting_end'], errors='coerce

# Fill missing values for conversions
data['total_conversion'] = data['total_conversion'].fillna(0)
data['approved_conversion'] = data['approved_conversion'].fillna(0)

# Calculate correlations
correlation_matrix = data[['impressions', 'clicks', 'total_conversion']].cor
print("Correlation Matrix:")
print(correlation_matrix)

# Heatmap of correlations
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap: Impressions, Clicks, and Conversions')
plt.show()

# Scatterplot: Impressions vs Clicks
plt.figure(figsize=(8, 6))
sns.scatterplot(x='impressions', y='clicks', data=data, alpha=0.6)
plt.title('Scatterplot: Impressions vs Clicks')
plt.xlabel('Impressions')
plt.ylabel('Clicks')
plt.show()

# Scatterplot: Clicks vs Total Conversions
plt.figure(figsize=(8, 6))
sns.scatterplot(x='clicks', y='total_conversion', data=data, alpha=0.6)
plt.title('Scatterplot: Clicks vs Total Conversions')
plt.xlabel('Clicks')
plt.ylabel('Total Conversions')
plt.show()
```

```
Correlation Matrix:
                 impressions    clicks  total_conversion
impressions         1.000000  0.965629          0.892113
clicks              0.965629  1.000000          0.810449
total_conversion    0.892113  0.810449          1.000000
```

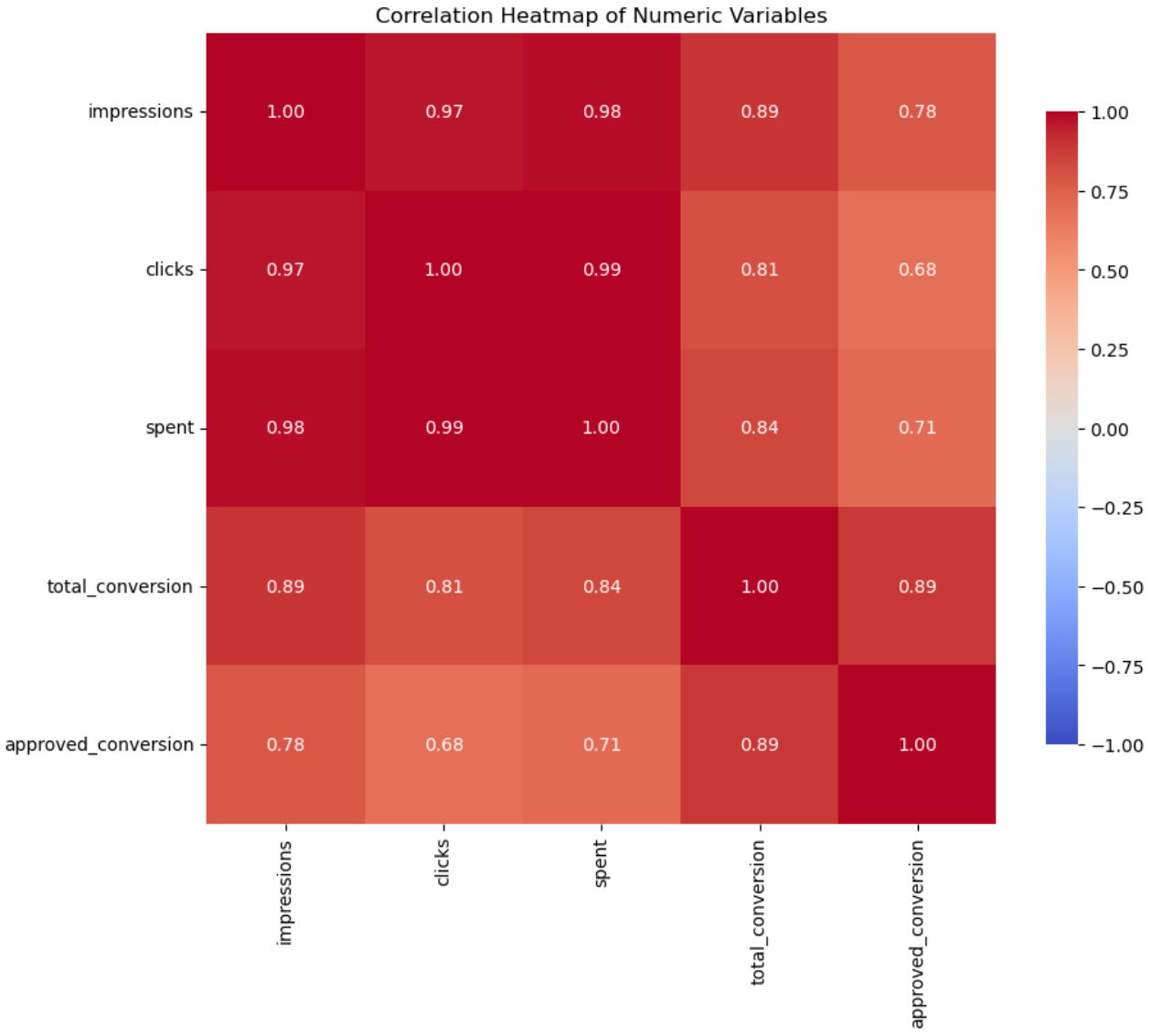

Correlation Heatmap: Impressions, Clicks, and Conversions
impressions
clicks
total_conversion
1.00
0.97
0.89
0.97
1.00
0.81
0.89
0.81
1.00
impressions
clicks
total_conversion
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825


Scatterplot: Impressions vs Clicks
Clicks
Impressions
1e6
0
50
100
150
200
250
300
350
0.0
0.5
1.0
1.5
2.0
2.5
3.0

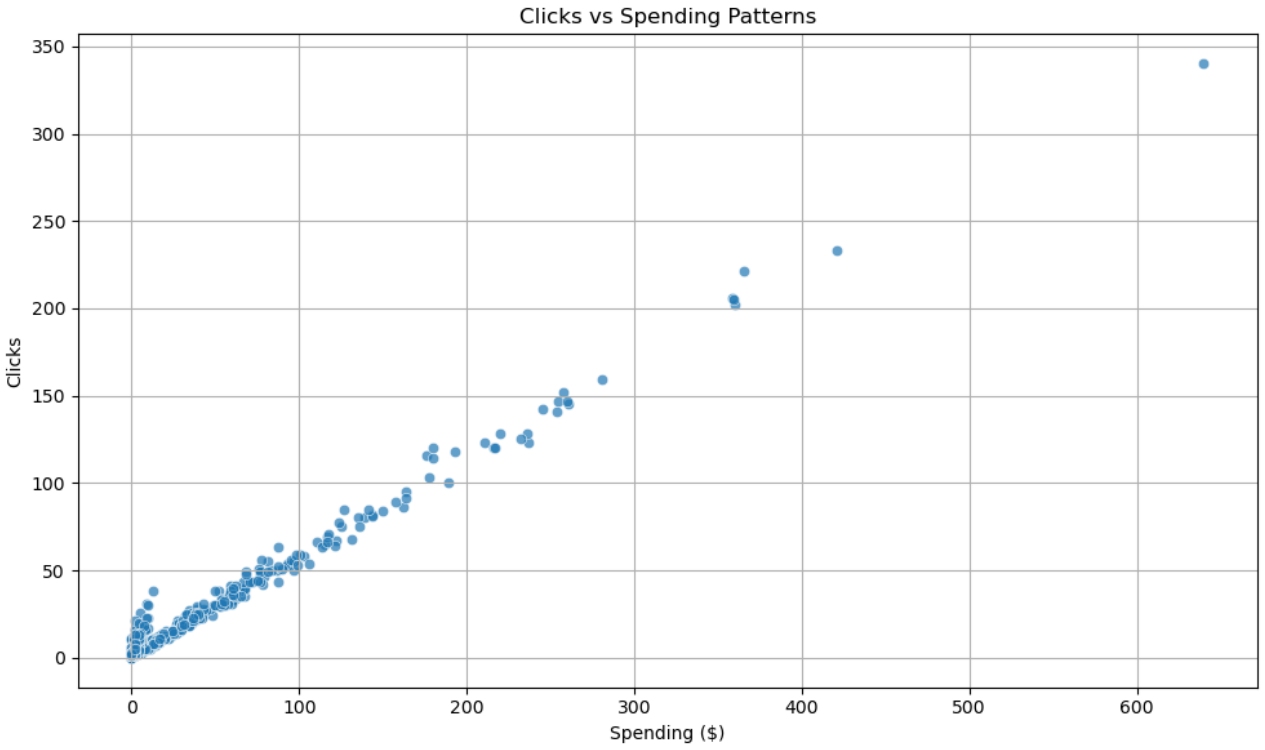


In [12]:
```
# Analyze conversion efficiency by demographics (age and gender)
conversion_efficiency = data.groupby(['age', 'gender']).agg({
    'total_conversion': 'sum',
    'approved_conversion': 'sum',
    'clicks': 'sum',
    'impressions': 'sum'
}).reset_index()

# Calculate conversion efficiency metrics
conversion_efficiency['Conversion Rate (%)'] = (conversion_efficiency['total
conversion_efficiency['Approval Rate (%)'] = (conversion_efficiency['approve
```

In [13]:
```
# Compute the correlation matrix for relevant numeric columns
correlation_matrix = data[['impressions', 'clicks', 'spent', 'total_conversi

# Visualization: Correlation Heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(
    correlation_matrix,
    annot=True,  # Display correlation values
    fmt='.2f',  # Format the annotation to 2 decimal places
    cmap='coolwarm',  # Color map for the heatmap
    square=True,  # Keep the heatmap squares
    cbar_kws={'shrink': 0.8},  # Adjust color bar size
    vmin=-1, vmax=1  # Set fixed color scale for comparison
)
plt.title('Correlation Heatmap of Numeric Variables')
plt.show()
```



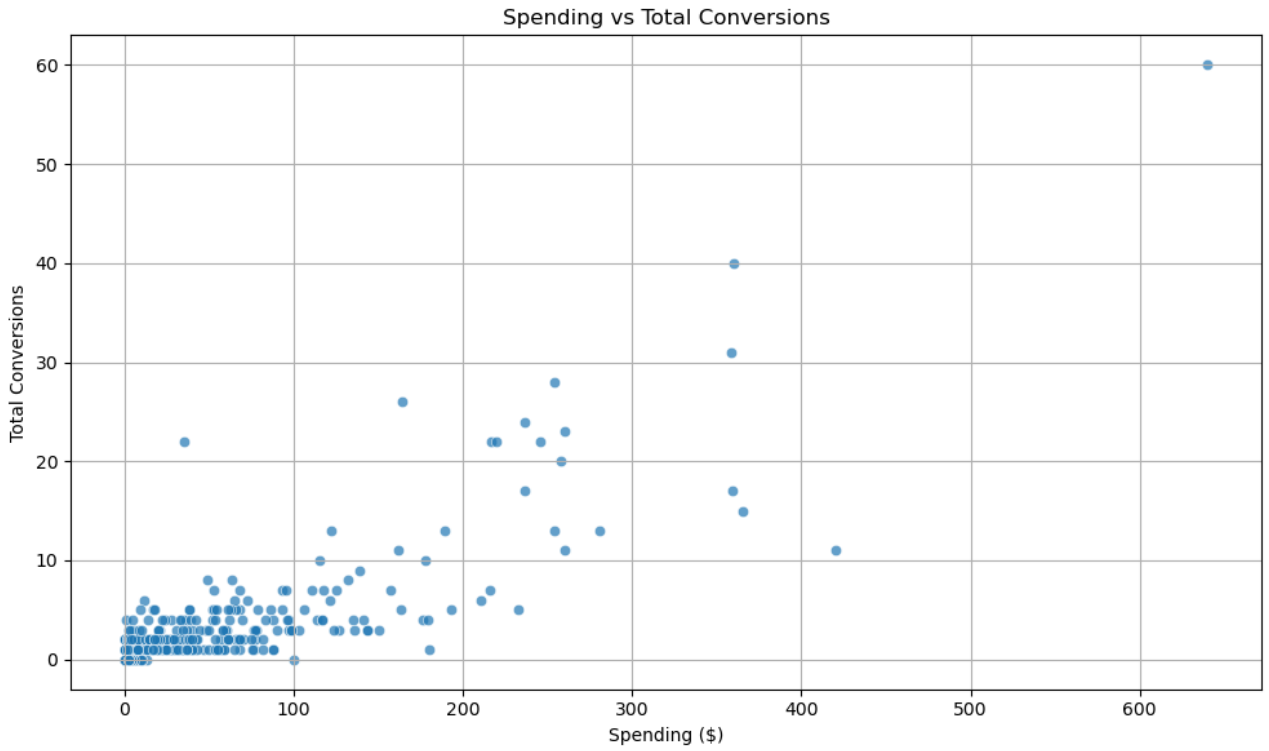
In [14]:
```
# Scatterplot: Clicks vs Spending
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=data,
    x='spent',
    y='clicks',
    alpha=0.7
)
plt.title('Clicks vs Spending Patterns')
plt.xlabel('Spending ($)')
plt.ylabel('Clicks')
plt.grid(True)
plt.tight_layout()
plt.show()

# Calculate the correlation between clicks and spending
clicks_spent_correlation = data[['spent', 'clicks']].corr().iloc[0, 1]
print(f"Correlation between Spending and Clicks: {clicks_spent_correlation:.
```

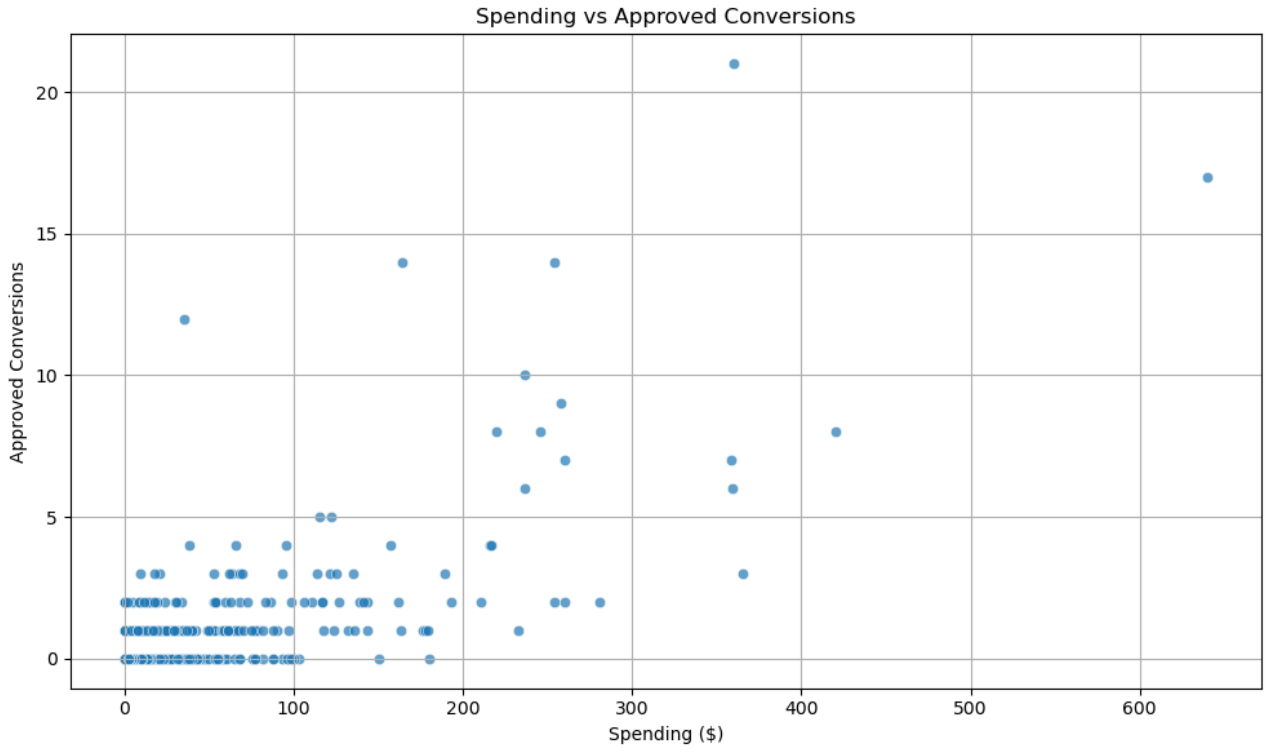



```
Correlation between Spending and Clicks: 0.99
```

In [15]:

```
# Scatterplot: Spending vs Total Conversions
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=data,
    x='spent',
    y='total_conversion',
    alpha=0.7
)
plt.title('Spending vs Total Conversions')
plt.xlabel('Spending ($)')
plt.ylabel('Total Conversions')
plt.grid(True)
plt.tight_layout()
plt.show()

# Scatterplot: Spending vs Approved Conversions
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=data,
    x='spent',
    y='approved_conversion',
    alpha=0.7
)
plt.title('Spending vs Approved Conversions')
plt.xlabel('Spending ($)')
plt.ylabel('Approved Conversions')
plt.grid(True)
plt.tight_layout()
plt.show()

# Correlation analysis
conversion_correlation = data[['spent', 'total_conversion', 'approved_conver

# Display correlation matrix
print("Correlation Matrix for Spending and Conversions:")
print(conversion_correlation)
```





Correlation Matrix for Spending and Conversions:

	spent	total_conversion	approved_conversion
spent	1.000000	0.836366	0.706719
total_conversion	0.836366	1.000000	0.889515
approved_conversion	0.706719	0.889515	1.000000

In [20]:
```
# Group data by age to analyze spending efficiency
age_group_efficiency = data.groupby('age').agg({
    'spent': 'sum',
    'total_conversion': 'sum',
    'approved_conversion': 'sum'
}).reset_index()

# Calculate spending efficiency metrics
age_group_efficiency['Cost per Total Conversion ($)'] = age_group_efficiency
age_group_efficiency['Cost per Approved Conversion ($)'] = age_group_efficie

# Display the DataFrame
print(age_group_efficiency)

# Optional: Save to CSV
age_group_efficiency.to_csv('age_group_spending_efficiency.csv', index=False
print("Age group spending efficiency data saved to 'age_group_spending_effic

# Visualization: Cost per Total Conversion by Age Group
plt.figure(figsize=(12, 6))
sns.barplot(
    data=age_group_efficiency,
    x='age',
    y='Cost per Total Conversion ($)',
    ci=None
)
plt.title('Spending Efficiency by Age Group (Cost per Total Conversion)')
plt.xlabel('Age Group')
plt.ylabel('Cost per Total Conversion ($)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Visualization: Cost per Approved Conversion by Age Group
plt.figure(figsize=(12, 6))
sns.barplot(
    data=age_group_efficiency,
    x='age',
    y='Cost per Approved Conversion ($)',
    ci=None
)
plt.title('Spending Efficiency by Age Group (Cost per Approved Conversion)')
plt.xlabel('Age Group')
plt.ylabel('Cost per Approved Conversion ($)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
      age        spent  total_conversion  approved_conversion  \
0      10    24.000000               0.0                  0.0
1     100     9.000000               0.0                  0.0
2     101    25.000000               0.0                  0.0
3     102     7.000000               0.0                  0.0
4     103     5.000000               0.0                  0.0
5     104     8.000000               0.0                  0.0
6     105     6.000000               0.0                  0.0
```

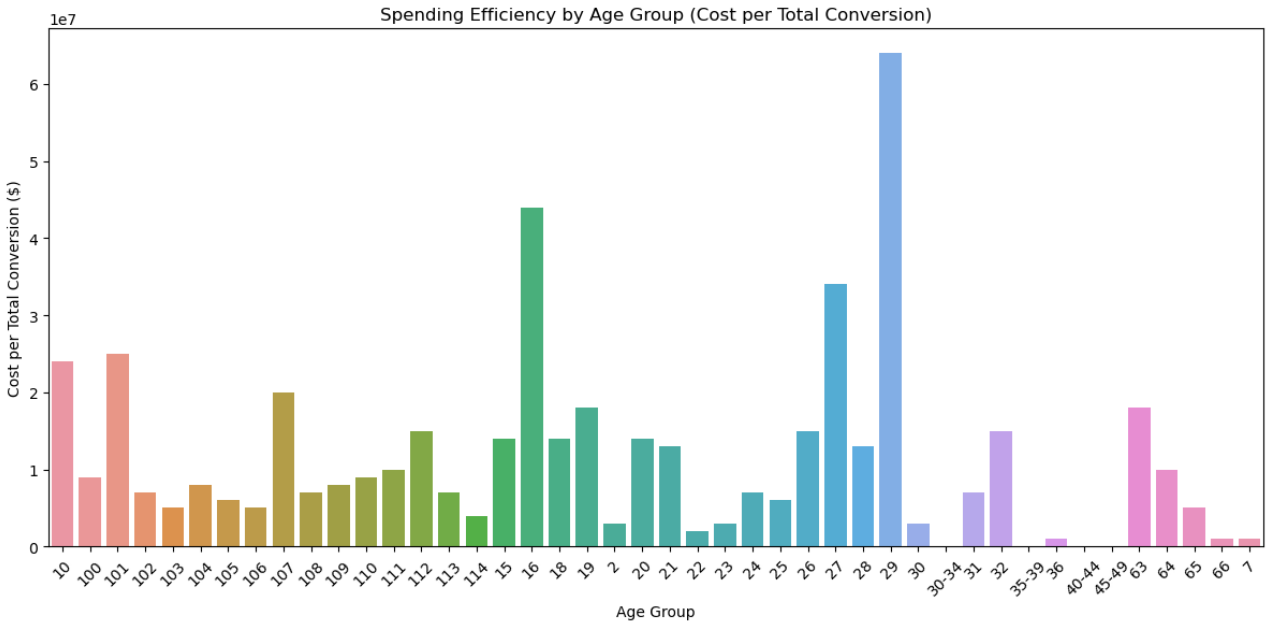
7	106	5.000000	0.0	0.0
8	107	20.000000	0.0	0.0
9	108	7.000000	0.0	0.0
10	109	8.000000	0.0	0.0
11	110	9.000000	0.0	0.0
12	111	10.000000	0.0	0.0
13	112	15.000000	0.0	0.0
14	113	7.000000	0.0	0.0
15	114	4.000000	0.0	0.0
16	15	14.000000	0.0	0.0
17	16	44.000000	0.0	0.0
18	18	14.000000	0.0	0.0
19	19	18.000000	0.0	0.0
20	2	3.000000	0.0	0.0
21	20	14.000000	0.0	0.0
22	21	13.000000	0.0	0.0
23	22	2.000000	0.0	0.0
24	23	3.000000	0.0	0.0
25	24	7.000000	0.0	0.0
26	25	6.000000	0.0	0.0
27	26	15.000000	0.0	0.0
28	27	34.000000	0.0	0.0
29	28	13.000000	0.0	0.0
30	29	64.000000	0.0	0.0
31	30	3.000000	0.0	0.0
32	30-34	7693.219994	890.0	328.0
33	31	7.000000	0.0	0.0
34	32	15.000000	0.0	0.0
35	35-39	5145.520004	357.0	129.0
36	36	1.000000	0.0	0.0
37	40-44	4337.629999	235.0	82.0
38	45-49	2443.870000	163.0	46.0
39	63	18.000000	0.0	0.0
40	64	10.000000	0.0	0.0
41	65	5.000000	0.0	0.0
42	66	1.000000	0.0	0.0
43	7	1.000000	0.0	0.0

	Cost per Total Conversion ($)	Cost per Approved Conversion ($)
0	2.400000e+07	2.400000e+07
1	9.000000e+06	9.000000e+06
2	2.500000e+07	2.500000e+07
3	7.000000e+06	7.000000e+06
4	5.000000e+06	5.000000e+06
5	8.000000e+06	8.000000e+06
6	6.000000e+06	6.000000e+06
7	5.000000e+06	5.000000e+06
8	2.000000e+07	2.000000e+07
9	7.000000e+06	7.000000e+06
10	8.000000e+06	8.000000e+06
11	9.000000e+06	9.000000e+06
12	1.000000e+07	1.000000e+07
13	1.500000e+07	1.500000e+07
14	7.000000e+06	7.000000e+06
15	4.000000e+06	4.000000e+06
16	1.400000e+07	1.400000e+07
17	4.400000e+07	4.400000e+07

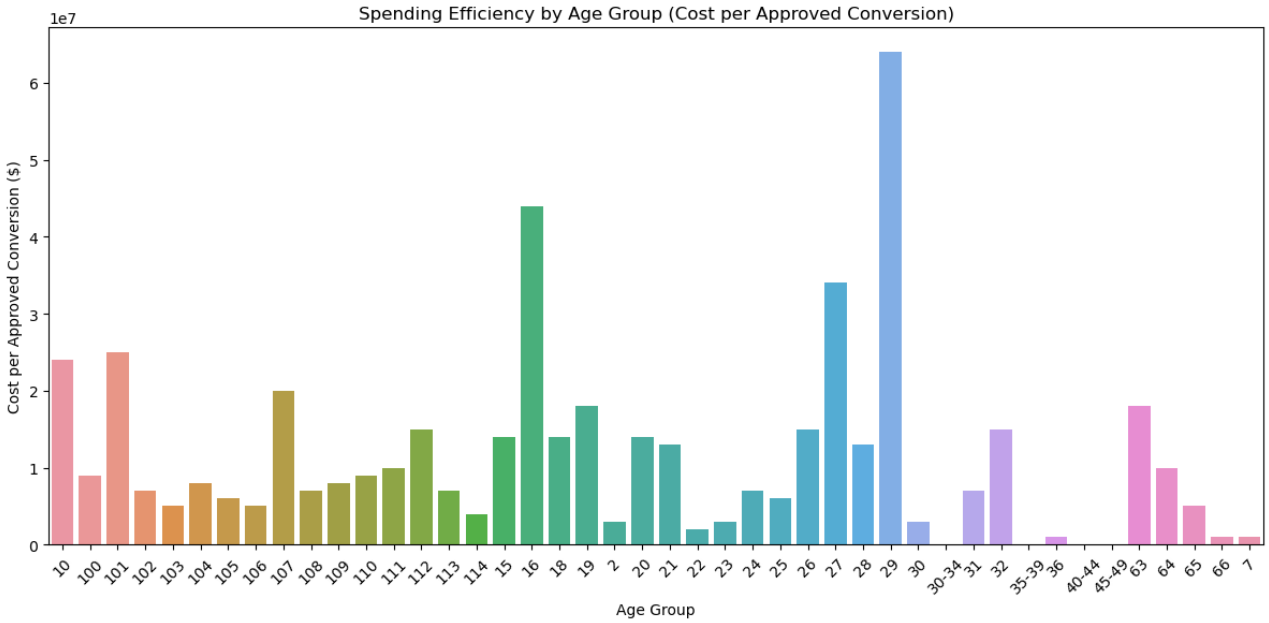
18	1.400000e+07	1.400000e+07
19	1.800000e+07	1.800000e+07
20	3.000000e+06	3.000000e+06
21	1.400000e+07	1.400000e+07
22	1.300000e+07	1.300000e+07
23	2.000000e+06	2.000000e+06
24	3.000000e+06	3.000000e+06
25	7.000000e+06	7.000000e+06
26	6.000000e+06	6.000000e+06
27	1.500000e+07	1.500000e+07
28	3.400000e+07	3.400000e+07
29	1.300000e+07	1.300000e+07
30	6.400000e+07	6.400000e+07
31	3.000000e+06	3.000000e+06
32	8.644067e+00	2.345494e+01
33	7.000000e+06	7.000000e+06
34	1.500000e+07	1.500000e+07
35	1.441322e+01	3.988775e+01
36	1.000000e+06	1.000000e+06
37	1.845800e+01	5.289793e+01
38	1.499307e+01	5.312761e+01
39	1.800000e+07	1.800000e+07
40	1.000000e+07	1.000000e+07
41	5.000000e+06	5.000000e+06
42	1.000000e+06	1.000000e+06
43	1.000000e+06	1.000000e+06

```
Age group spending efficiency data saved to 'age_group_spending_efficiency.csv'.
```

```
/var/folders/fp/drm5pw1d13x1n4wvd2q14v0m0000gn/T/ipykernel_75272/1085714078.py:21: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(
```



```
/var/folders/fp/drm5pw1d13x1n4wvd2q14v0m0000gn/T/ipykernel_75272/1085714078.
py:36: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(
```



In [ ]:

In [ ]:

In [ ]: