In [2]:

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

Load the dataset

In [9]:

```
file_path = 'C:/Users/wolfr/OneDrive/Desktop/Logeshwaran_WorkSpace/Instagram_Influencer/Influencer.csv'
data = pd.read_csv(file_path)
```

Function to remove unwanted symbols

In [10]:

```
# Function to remove unwanted symbols
def clean_symbols(text):
    return text.replace('/', '').replace('\\', '').strip()
```

Function to convert shorthand notations like 'm', 'k', and 'b' to full numerical values

In [11]:

```
def shorthand_to_number(value):
    if 'b' in value:
        return float(value.replace('b', '')) * 1_000_000_000
    elif 'm' in value:
        return float(value.replace('m', '')) * 1_000_000
    elif 'k' in value:
        return float(value.replace('k', '')) * 1_000
    else:
        try:
            return float(value)
        except ValueError:
            return None  # Return None if conversion is not possible
```

Applying cleaning functions to relevant columns

In [12]:

```
data['Channel Info'] = data['Channel Info'].apply(clean_symbols)
data['Followers'] = data['Followers'].apply(shorthand_to_number)
data['Total Likes'] = data['Total Likes'].apply(shorthand_to_number)
data['New Post Avg. Likes'] = data['New Post Avg. Likes'].apply(shorthand_to_number)
data['Avg. Likes'] = data['Avg. Likes'].apply(shorthand_to_number)
data['Posts'] = data['Posts'].apply(shorthand_to_number)
```

Remove rows where 'Country Or Region' is blank

In [13]:

```
data.dropna(subset=['Country Or Region'], inplace=True)
```

Save the cleaned dataset to a new CSV file

In [14]:

```
cleaned_file_path = 'C:/Users/wolfr/OneDrive/Desktop/Logeshwaran_WorkSpace/Instagram_Influencer/Cleaned_Influencer_Data.csv'

data.to_csv(cleaned_file_path, index=False)
print("Data cleaned and saved successfully.")
```

```
Data cleaned and saved successfully.
```

Reload the cleaned data for analysis

In [15]:

```
data = pd.read_csv(cleaned_file_path)
```

1. Check for correlated features

Filter out non-numeric data before computing the correlation matrix

In [16]:

```
numeric_data = data.select_dtypes(include=[np.number])  # Only include numeric columns
correlation_matrix = numeric_data.corr()
print("Correlation matrix:\n", correlation_matrix)
```

```
Correlation matrix:
                        Rank  Influence Score  Followers  Avg. Likes  \
Rank                1.000000        -0.482216  -0.703721   -0.454651
Influence Score    -0.482216         1.000000   0.442241    0.208166
Followers          -0.703721         0.442241   1.000000    0.607207
Avg. Likes         -0.454651         0.208166   0.607207    1.000000
Posts              -0.030802         0.174080   0.057784   -0.321193
60-Day Eng Rate     0.000530        -0.114907  -0.098256    0.542864
New Post Avg. Likes -0.402517         0.162473   0.465740    0.846725
Total Likes         -0.467106         0.295035   0.693586    0.669669

                       Posts  60-Day Eng Rate  New Post Avg. Likes  \
Rank                -0.030802         0.000530            -0.402517
Influence Score      0.174080        -0.114907             0.162473
Followers            0.057784        -0.098256             0.465740
Avg. Likes          -0.321193         0.542864             0.846725
Posts                1.000000        -0.370828            -0.227671
60-Day Eng Rate     -0.370828         1.000000             0.702672
New Post Avg. Likes -0.227671         0.702672             1.000000
Total Likes          0.194077         0.108434             0.613992

                    Total Likes
Rank                  -0.467106
Influence Score        0.295035
Followers              0.693586
Avg. Likes             0.669669
Posts                  0.194077
60-Day Eng Rate        0.108434
New Post Avg. Likes    0.613992
Total Likes            1.000000
```

Find the highest correlated pairs (excluding self-correlation of 1)

In [17]:

```
sorted_pairs = correlation_matrix.unstack().sort_values(kind="quicksort", ascending=Fals
e)
strong_pairs = sorted_pairs[(sorted_pairs < 1) & (sorted_pairs > 0.5)]
print("Highly correlated pairs:\n", strong_pairs)
```

```
Highly correlated pairs:
 Avg. Likes           New Post Avg. Likes    0.846725
New Post Avg. Likes  Avg. Likes             0.846725
60-Day Eng Rate      New Post Avg. Likes    0.702672
New Post Avg. Likes  60-Day Eng Rate        0.702672
Total Likes          Followers              0.693586
Followers            Total Likes            0.693586
Total Likes          Avg. Likes             0.669669
Avg. Likes           Total Likes            0.669669
New Post Avg. Likes  Total Likes            0.613992
Total Likes          New Post Avg. Likes    0.613992
Followers            Avg. Likes             0.607207
Avg. Likes           Followers              0.607207
60-Day Eng Rate      Avg. Likes             0.542864
Avg. Likes           60-Day Eng Rate        0.542864
dtype: float64
```

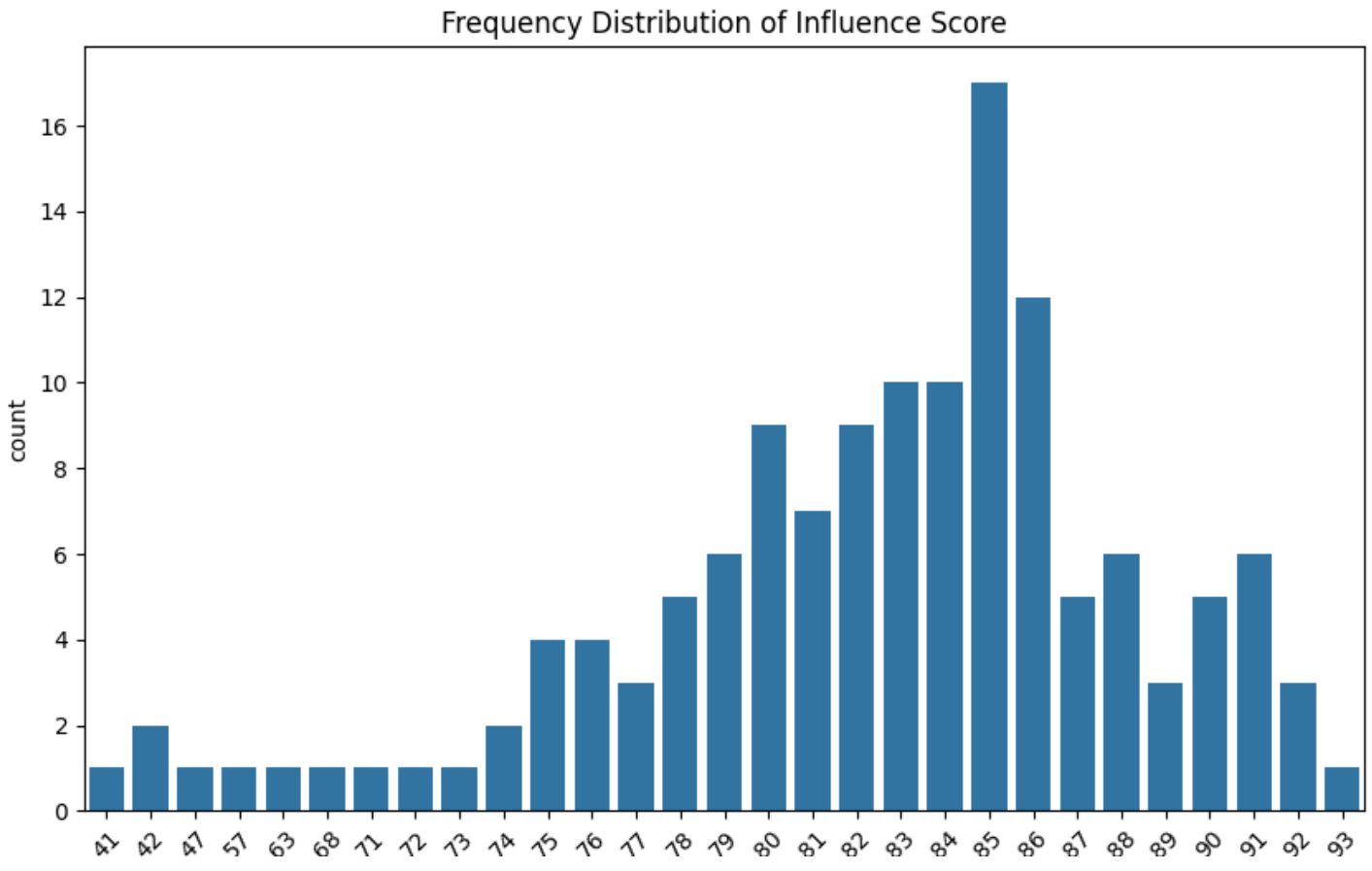
2. Frequency distribution of specified features

In [18]:

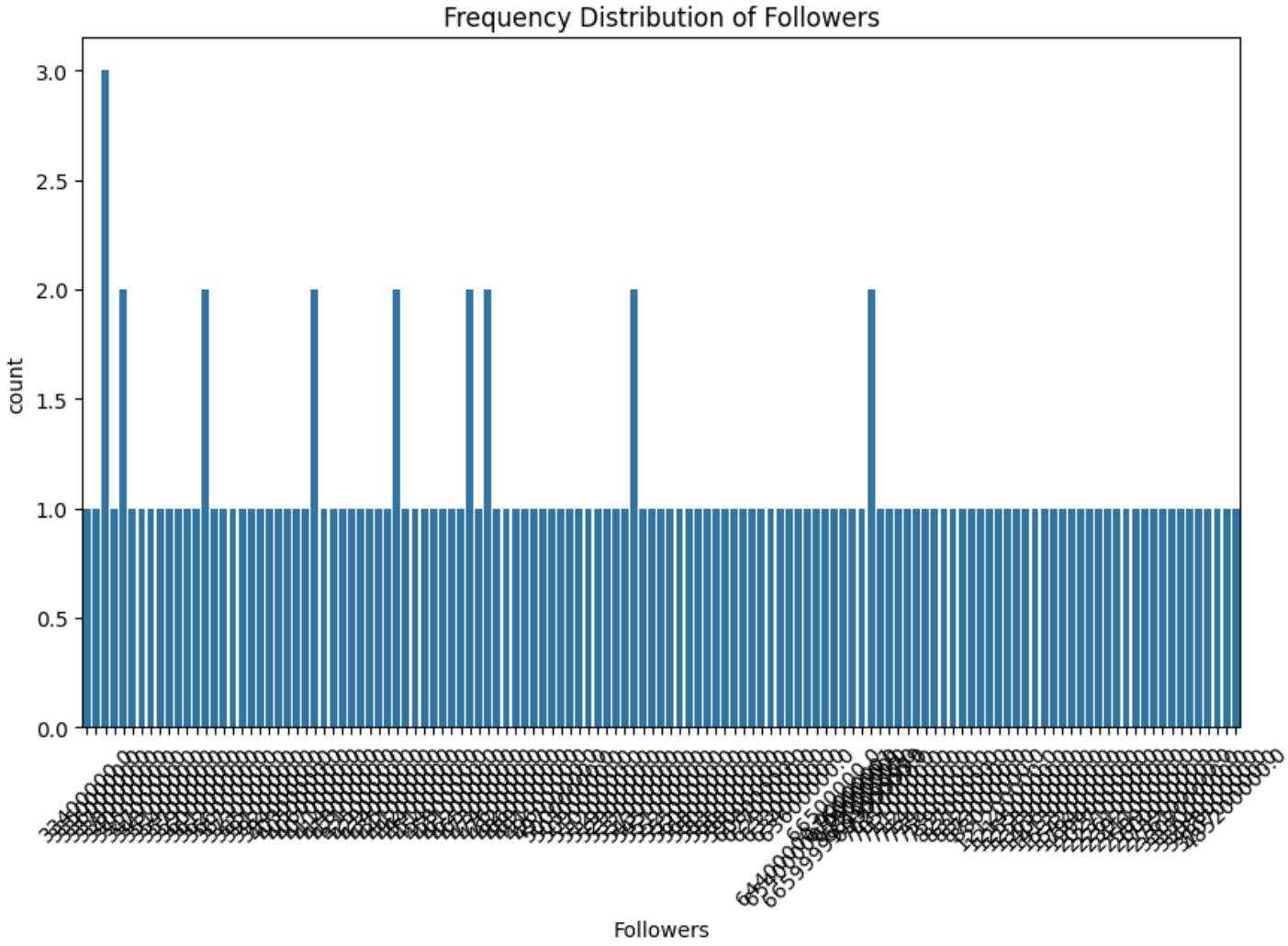
```
def plot_distribution(column, title):
    plt.figure(figsize=(10, 6))
    sns.countplot(x=column, data=data)
    plt.title(f'Frequency Distribution of {title}')
    plt.xticks(rotation=45)
    plt.show()
```

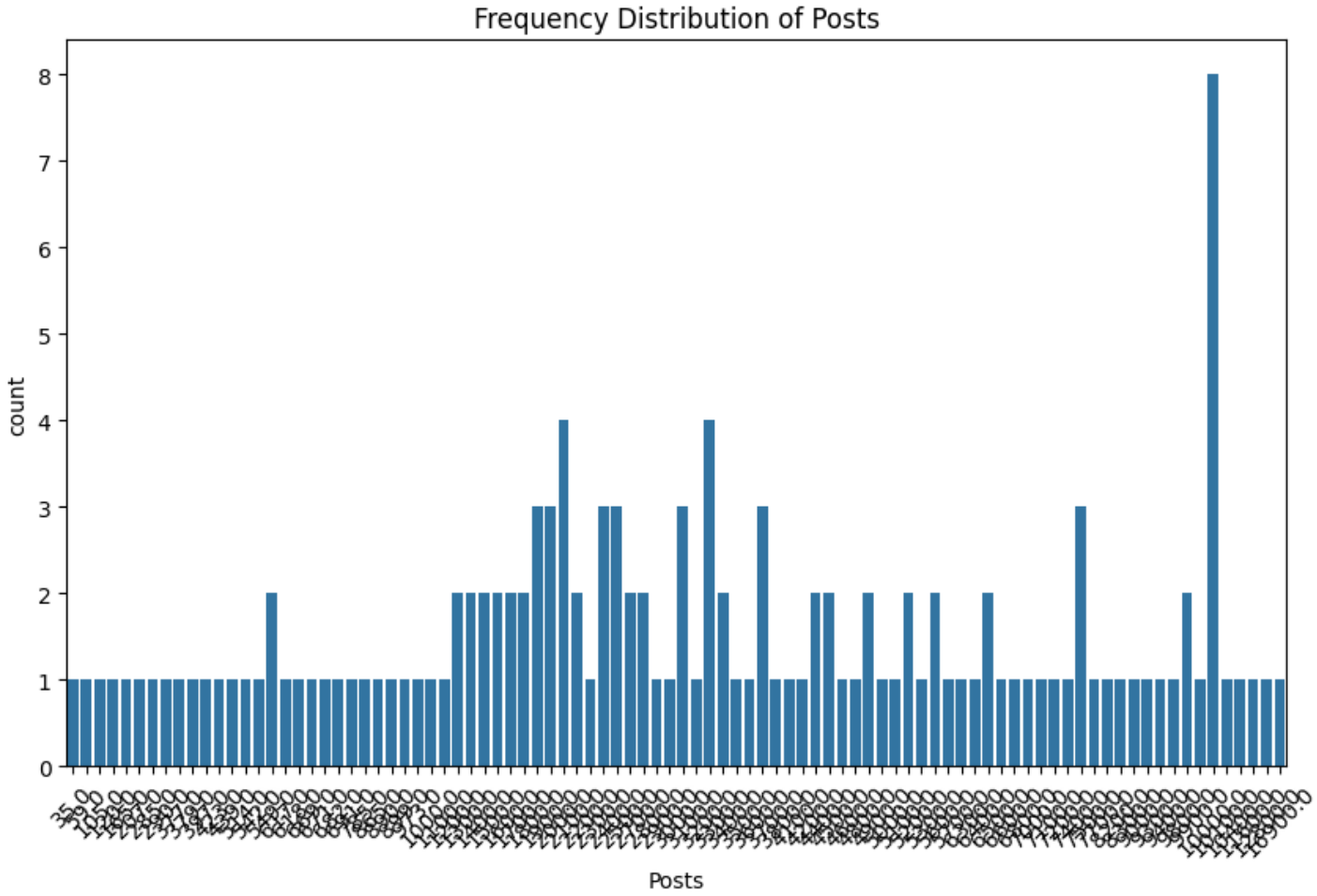
In [19]:

```
plot_distribution('Influence Score', 'Influence Score')
plot_distribution('Followers', 'Followers')
plot_distribution('Posts', 'Posts')
```



Influence Score



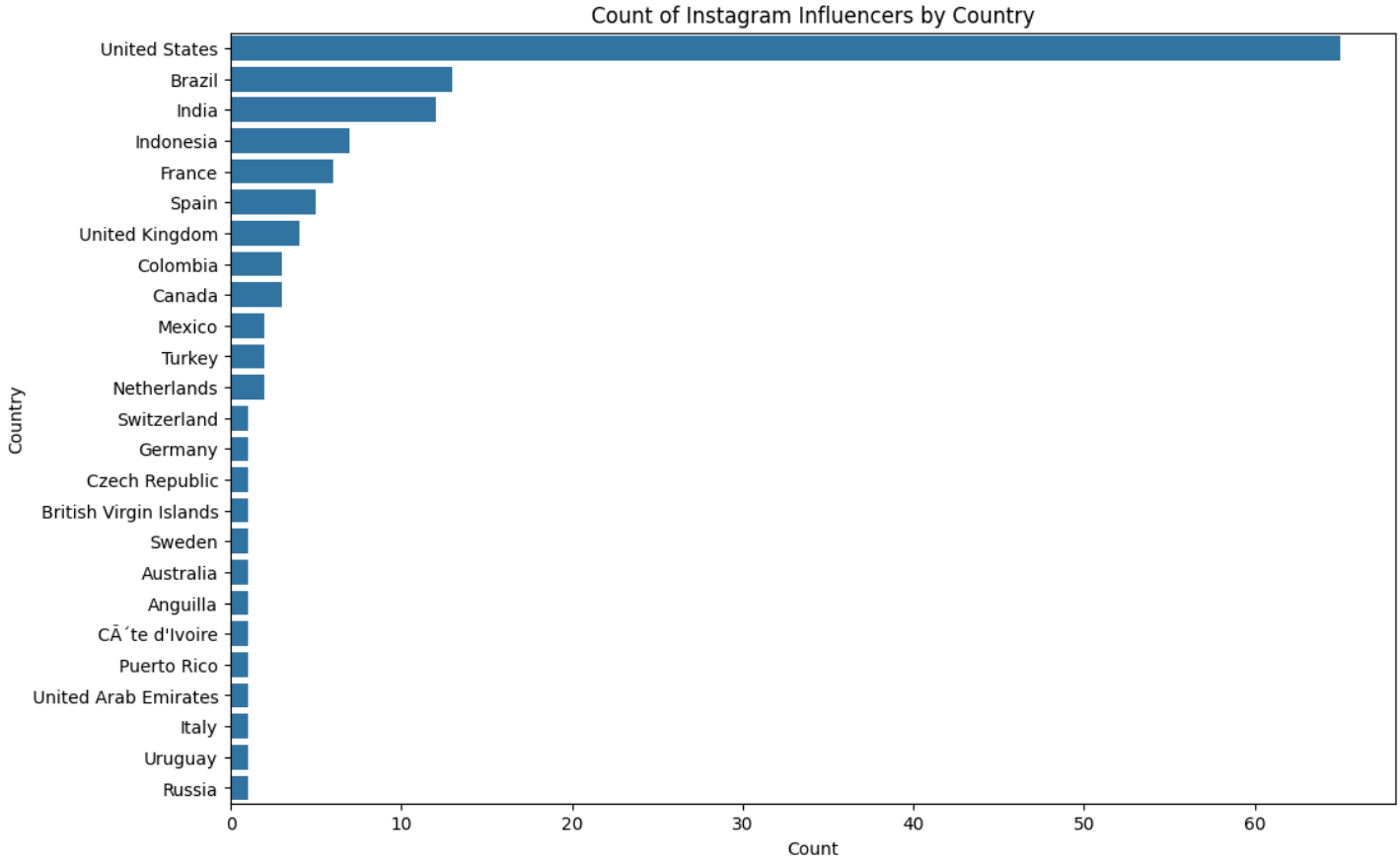


2. Which country houses the highest number of Instagram

3. Which country houses the highest number of Instagram Influencers?

In [20]:

```
plt.figure(figsize=(12, 8))
sns.countplot(y='Country Or Region', data=data, order=data['Country Or Region'].value_cou
nts().index)
plt.title('Count of Instagram Influencers by Country')
plt.xlabel('Count')
plt.ylabel('Country')
plt.show()
```



4. Top 10 influencers based on various features

In [21]:

```
def top_influencers(feature, n=10):
    top_infs = data.sort_values(by=feature, ascending=False).head(n)
    print(f"Top 10 influencers based on {feature}:\n", top_infs[['Channel Info', feature
]])

top_influencers('Followers')
top_influencers('Avg. Likes')
top_influencers('Total Likes')
```

```
Top 10 influencers based on Followers:
       Channel Info    Followers
0        cristiano  485200000.0
1      kyliejenner  370700000.0
2      selenagomez  348800000.0
3          therock  339400000.0
4     arianagrande  333000000.0
5    kimkardashian  330700000.0
6          beyonce  276100000.0
7  khloekardashian  273900000.0
8     justinbieber  260000000.0
9    kendalljenner  258900000.0
Top 10 influencers based on Avg. Likes:
          Channel Info  Avg. Likes
```

```
0            cristiano   8700000.0
1          kyliejenner   8200000.0
2          selenagomez   6100000.0
18             zendaya   5900000.0
9         kendalljenner   5500000.0
85                zayn   4700000.0
73               adele   4700000.0
82         harrystyles   4700000.0
56    milliebobbybrown   4100000.0
41  bts.bighitofficial   4100000.0
Top 10 influencers based on Total Likes:
      Channel Info   Total Likes
1      kyliejenner  5.740000e+10
0        cristiano  2.910000e+10
18         zendaya  2.080000e+10
5    kimkardashian  1.980000e+10
4     arianagrande  1.850000e+10
21      badgalriri  1.800000e+10
16        neymarjr  1.440000e+10
8     justinbieber  1.400000e+10
14      nickiminaj  1.290000e+10
3           therock  1.260000e+10
```

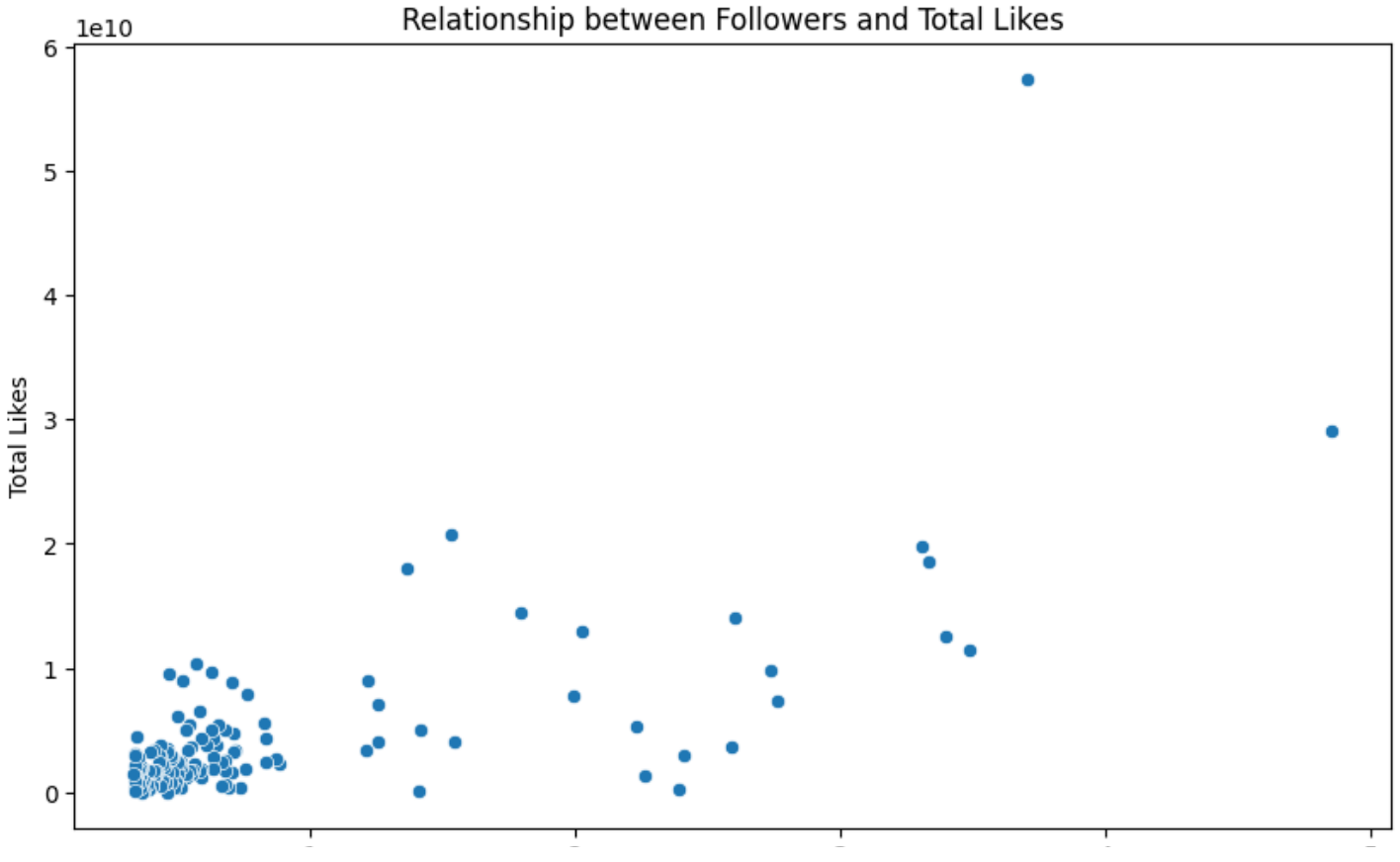
5. Relationship between pairs of features

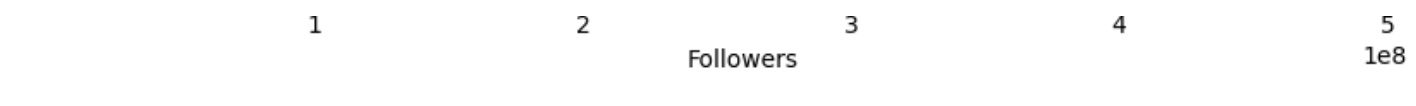
In [22]:

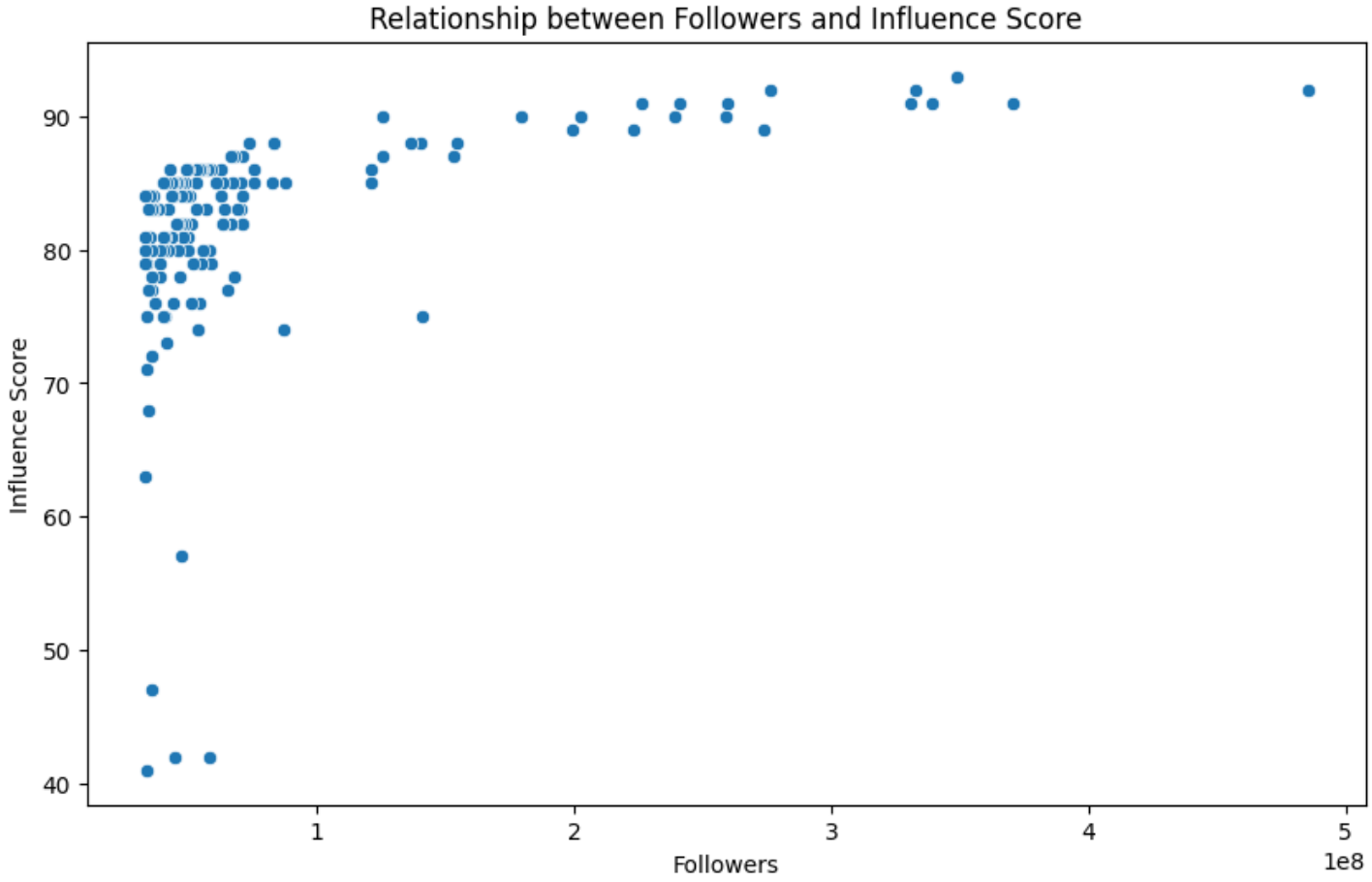
```
def plot_relationship(x, y):
    plt.figure(figsize=(10, 6))
    sns.scatterplot(x=x, y=y, data=data)
    plt.title(f'Relationship between {x} and {y}')
    plt.xlabel(x)
    plt.ylabel(y)
    plt.show()
```

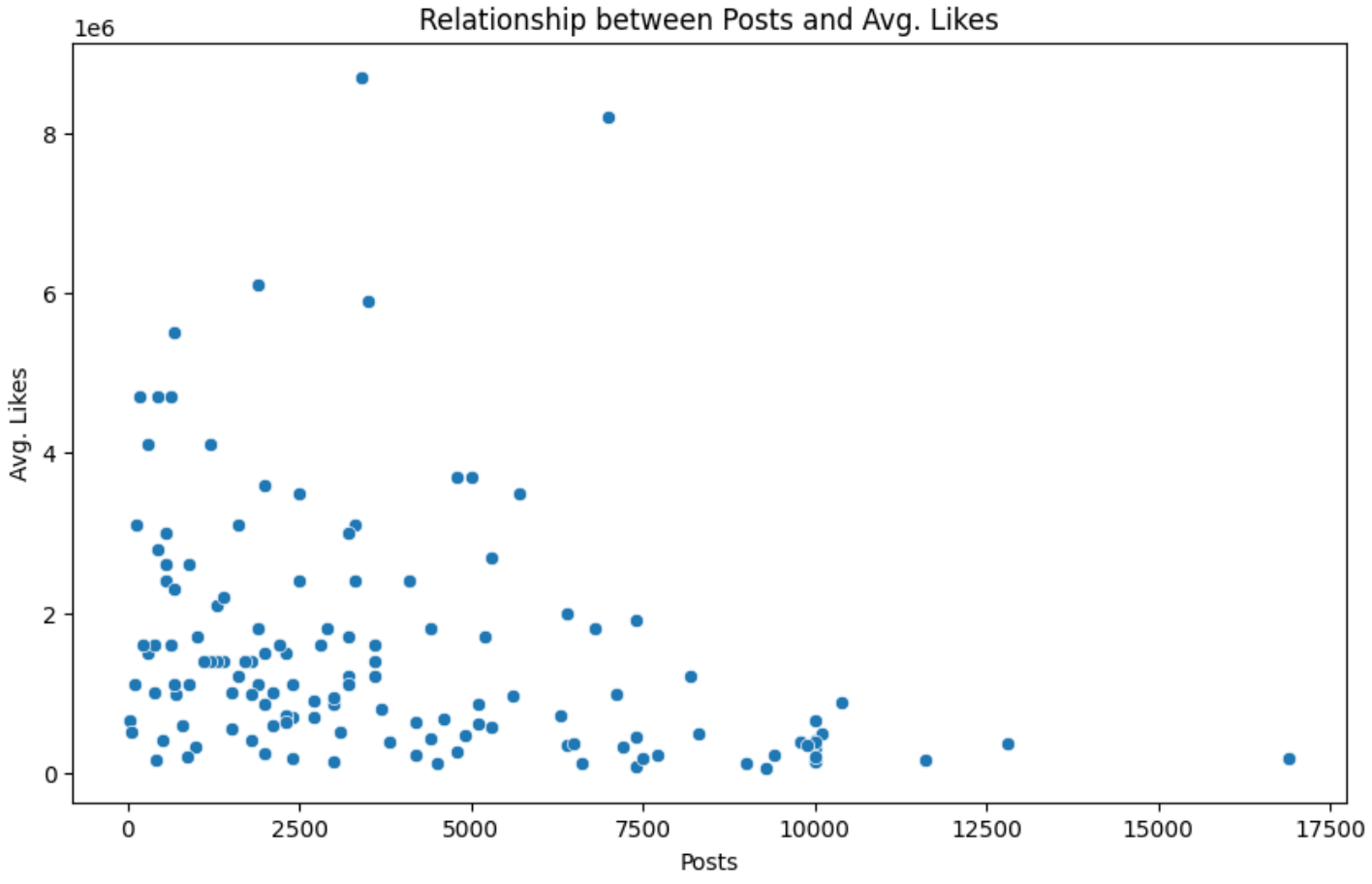
In [23]:

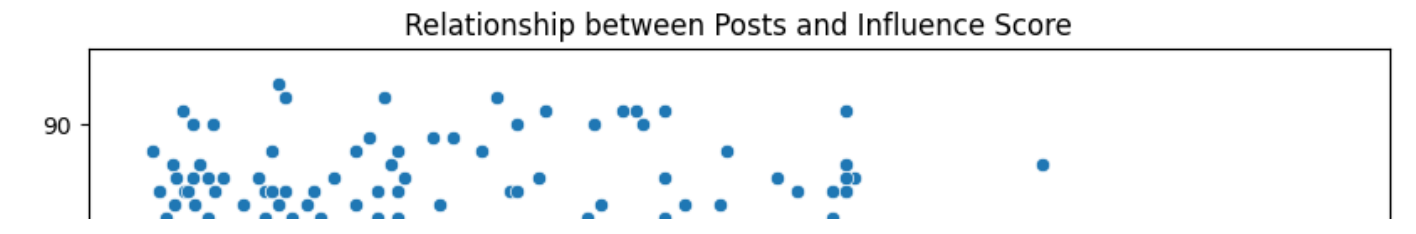
```
plot_relationship('Followers', 'Total Likes')
plot_relationship('Followers', 'Influence Score')
plot_relationship('Posts', 'Avg. Likes')
plot_relationship('Posts', 'Influence Score')
```

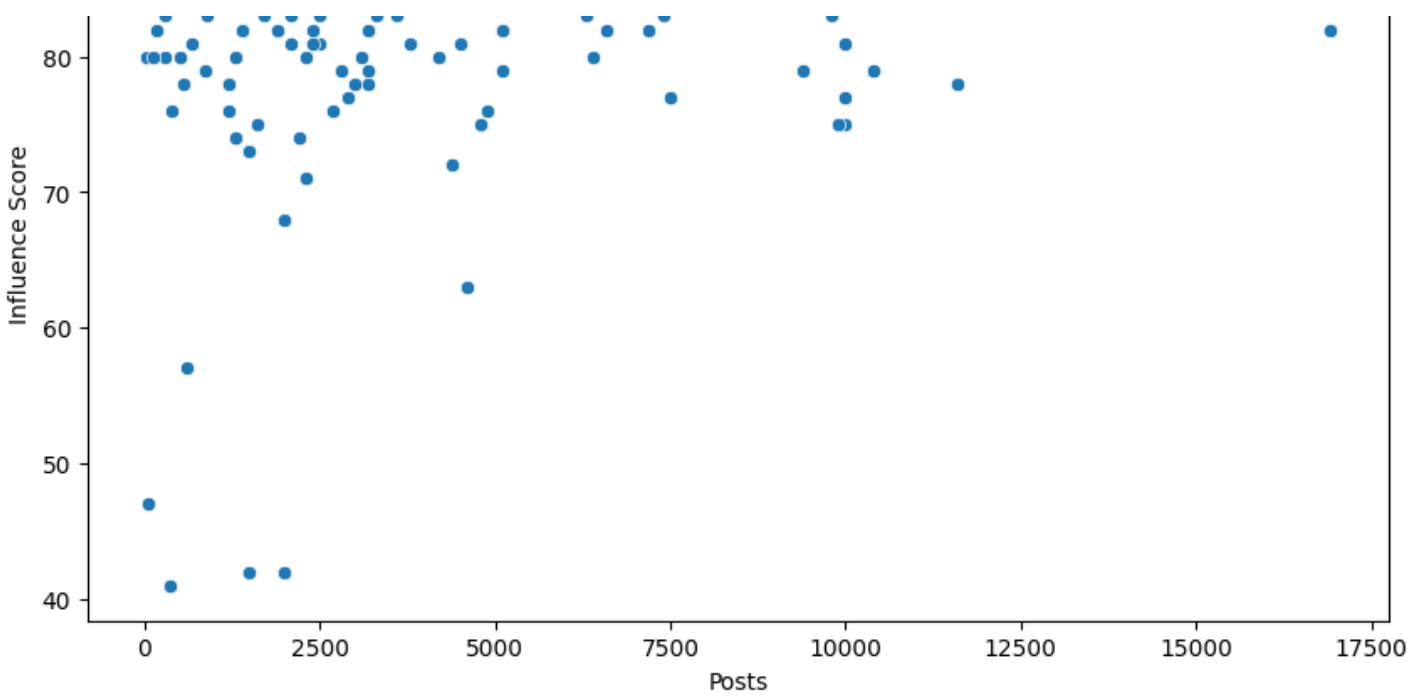



1
2
3
4
5
Followers
1e8


Relationship between Followers and Influence Score
Influence Score
90
80
70
60
50
40
1
2
3
4
5
Followers
1e8


Relationship between Posts and Avg. Likes
1e6
Avg. Likes
8
6
4
2
0
0
2500
5000
7500
10000
12500
15000
17500
Posts


Relationship between Posts and Influence Score
90


Influence Score
80
70
60
50
40
0
2500
5000
7500
10000
12500
15000
17500
Posts