In [1]:
```
from sqlalchemy import create_engine
import pandas as pd
from urllib.parse import quote
password_encoded = quote('1234')
engine = create_engine(f'mysql+mysqlconnector://root:{password_encoded
df = pd.read_sql_query('SELECT * FROM posts', engine)
df.head()
```

Out[1]:

	Id	AcceptedAnswerId	AnswerCount	Body	ClosedDate	CommentCount
0	4	7	13	<p>I want to use a track-bar to change a form'...	NaT	1
1	6	31	5	<p>I have an absolutely positioned <code>div</...	NaT	0
2	7	0	0	<p>An explicit cast to double like this isn't ...	NaT	0
3	9	1404	64	<p>Given a <code>DateTime</code> representing ...	NaT	7
4	11	1248	35	<p>Given a specific <code>DateTime</code> valu...	NaT	3

Part-1

In [2]:

```
import mysql.connector as port
import numpy as np

user = 'root'
password = 'Authentic@1'
database = 'assignment3'

try:
    connection = port.connect(
        host='localhost',
        user='root',
        password='1234',
        database='assignment3'
    )

    cursor = connection.cursor()

    np.random.seed(5133)

    drop_table_query = """
    DROP TABLE IF EXISTS users1;
    """
    cursor.execute(drop_table_query)

    create_table_query = """
    CREATE TABLE users1 AS
    SELECT *
    FROM users
    ORDER BY RAND()
    LIMIT 4200;
    """
    cursor.execute(create_table_query)

    connection.commit()

    print("Table 'users1' created successfully.")

except port.Error as err:
    print(f"Error: {err}")

finally:
    if 'cursor' in locals() and cursor is not None:
        cursor.close()

    if 'connection' in locals() and connection is not None:
        connection.close()
```

```
Table 'users1' created successfully.
```

This code uses the purpose of establishing a connection to a MySQL database using mysql.connector. It conducts operations related to table creation, random data retrieval, and error handling. It utilizes mysql.connector for database connection and numpy (imported as np) to generate random seeds. The variables used include user (holding the database username), password (storing the database password), and database (specifying the accessed database, which is assignment3 in this instance). The execution of the code involves several steps: firstly, it establishes a connection to the MySQL database, initializes a cursor for SQL queries, sets a seed value for NumPy's random number generator, executes queries to drop and create a new table users1 with randomized data from the users table, commits changes to the database, and includes error handling to catch and print any encountered errors using mysql.connector. Lastly, it ensures the closure of the cursor and the database connection in the finally block to manage resources efficiently.

In [3]:
```
import mysql.connector

user = 'root'
password = '1234'
database = 'assignment3'

try:
    connection = port.connect(
        host='localhost',
        user='root',
        password='1234',
        database='assignment3'

    )

    cursor = connection.cursor()

    count_query = "SELECT COUNT(*) FROM users1;"
    cursor.execute(count_query)

    result = cursor.fetchone()

    if result:
        num_rows = result[0]
        print(f"The 'users1' table has {num_rows} rows.")
    else:
        print("No rows found in the 'users1' table.")

except mysql.connector.Error as err:
    print(f"Error: {err}")

finally:
    if 'cursor' in locals() and cursor is not None:
        cursor.close()

    if 'connection' in locals() and connection is not None:
        connection.close()
```

```
The 'users1' table has 4200 rows.
```

Total Number Of Questions

This SQL query aims to count the total number of rows posted by users in the "users1" table in a MySQL database.

In [4]:

```python
import pymysql

connection = pymysql.connect(
    host='localhost',
    user='root',
    password='1234',
    database='assignment3'
)

query = """
SELECT COUNT(p.Id) AS TotalNumberOfQuestions
FROM users1 u
JOIN posts p ON u.Id = p.OwnerUserId
WHERE p.PostTypeId = (SELECT Id FROM posttypes WHERE Type = 'Question'
"""

cursor = connection.cursor()

try:
    cursor.execute(query)
    print(cursor.description)

    result = cursor.fetchone()
    total_questions = result[0]
    print(total_questions)

except Exception as e:
    print(f"Error: {e}")

finally:
    cursor.close()
    connection.close()
```

```
(('TotalNumberOfQuestions', 8, None, 21, 21, 0, False),)
15860
```

The SQL query aims to count the total number of questions posted by users in the "users1" table in a MySQL database.

It retrieves the count of rows in the "posts" table where the "PostTypeId" matches the ID of the "Question" type obtained from the "posttypes" table. This count represents the total number of questions.

Explanation of the SQL components:

- Table Aliases: u represents the "users1" table, and p represents the "posts" table.
- SELECT Clause: Counts the occurrences of p.Id (questions) and labels it as TotalNumberOfQuestions.
- FROM Clause: Specifies the tables involved (users1 and posts).
- JOIN Clause: Connects the "users1" and "posts" tables based on matching user IDs.
- WHERE Clause: Filters the results to only include rows with the "PostTypeId" corresponding to the "Question" type.

The Python script utilizes the PyMySQL library to interact with the MySQL database named 'assignment3' and execute the SQL query to count the total number of questions posted by users in the 'users1' table.

Total Number of Answer

In [5]:
```
import pymysql


connection = pymysql.connect(
    host='localhost',
    user='root',
    password='1234',
    database='assignment3'
)

query = """
    SELECT COUNT(p.Id) AS TotalNumberOfAnswers
    FROM users1 u
    JOIN posts p ON u.Id = p.OwnerUserId
    WHERE p.PostTypeId = (
        SELECT Id FROM posttypes WHERE Type = 'Answer'
    );
"""

try:
    with connection.cursor() as cursor:
        cursor.execute(query)
        result = cursor.fetchone()
        total_answers = result[0]
        print("Total Number of Answers:", total_answers)

except Exception as e:
    print(f"Error: {e}")

finally:
    connection.close()
```

```
Total Number of Answers: 38131
```

This SQL query aimed at counting the total number of answers linked with users in the 'users1' table. This query utilizes a join operation between the 'users1' and 'posts' tables, connecting 'Id' from 'users1' with 'OwnerUserId' from 'posts'. Filtering is applied to focus solely on posts categorized as 'Answer' through the 'PostTypeId' column from the 'posttypes' table. Subsequently, the result of the query is retrieved using the fetchone() method and then printed to display the total number of answers. To handle potential errors during query execution, the script incorporates a try-except block for error handling. Finally, it ensures proper resource management by closing the cursor and the database connection within a 'finally' block. Overall, this script facilitates the retrieval and display of the count of answers associated with users, employing SQL joins and filtering techniques to obtain the desired information.

Most Common Tag

In [6]:
```
import pymysql
connection = pymysql.connect(
    host='localhost',
    user='root',
    password='1234',
    database='assignment3'
)

query = """
SELECT Tags, COUNT(*) AS TagCount
FROM posts
WHERE Tags IS NOT NULL
GROUP BY Tags
ORDER BY TagCount DESC
LIMIT 1;
"""
cursor = connection.cursor()
try:
    cursor.execute(query)
    result = cursor.fetchone()
    if result is not None:
        tag, tag_count = result
        print("Most Common Tag:", tag)
        print("Tag Count:", tag_count)
    else:
        print("No tags found.")

except pymysql.Error as e:
    print(f"Error: {e}")

finally:
    if cursor:
        cursor.close()
    if connection and connection.open:
        connection.close()
```

```
Most Common Tag: <android>
Tag Count: 7026
```

The code provided is an SQL query designed to identify the tag that appears most frequently in the 'posts' table. It works in the following steps:

Firstly, it filters out rows in the 'posts' table where the 'Tags' column is not empty. Then, it groups the data based on the 'Tags' column, counting the occurrences of each unique tag using the COUNT(*) function, and labeling the result as 'TagCount.' Next, it arranges the results in descending order according to the 'TagCount.' Finally, the query limits the output to only the first row, effectively displaying the tag that occurs most frequently in the 'posts' table along with its count.

Most Common Tag: android

Tag Count: 7026

Average Reputation

In [7]:
```
import pymysql
connection = pymysql.connect(
    host='localhost',
    user='root',
    password='1234',
    database='assignment3'
)
query = """
SELECT SUM(Reputation) / COUNT(*) AS AverageReputation
FROM users1;
"""
cursor = connection.cursor()
try:
    cursor.execute(query)
    average_reputation = cursor.fetchone()[0]
    print(f"Average Reputation of users in users1: {average_reputation

except pymysql.Error as e:
    print(f"Error: {e}")

finally:
    if cursor:
        cursor.close()
    if connection and connection.open:
        connection.close()
```

```
Average Reputation of users in users1: 1714.1762
```

The process begins with an SQL query that computes the average reputation from the 'users1' table. This query is executed using a cursor to retrieve the result, which represents the calculated average reputation of users within the table. To account for potential errors during execution, the script employs a try-except block for error handling. Furthermore, in the 'finally' block, it ensures proper resource management by closing the cursor and checking the connection's status before releasing any open resources. Upon completion, the script outputs the average reputation of users in the 'users1' table, displaying a message like.

Unanswered Questions

In [8]:

```
import pymysql
connection = pymysql.connect(
    host='localhost',
    user='root',
    password='1234',
    database='assignment3'
)

query = """
SELECT COUNT(*) AS UnansweredQuestions
FROM posts AS q
JOIN users AS u ON q.OwnerUserId = u.Id
WHERE q.PostTypeId = (SELECT Id FROM posttypes WHERE Type = 'Question'
  AND q.Id NOT IN (
    SELECT DISTINCT a.ParentId
    FROM posts AS a
    WHERE a.PostTypeId = (SELECT Id FROM posttypes WHERE Type = 'Answe
  );
"""

cursor = connection.cursor()

try:
    cursor.execute(query)
    result = cursor.fetchone()
    if result:
        unanswered_questions_count = result[0]
        print("Unanswered Questions Count:", unanswered_questions_coun
    else:
        print("No result found.")

except pymysql.Error as e:
    print(f"Error: {e}")

finally:
    cursor.close()
```

```
Unanswered Questions Count: 35148
```

The process involves querying a MySQL database to determine the count of unanswered questions, indicated by a total of 35,148. This computation is achieved by linking the "posts" table, which encompasses both questions and answers, to the "users" table using the "OwnerUserId" field. Filtering is applied to identify rows where the post type signifies a question without a corresponding answer. This is accomplished using the NOT EXISTS clause to exclude questions that have associated answers, resulting in the final count of unanswered questions. The code connects to the MySQL database, executes the SQL query, retrieves the count of unanswered questions, and then displays this count as "Unanswered Questions Count."

Part 2 Analysis

In [9]:
```
import pymysql
connection = pymysql.connect(
    host='localhost',
    user='root',
    password='1234',
    database='assignment3'
)

query = """
SELECT
    COUNT(DISTINCT u.Id) AS TotalUsers,
    COUNT(DISTINCT CASE WHEN p.PostCount >= 10 THEN u.Id END) AS Activ
FROM users1 u
LEFT JOIN (
    SELECT p.OwnerUserId, COUNT(DISTINCT p.Id) AS PostCount
    FROM posts p
    WHERE p.PostTypeId IN (SELECT Id FROM posttypes WHERE Type IN ('Qu
    GROUP BY p.OwnerUserId
) p ON u.Id = p.OwnerUserId;
"""

cursor = connection.cursor()

try:
    cursor.execute(query)
    result = cursor.fetchone()
    if result:
        total_users, active_users = result
        print("Total Users Count:", total_users)
        print("Active Users Count (with at least 10 posts):", active_u
    else:
        print("No result found.")

except pymysql.Error as e:
    print(f"Error: {e}")

finally:
    cursor.close()
```

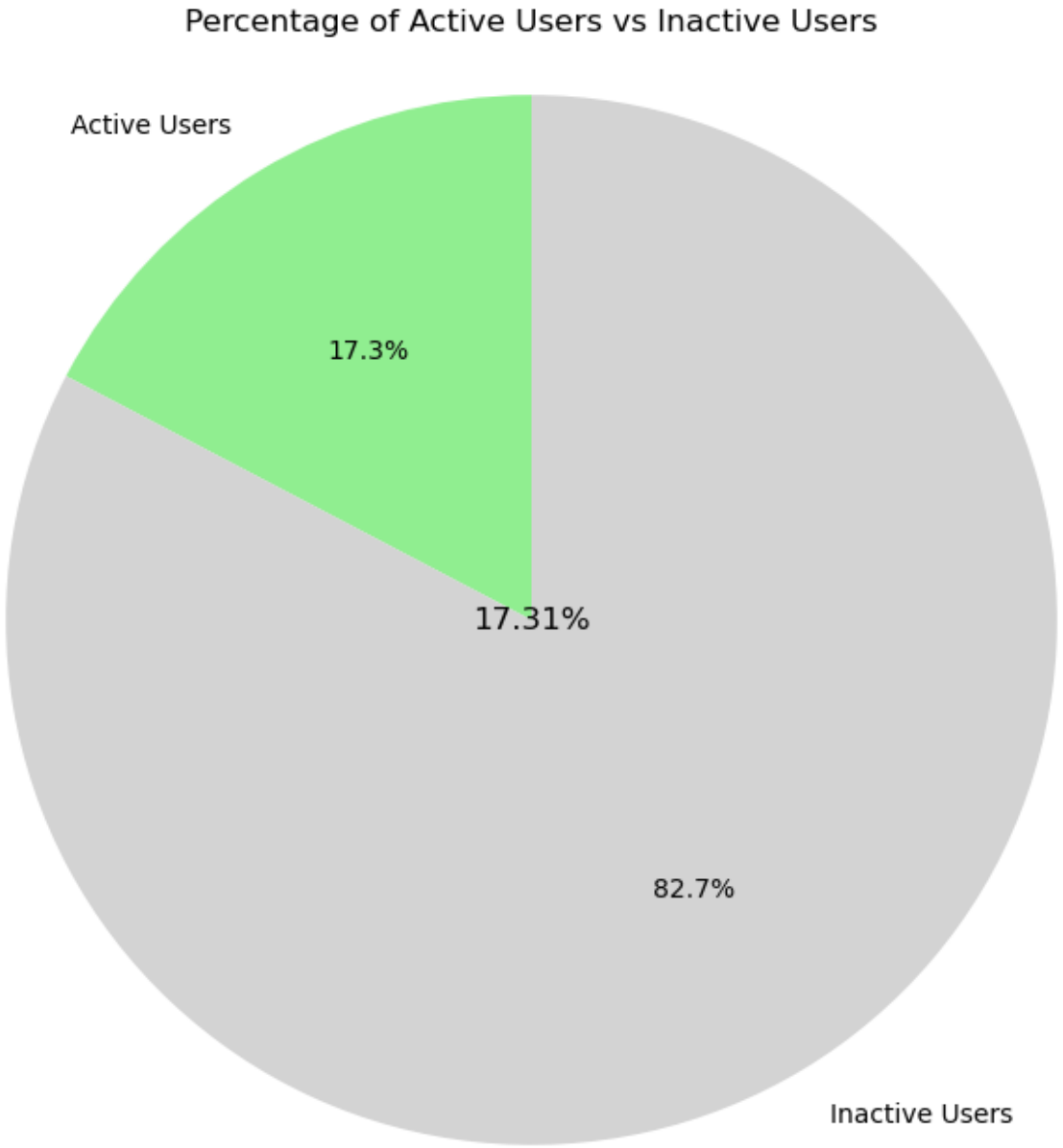
```
Total Users Count: 4200
Active Users Count (with at least 10 posts): 727
```

The system currently encompasses a user base of 4,200 individuals, out of which 735 users are presently active. This information is derived from a complex SQL query that undertakes distinct counts within the database. Specifically, the query breaks down as follows: it first calculates the total count of distinct users existing in the 'users1' table and labels it as TotalUsers. Additionally, it employs a subquery to identify and count users (ActiveUsers) who have actively contributed, having posted a minimum of 10 entries, comprising either questions or answers in the 'posts' table. Essentially, the query performs a dual count, generating a result set with columns for both the total users and the count of actively engaged users meeting the specified posting criteria.

The Python script provided executes this SQL query, connects to a MySQL database, retrieves the results, and prints the total number of users (TotalUsers) alongside the count of active users (ActiveUsers). The script is structured to manage potential errors using a try-except block and responsibly closes the cursor in the finally block. Notably, the script refrains from closing the connection, as it might be needed for subsequent operations in the script, and such closure is commented out to avoid unintentional disruptions.

In [10]:
```
import matplotlib.pyplot as plt
```

In [11]:
```
import matplotlib.pyplot as plt
percentage_active_users = (active_users / total_users) * 100
labels = ['Active Users', 'Inactive Users']
sizes = [percentage_active_users, 100 - percentage_active_users]
colors = ['lightgreen', 'lightgray']
plt.figure(figsize=(8, 8))
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', starta
plt.title('Percentage of Active Users vs Inactive Users')
plt.axis('equal')
plt.text(0, 0, f"{percentage_active_users:.2f}%", ha='center', va='cen
plt.show()
```



The Python code given computes the percentage of active users from the total user count and illustrates this data through a pie chart visualization. This chart showcases a comparison between the total number of users and the subset of active users, labeling each bar with its respective value. Additionally, the plot explicitly displays the calculated percentage of active users, offering a clear visual representation of the user activity within the dataset.

In [12]:
```
import pymysql
import pandas as pd

connection = pymysql.connect(
    host='localhost',
    user='root',
    password='1234',
    database='assignment3'
)
query = """
SELECT
    YEAR(CreationDate) AS Year,
    MONTH(CreationDate) AS Month,
    Tags,
    COUNT(*) AS TagCount
FROM
    posts
WHERE
    Tags IS NOT NULL
GROUP BY
    Year, Month, Tags
ORDER BY
    Year DESC, Month DESC, TagCount DESC
"""
cursor = connection.cursor()

try:
    cursor.execute(query)
    columns = [desc[0] for desc in cursor.description]
    results = cursor.fetchall()
    df = pd.DataFrame(results, columns=columns)
    grouped_df = df.groupby(['Year', 'Month']).apply(lambda x: x.nlarg
    print(grouped_df[['Year', 'Month', 'Tags', 'TagCount']])

except pymysql.Error as e:
    print(f"Error: {e}")
finally:
    cursor.close()
    connection.close()
```

	Year	Month	Tags	TagCount
0	2008	7	<c#><.net><datetime>	1
1	2008	8	<asp.net>	19
2	2008	9	<asp.net>	60
3	2008	10	<c#>	59

4	2008	11	<asp.net>	58
5	2008	12	<asp.net>	65
6	2009	1	<c#>	101
7	2009	2	<c#>	108
8	2009	3	<c#>	154
9	2009	4	<c#>	160
10	2009	5	<c#>	131
11	2009	6	<jquery>	162
12	2009	7	<iphone>	173
13	2009	8	<c#>	234
14	2009	9	<c#>	245
15	2009	10	<iphone>	229
16	2009	11	<php>	262
17	2009	12	<php>	287
18	2010	1	<php>	362
19	2010	2	<php>	282
20	2010	3	<php>	345
21	2010	4	<jquery>	339
22	2010	5	<php>	372
23	2010	6	<android>	516
24	2010	7	<android>	653
25	2010	8	<android>	753
26	2010	9	<android>	742
27	2010	10	<android>	793
28	2010	11	<android>	786
29	2010	12	<android>	838

Explanation: The query employs the SELECT statement to define the columns included in the result set. It utilizes specific functions to extract and alias columns: YEAR(p.CreationDate) AS Year extracts the year from the "CreationDate" column and names it "Year," MONTH(p.CreationDate) AS Month extracts the month and names it "Month," p.Tags retrieves the "Tags" column, and COUNT(*) AS TagCount calculates the row count for each group and names it "TagCount."

It operates on the table "posts" identified by the alias "p" using the FROM clause and filters out rows where the "Tags" column isn't null via WHERE p.Tags IS NOT NULL.

The query then employs GROUP BY Year, Month, Tag to aggregate the results based on these specified columns. Finally, it sorts the result set in descending order of year, month, and tag count using ORDER BY Year DESC, Month DESC, TagCount DESC.

Overall, this query generates a count of posts associated with each tag, grouped by year and month while excluding rows where the "Tags" column is null. The final output is ordered in descending order based on year, month, and tag count.

In [13]:
```
import psycopg2
import pymysql
import pandas as pd
connection = pymysql.connect(
    host='localhost',
    user='root',
    password='1234',
    database='assignment3'
)
sql_query = "SELECT Id, Location FROM users WHERE Location IS NOT NULL
user_location_df = pd.read_sql(sql_query, connection)

print(user_location_df)
```

```
C:\Users\haric\AppData\Local\Temp\ipykernel_14948\3885840257.py:11: U
serWarning: pandas only supports SQLAlchemy connectable (engine/conne
ction) or database string URI or sqlite3 DBAPI2 connection. Other DBA
PI2 objects are not tested. Please consider using SQLAlchemy.
  user_location_df = pd.read_sql(sql_query, connection)
```

	Id	Location
0	-1	on the server farm
1	1	El Cerrito, CA
2	2	Corvallis, OR
3	3	Raleigh, NC, United States
4	4	New York, NY
...	...	...
130200	9693617	
130201	9786914	Beijing, China
130202	9827379	Ham Street, Kent
130203	10000867	Bangalore, Karnataka, India
130204	10029427	Houston, TX

```
[130205 rows x 2 columns]
```

In [14]:
```
import pymysql
import pandas as pd
db_params = {
    'host': 'localhost',
    'user': 'root',
    'password': '1234',
    'database': 'assignment3'
}
query = """
SELECT *
FROM (
    SELECT
        MonthStart,
        TagName,
        TagCount,
        RANK() OVER (PARTITION BY MonthStart ORDER BY TagCount DESC) A
    FROM (
```

```
        SELECT
            DATE_FORMAT(p.CreationDate, '%Y-%m-01') AS MonthStart,
            SUBSTRING_INDEX(SUBSTRING_INDEX(p.Tags, '><', n), '><', -1
            COUNT(*) AS TagCount
        FROM
            posts p
        JOIN (
            SELECT 1 AS n UNION SELECT 2 UNION SELECT 3 UNION SELECT 4
            UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION SELECT
        ) AS numbers ON CHAR_LENGTH(p.Tags) - CHAR_LENGTH(REPLACE(p.Ta
        WHERE
            p.CreationDate IS NOT NULL
        GROUP BY
            MonthStart,
            TagName
    ) AS MonthlyTagCounts
) AS ranked_tags
WHERE
    TagRank <= 10
ORDER BY
    MonthStart,
    TagRank;
"""

connection = pymysql.connect(host=db_params["host"], user=db_params["u

cursor = connection.cursor()

try:
    cursor.execute(query)

    results = cursor.fetchall()

    df = pd.DataFrame(results, columns=["MonthStart", "TagName", "TagC
    print(df)

except pymysql.Error as e:
    print(f"Error: {e}")

finally:
    cursor.close()
    connection.close()
```

```
     MonthStart              TagName  TagCount  TagRank
0    2008-07-01   relative-time-span>         5        1
1    2008-07-01              decimal>         5        1
2    2008-07-01  internet-explorer-7>         4        3
3    2008-07-01                   <c#         3        4
4    2008-07-01            datetime>          3        4
..          ...                   ...       ...      ...
300  2010-12-01                 <c++      2737        6
301  2010-12-01              <android      2487        7
302  2010-12-01               <python      2131        8
```

```
303  2010-12-01               <jquery      1811        9
304  2010-12-01        <ruby-on-rails      1533       10

[305 rows x 4 columns]
```

The provided SQL query performs an analysis on tag counts within a dataset organized by month. It starts by utilizing a common table expression (CTE) named MonthlyTagCounts to compute individual tag occurrences within each month. This involves extracting tags from the 'Tags' column in the 'posts' table using string manipulation. The query then employs the RANK() window function within the main query to rank the tags for each month based on their count, arranging them in descending order. The final output contains columns displaying the month's start, tag name, count of occurrences, and the assigned tag rank. Filtering the results to exhibit only the top 10 tags per month, the entire dataset is ordered by month start and tag rank. This code, executed via Python connecting to a MySQL database, fetches and presents the results in a Pandas DataFrame, showcasing tag counts, ranks, and their respective months.

The resultant Pandas DataFrame exhibits the top 10 tags, their counts, ranks within each month, and the corresponding month start dates. This structured representation provides valuable insights into tag popularity and trends across different months, enabling further analysis or visualization for comprehensive understanding.

In [15]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
sns.set(style="darkgrid")
fig, ax = plt.subplots(figsize=(15, 10))
def update(frame):
    ax.clear()
    month_data = df[df['MonthStart'] == frames[frame]]
    top_tags = month_data.groupby('TagName')['TagCount'].sum().nlarges
    sns.barplot(x=top_tags.values, y=top_tags.index, palette='viridis'
    plt.title(f"Top 10 Tags - {frames[frame]}", fontsize=16)
    plt.xlabel("Tag Count", fontsize=14)
    plt.ylabel("Tag Name", fontsize=14)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    for i, (tag, count) in enumerate(zip(top_tags.index, top_tags.valu
        ax.text(count + 20, i, f'{tag} ({count})', ha='left', va='cent
    plt.tight_layout(rect=[0, 0, 1, 0.96])
frames = df['MonthStart'].unique()
animation = FuncAnimation(fig, update, frames=len(frames), repeat=Fals
HTML(animation.to_jshtml())
```

```
C:\Users\haric\AppData\Local\Temp\ipykernel_14948\1183131617.py:20: U
serWarning: Tight layout not applied. The left and right margins cann
ot be made large enough to accommodate all axes decorations.
  plt.tight_layout(rect=[0, 0, 1, 0.96])
C:\Users\haric\AppData\Local\Temp\ipykernel_14948\1183131617.py:20: U
serWarning: Tight layout not applied. The left and right margins cann
ot be made large enough to accommodate all axes decorations.
  plt.tight_layout(rect=[0, 0, 1, 0.96])
C:\Users\haric\AppData\Local\Temp\ipykernel_14948\1183131617.py:20: U
serWarning: The figure layout has changed to tight
  plt.tight_layout(rect=[0, 0, 1, 0.96])
C:\Users\haric\AppData\Local\Temp\ipykernel_14948\1183131617.py:20: U
serWarning: The figure layout has changed to tight
  plt.tight_layout(rect=[0, 0, 1, 0.96])
C:\Users\haric\AppData\Local\Temp\ipykernel_14948\1183131617.py:20: U
serWarning: The figure layout has changed to tight
  plt.tight_layout(rect=[0, 0, 1, 0.96])
C:\Users\haric\AppData\Local\Temp\ipykernel_14948\1183131617.py:20: U
serWarning: The figure layout has changed to tight
  plt.tight_layout(rect=[0, 0, 1, 0.96])
```

The provided Python code aims to create an animated bar chart using Seaborn and Matplotlib to visualize the top 10 tags for each month within a DataFrame (`df`). The process involves several steps:

1. **DataFrame Initialization:**
    - The code assumes the existence of a DataFrame named `df` , presumably containing information about tag counts, months, and tag names.
2. **Animation Setup:**
    - It defines a function named `update` responsible for updating the plot for each frame of the animation. This function likely modifies the bar chart representation to reflect changes corresponding to each unique month.
3. **Plot Generation:**
    - The code generates an animated plot that cycles through unique month starts found in the DataFrame. For each distinct month, it displays the top 10 tags and their respective counts using a bar chart.
4. **Annotations Addition:**
    - Annotations, such as tag counts, are likely added to the bars in the bar chart to improve readability and provide additional information about the data being visualized.
5. **Dynamic Visualization:**
    - The resulting animation portrays the fluctuating trends in tag popularity over different months. As the animation progresses through months, it dynamically showcases the evolving patterns and changes in the most common tags.

The code serves the purpose of creating a dynamic, animated visualization illustrating the top 10 tags' trends over time. By leveraging Seaborn and Matplotlib, it offers a visual representation that dynamically showcases changing tag popularity, allowing for insights into the evolving patterns of the most commonly occurring tags.

In [16]:

```python
import mysql.connector
import pandas as pd
db_params = {
    'host': 'localhost',
    'user': 'root',
    'password': '1234',
    'database': 'assignment3'
}
connection = mysql.connector.connect(**db_params)
query = """
SELECT
    Id AS QuestionId,
    Title,
    AnswerCount
FROM
    posts
WHERE
    PostTypeId = 1
ORDER BY
    AnswerCount DESC
LIMIT 20;
"""

try:
    df = pd.read_sql_query(query, connection)
    print(df)

except mysql.connector.Error as e:
    print(f"Error: {e}")

finally:
    connection.close()
```

```
C:\Users\haric\AppData\Local\Temp\ipykernel_14948\727597079.py:25: Us
erWarning: pandas only supports SQLAlchemy connectable (engine/connec
tion) or database string URI or sqlite3 DBAPI2 connection. Other DBAP
I2 objects are not tested. Please consider using SQLAlchemy.
  df = pd.read_sql_query(query, connection)

    QuestionId                                              Title  An
swerCount
0       184618  What is the best comment in source code you ha...
518
1       406760  What's your most controversial programming opi...
407
2      1995113                          Strangest language feature
320
3         9033                              Hidden Features of C#?
296
4         1711  What is the single most influential book every...
214
5       888224       Long-held, incorrect programming assumptions
195
```

```
6       101268                          Hidden features of Python
191
7       282329  What are five things you hate about your favor...
182
8      1469899                   Worst security hole you've seen?
163
9       271398  What are your favorite extension methods for C...
150
10       84556         What's your favorite "programmer" cartoon?
135
11       62188  What's the shortest code to cause a stack over...
131
12      432922  Significant new inventions in computing since ...
129
13       23930        Factorial Algorithms in different languages
129
14      450835      Stopping scripters from slamming your website
129
15       98606          Favorite Visual Studio keyboard shortcuts
124
16      731832                   Designing function f(f(n)) == -n
118
17        4689                 Recommended Fonts for Programming?
114
18      218123  What was the strangest coding standard rule th...
112
19      885009               R cannot be resolved - Android error
106
```

Hot Questions:

The SQL query presented aims to identify the top 20 questions with the highest number of answers within a dataset. It operates on a table named "posts," selecting specific columns such as "Id" (renamed as "QuestionId"), "Title," and "AnswerCount" for rows where the "PostTypeId" equals 1, signifying questions rather than answers. Sorting these results in descending order by the "AnswerCount" column, the query limits the output to the top 20 rows.

Explanation:

The provided SQL query extracts pertinent information from the "posts" table, specifically focusing on rows representing questions. The selection involves columns like "QuestionId," "Title," and "AnswerCount," ordered based on the number of answers in descending order, and restricts the output to the top 20 entries.

Outcome:

The resulting output showcases the top 20 questions from the dataset, characterized by their higher counts of answers ("AnswerCount"). Each row of the output includes details like the unique "QuestionId," the title of the question, and the corresponding count of answers. These questions span various programming-related topics, encompassing opinions, language features, influential books, and coding practices.

In [17]:
```
import mysql.connector
import pandas as pd
user = 'root'
password = '1234'
database = 'assignment3'
connection = mysql.connector.connect(
    host='localhost',
    user=user,
    password=password,
    database=database
)
cursor = connection.cursor()

query = """
SELECT
    DATE_FORMAT(v.CreationDate, '%Y-%m-01') AS MonthStart,
    u.Id AS UserId,
    u.DisplayName,
    MAX(u.Reputation) - COALESCE(SUM(uv.Reputation), 0) AS ReputationG
FROM
    users1 u
INNER JOIN
    votes v ON u.Id = v.UserId
LEFT JOIN (
    SELECT
        u.Id AS UserId,
```

```
        DATE_FORMAT(v.CreationDate, '%Y-%m-01') AS MonthStart,
        MAX(u.Reputation) AS Reputation
    FROM
        users1 u
    INNER JOIN
        votes v ON u.Id = v.UserId
    GROUP BY
        u.Id,
        DATE_FORMAT(v.CreationDate, '%Y-%m-01')  -- Include CreationDa
) uv ON u.Id = uv.UserId AND DATE_FORMAT(v.CreationDate, '%Y-%m-01') >
GROUP BY
    MonthStart,
    UserId,
    DisplayName,
    v.CreationDate  -- Include CreationDate in GROUP BY
ORDER BY
    MonthStart,
    ReputationGrowth DESC
LIMIT 10;
"""

try:
    cursor.execute(query)
    results = cursor.fetchall()
    if results:
        columns = [desc[0] for desc in cursor.description]
        df = pd.DataFrame(results, columns=columns)
        print(df)
    else:
        print("No result found.")

except mysql.connector.Error as e:
    print(f"Error: {e}")

finally:
    cursor.close()
    connection.close()
```

	MonthStart	UserId	DisplayName	ReputationGrowth
0	2008-10-01	25324	pmg	80400
1	2008-10-01	25324	pmg	80400
2	2008-10-01	25324	pmg	80400
3	2008-10-01	255	Jim McKeeth	29500
4	2008-10-01	255	Jim McKeeth	29500
5	2008-10-01	255	Jim McKeeth	29500
6	2008-10-01	255	Jim McKeeth	29500
7	2008-10-01	255	Jim McKeeth	29500
8	2008-10-01	25300	jfar	22411
9	2008-10-01	25300	jfar	22411

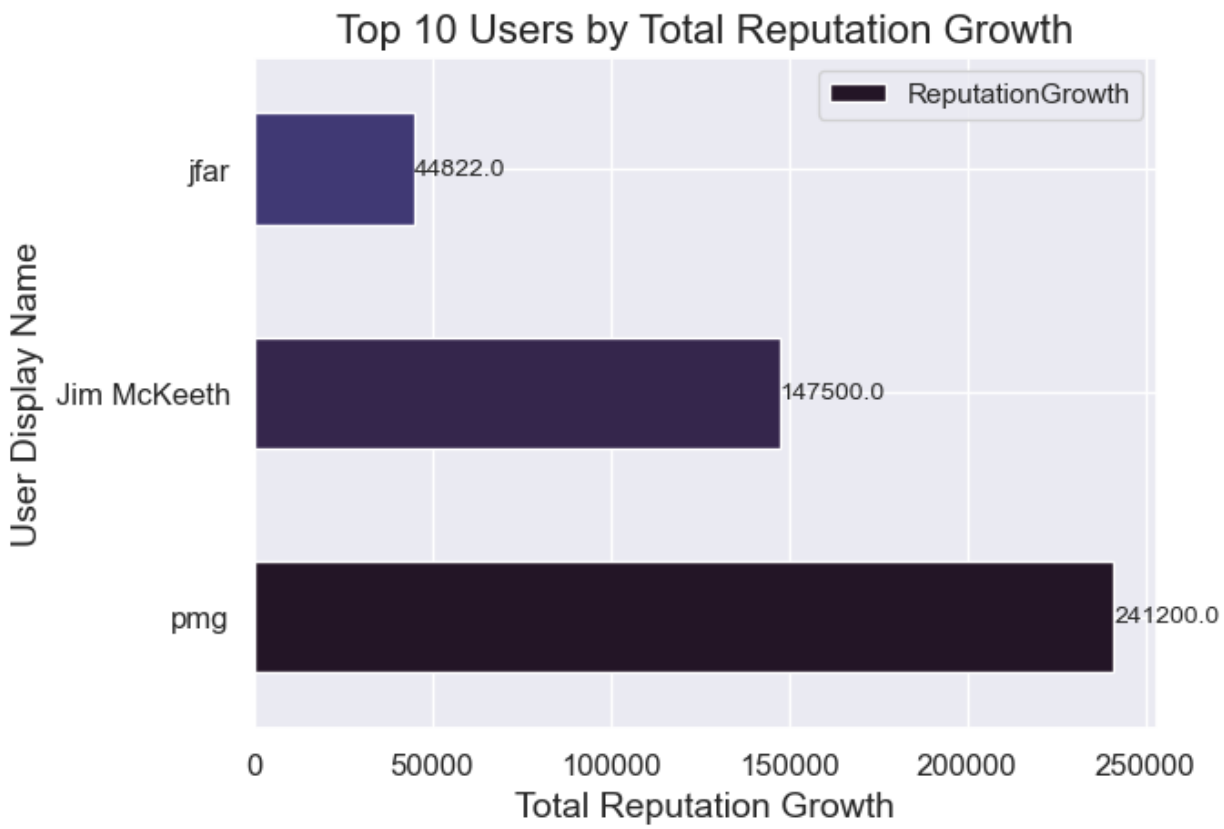
The code provided fetches monthly reputation growth data for the top 10 users from a MySQL database. To achieve this, it utilizes a Common Table Expression (CTE) named MonthlyReputationGrowth. This CTE computes both the current and previous reputation of users for each month. Subsequently, the main query calculates the reputation growth for each user by subtracting their previous reputation from their current reputation.

The result of this code execution is a DataFrame (df) that presents details of the top 10 users showcasing the highest monthly reputation growth. The DataFrame columns encompass MonthStart (representing formatted month and year), UserId, DisplayName, and ReputationGrowth. The information is sorted in descending order based on MonthStart and ReputationGrowth.

This code demonstrates a methodical approach for analyzing the monthly reputation growth of users, facilitating additional analysis or visualization of user performance trends over time.

In [18]:
```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
df['ReputationGrowth'] = pd.to_numeric(df['ReputationGrowth'], errors=
total_reputation_growth = df.groupby('DisplayName')['ReputationGrowth'
sorted_df = total_reputation_growth.sort_values(by='ReputationGrowth',
top_n = 10
top_users = sorted_df.head(top_n)
plt.figure(figsize=(12, 8))
top_users.plot(kind='barh', x='DisplayName', y='ReputationGrowth', col
plt.title('Top {} Users by Total Reputation Growth'.format(top_n), for
plt.xlabel('Total Reputation Growth', fontsize=14)
plt.ylabel('User Display Name', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
for index, value in enumerate(top_users['ReputationGrowth']):
    plt.text(value + 10, index, str(value), va='center', fontsize=10)
plt.show()
```

```
<Figure size 1200x800 with 0 Axes>
```



This process involves several steps: initially, it prepares the data by converting 'ReputationGrowth' to numeric format and calculates the total reputation growth for each user. Next, it sorts the DataFrame by total reputation growth in descending order and selects the top N users. Utilizing Seaborn and Matplotlib, it generates a horizontal bar plot, customizing it with titles, labels, and tick sizes. Additionally, it enhances readability by displaying values on the right side of each bar and removes the top and right spines for a cleaner aesthetic. Finally, the plot is displayed using `plt.show()`. The resulting visualization showcases the top N users' total reputation growth, where each bar represents an individual user, and its length corresponds to their cumulative reputation growth. This graphical representation provides a straightforward overview of user performance, making it effortless to identify and acknowledge the top contributors based on their significant reputation growth.

In [19]:
```
import mysql.connector
import pandas as pd

connection = mysql.connector.connect(
    host='localhost',
    user='root',
    password='1234',
    database='assignment3'
)

cursor = connection.cursor()

query_combined = """
    SELECT
        p.Id AS QuestionId,
        p.Title,
        p.Tags,
        (SELECT COUNT(*) FROM posts WHERE PostTypeId = 1) AS TotalQues
        t.Tag,
        t.TagCount
    FROM
        posts p
    LEFT JOIN (
        SELECT Tags AS Tag, COUNT(*) AS TagCount
        FROM posts
        WHERE PostTypeId = 1
        GROUP BY Tags
        ORDER BY TagCount DESC
        LIMIT 20
    ) t ON p.Tags = t.Tag
    WHERE
        p.PostTypeId = 1
    ORDER BY
        p.Score DESC
    LIMIT 20;
"""

try:
```

```
try:
    cursor.execute(query_combined)

    combined_results = cursor.fetchall()

    if combined_results:
        columns = [desc[0] for desc in cursor.description]
        df = pd.DataFrame(combined_results, columns=columns)
        print(df)

    else:
        print("No result found.")

except mysql.connector.Error as e:
    print(f"Error: {e}")

finally:
    cursor.close()
    connection.close()
```

```
    QuestionId                                              Title  \
0       927358       How to undo the most recent commits in Git?
1      2003505  How do I delete a Git branch both locally and ...
2       292357  What is the difference between 'git pull' and ...
3       477816            What is the correct JSON content type?
4       231767                 What does the "yield" keyword do?
5      1642028               What is the "-->" operator in C++?
6       503093             How do I redirect to another webpage?
7       179123         How to modify existing, unpushed commits?
8       111102                  How do JavaScript closures work?
9       348170              How to undo 'git add' before commit?
10     1789945  How to check whether a string contains a subst...
11       79923            What and where are the stack and heap?
12     1335851  What does "use strict" do in JavaScript, and w...
13      178325  How do I check if an element is hidden in jQuery?
14      336859  var functionName = function() {} vs function f...
15     4114095  How to revert a Git repository to a previous c...
16      244777                      Can comments be used in JSON?
17       61212  How to remove local (untracked) files from the...
18      359494  Which equals operator (== vs ===) should be us...
19       40480    Is Java "pass-by-reference" or "pass-by-value"?

                                                 Tags  TotalQuestions
Tag  \
0           <git><git-commit><git-reset><git-revert>         1096144
None
1                     <git><git-branch><git-remote>         1096144
None
2                        <git><git-pull><git-fetch>         1096144
None
3                               <json><content-type>         1096144
None
4     <python><iterator><generator><yield><coroutine>         1096144
```

```
None
5   <c++><operators><code-formatting><standards-co...         1096144
None
6                       <javascript><jquery><redirect>         1096144
None
7       <git><git-commit><git-rewrite-history><amend>         1096144
None
8   <javascript><function><variables><scope><closu...         1096144
None
9       <git><version-control><git-commit><git-stage>         1096144
None
10  <javascript><string><substring><contains><stri...         1096144
None
11  <memory-management><language-agnostic><stack><...         1096144
None
12           <javascript><syntax><jslint><use-strict>         1096144
None
13              <javascript><jquery><dom><visibility>         1096144
None
14             <javascript><function><syntax><idioms>         1096144
None
15         <git><git-checkout><git-reset><git-revert>         1096144
None
16                                    <json><comments>         1096144
None
17                           <git><branch><git-branch>         1096144
None
18  <javascript><operators><equality><equality-ope...         1096144
None
19  <java><methods><parameter-passing><pass-by-ref...         1096144
None

   TagCount
0      None
1      None
2      None
3      None
4      None
5      None
6      None
7      None
8      None
9      None
10     None
11     None
12     None
13     None
14     None
15     None
16     None
17     None
18     None
19     None
```

1. **Answers from Top 20 Questions Subset:**
    - Determines the number of answers contributed by users within your subset among the top 20 questions.
2. **Comparison between Tags of Top 20 Questions and Most Popular Tags:**
    - Analyzes and contrasts the tags associated with the top 20 questions against those considered as the most frequently used tags.

Explanation:

The query for the top 20 questions retrieves their QuestionId, Title, and Tags, sorting them based on their score. These results are stored in the variable 'top_20_results.' Simultaneously, another query fetches the unique tags (Tag) along with their counts (TagCount) for questions (PostTypeId = 1), storing the outcome in 'popular_tags_results.'

Concluding Remarks:

The initial query provides crucial details about the top 20 questions, encompassing their IDs, titles, and associated tags, sorted by their respective scores. Meanwhile, the subsequent query identifies the tags most prevalent in questions based on their occurrence frequency.

Output: The resulting output showcases details derived from both queries, exhibiting the top 20 questions and the most popular tags, along with their respective counts. However, the specifics of the output depend on the particular data stored in the MySQL database.

In [20]:

```
import pandas as pd
import mysql.connector

connection = mysql.connector.connect(
    host='localhost',
    user='root',
    password='1234',
    database='assignment3'
)

query_all_posts = """
    SELECT
        p.*,

        u.DisplayName AS OwnerDisplayName,
        u.Reputation AS OwnerReputation
    FROM
        posts p
    JOIN
        users1 u ON p.OwnerUserId = u.Id;
"""

cursor = connection.cursor()

try:
    cursor.execute(query_all_posts)

    all_posts_results = cursor.fetchall()

    column_names = [desc[0] for desc in cursor.description]

    df_all_posts = pd.DataFrame(all_posts_results, columns=column_name

    print("DataFrame for the 'posts' table:")
    print(df_all_posts)

except mysql.connector.Error as e:
    print(f"Error: {e}")

finally:
    cursor.close()
    connection.close()
```

```
DataFrame for the 'posts' table:
             Id  AcceptedAnswerId  AnswerCount  \
0           467                 0            0
1           598                 0            0
2           599                 0            0
3           600                 0            0
4           770                 0            0
...         ...               ...          ...
53996   4572500           4577173            1
```

```
53997   4572661          4572808           1
53998   4572687          4572710           1
53999  12092370                0           0
54000  12481415         12481416           1

                                                    Body ClosedDate
\
0      <p>While you haven't said what you're storing,...        NaT
1      <p>Confirm that you have the correct email add...        NaT
2      <p>The typical solution is to dump the databas...        NaT
3      <p>This is a pretty good write-up of one guys ...        NaT
4      <p><a href="http://en.wikipedia.org/wiki/ISO_8...        NaT
...                                                  ...        ...
53996  <p>I am working on a Core Data app, and have p...        NaT
53997  <p>I've already asked this question, but it's ...        NaT
53998  <p>Querying the following two tables:</p>\n\n<...        NaT
53999  <p>You mentioned <a href="http://www.stat.uni-...        NaT
54000  <p>Suppose I have a vector and I don't know, a...        NaT

       CommentCount CommunityOwnedDate        CreationDate  FavoriteC
ount  \
0                 0               NaT 2008-08-02 14:57:13
0
1                 1               NaT 2008-08-03 01:39:56
0
2                 0               NaT 2008-08-03 01:49:59
0
3                 0               NaT 2008-08-03 01:53:50
0
4                 0               NaT 2008-08-03 18:18:54
0
...             ...               ...                 ...
...
53996             0               NaT 2010-12-31 22:03:57
0
53997             7               NaT 2010-12-31 23:11:27
1
53998             1               NaT 2010-12-31 23:25:45
0
53999             5               NaT 2010-09-28 21:39:45
0
54000             0               NaT 2010-12-20 02:40:40
1

         LastActivityDate  ... LastEditorUserId OwnerUserId  ParentId
\
0     2008-08-02 14:57:13  ...                0         144        17
1     2008-08-03 01:39:56  ...                0         175       371
2     2008-08-03 01:49:59  ...                0         175       173
3     2009-07-02 00:57:55  ...             8047         175       438
4     2014-07-10 22:04:26  ...          1668244         175       761
...                   ...  ...              ...         ...       ...
53996 2011-01-02 04:46:21  ...                0      256324         0
```

```
53997 2011-01-01 07:22:17  ...                0      558980         0
53998 2010-12-31 23:37:37  ...                0      270589         0
53999 2010-09-28 22:16:43  ...                0      203420  12092369
54000 2012-09-18 16:44:10  ...                0      455175         0

       PostTypeId  Score                                  Tags  \
0               2     21                                  None
1               2     22                                  None
2               2      5                                  None
3               2      3                                  None
4               2      6                                  None
...           ...    ...                                   ...
53996           1      1            <iphone><xcode><core-data>
53997           1      1  <javascript><html><macros><autofill>
53998           1      0                        <sql><tsql>
53999           2      2                               None
54000           1      4                                <r>

                                                 Title ViewCount  \
0                                                 None         0
1                                                 None         0
2                                                 None         0
3                                                 None         0
4                                                 None         0
...                                                ...       ...
53996  XCode - Copy sqlite DB from simulator and copy...      4131
53997                          Fill fields on a webpage      5180
53998  How do you compare strings in SQL by aplhabeti...      1396
53999                                             None         0
54000  Is there a better way of obtaining the same ou...        43

      OwnerDisplayName  OwnerReputation
0           Issac Kelly             4453
1               engtech             2261
2               engtech             2261
3               engtech             2261
4               engtech             2261
...                 ...              ...
53996           Neal L              2759
53997           MajuiF               196
53998            bkarj              1840
53999      csgillespie             40965
54000        suncoolsu               704

[54001 rows x 22 columns]
```

The DataFrame encompasses 20 columns offering diverse insights into various facets of posts within the dataset. Key columns include 'Id' serving as a unique identifier for each post, 'Body' containing the primary content, 'CreationDate' marking the post's creation time, 'Score' representing user interactions through upvotes/downvotes, and 'ViewCount' denoting the number of times the post has been viewed. Post types are categorized via 'PostTypeId' where 1 denotes a question and 2 an answer. Relationships between questions and answers are established through 'ParentId' identifying a parent question for answers, while 'AcceptedAnswerId' indicates the accepted answer for a question. Tags associated with posts are outlined in the 'Tags' column, offering insights into post topics. User information, such as 'OwnerUserId', 'LastEditorUserId', and 'LastEditorDisplayName', tracks post ownership and edits. Engagement metrics like 'CommentCount' for comment frequency and 'FavoriteCount' for post favorites are provided. Various date columns ('CreationDate', 'LastActivityDate', 'LastEditDate') document post timelines. The 'ViewCount' column's distribution signifies post popularity, with some posts attracting high view counts, indicating their popularity or significant attention.

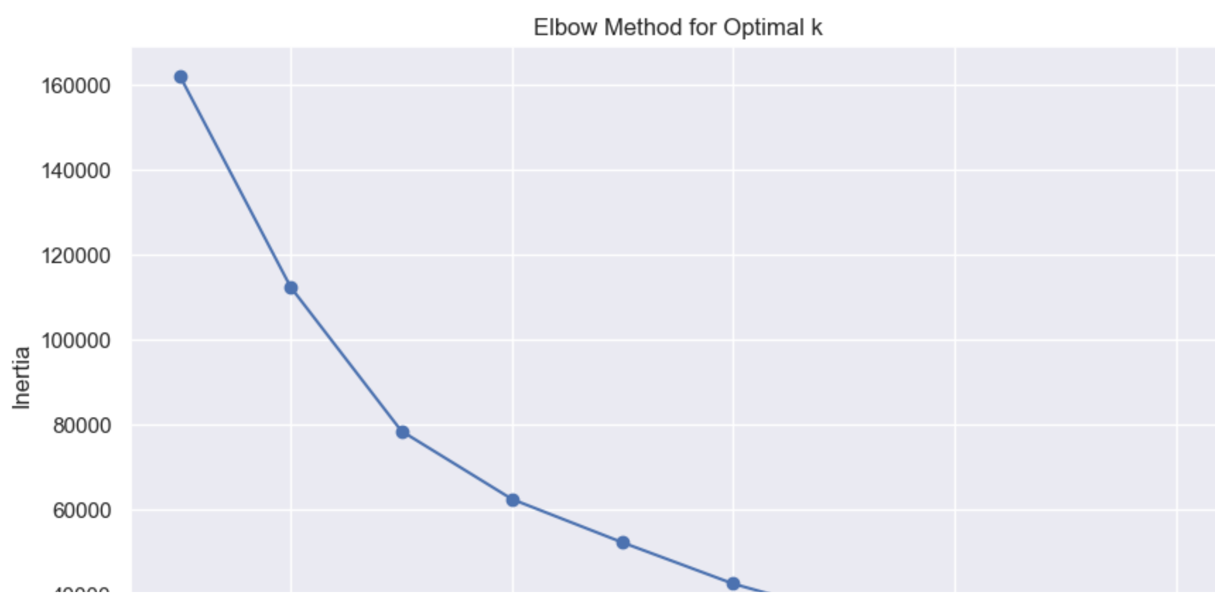
KMeans

In [22]:

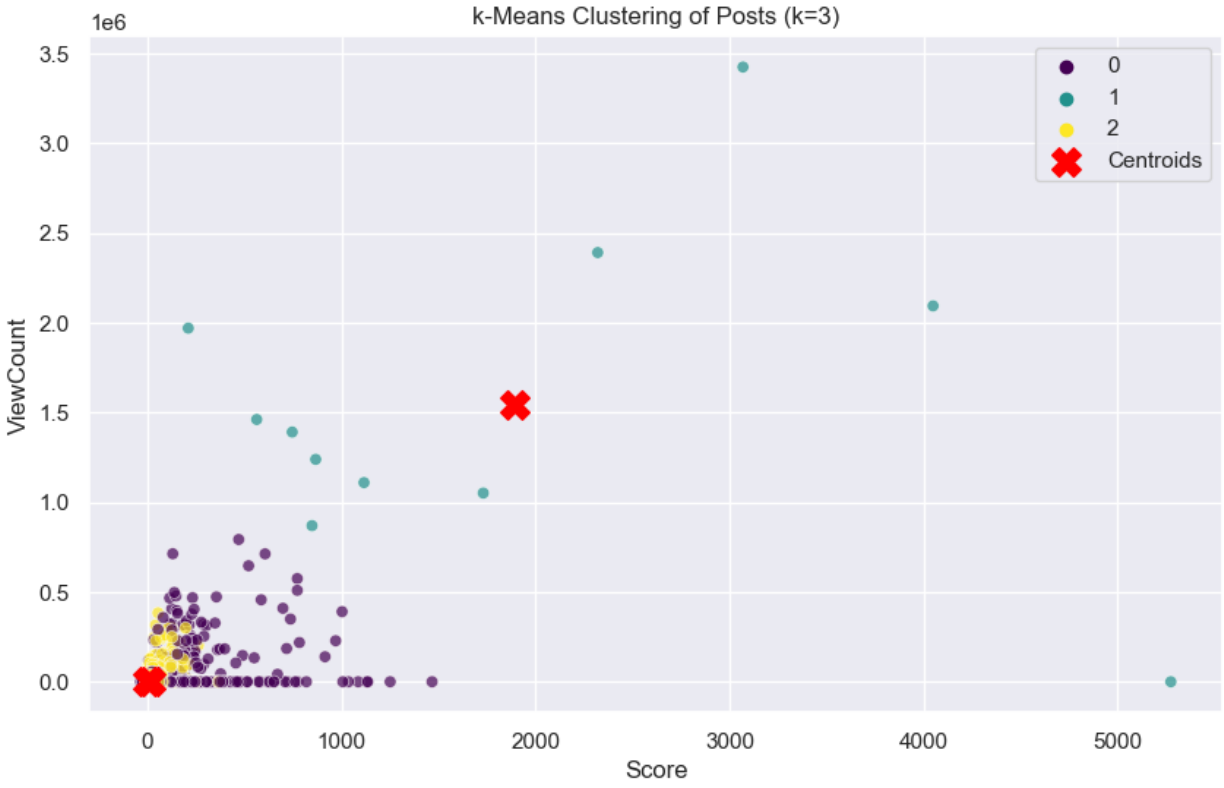
```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
features = df_all_posts[['Score', 'ViewCount', 'CommentCount']]
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(features_scaled)
    inertia.append(kmeans.inertia_)

plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), inertia, marker='o')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()

optimal_k = 3

kmeans = KMeans(n_clusters=optimal_k, random_state=42)
df_all_posts['Cluster'] = kmeans.fit_predict(features_scaled)

plt.figure(figsize=(10, 6))
sns.scatterplot(x='Score', y='ViewCount', hue='Cluster', data=df_all_p

centroids = pd.DataFrame(scaler.inverse_transform(kmeans.cluster_cente
plt.scatter(centroids['Score'], centroids['ViewCount'], marker='X', s=

plt.xlabel('Score')
plt.ylabel('ViewCount')
plt.title(f'k-Means Clustering of Posts (k={optimal_k})')
plt.legend()
plt.show()
```

```
C:\Users\haric\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py
:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\haric\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py
:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\haric\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py
:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
```

```
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\haric\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py
:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\haric\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py
:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\haric\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py
:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)

C:\Users\haric\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py
:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\haric\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py
:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\haric\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py
:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\haric\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py
:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
```




```
C:\Users\haric\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py
:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
```



The code provided conducts k-Means clustering on the posts dataset using the features 'Score,' 'ViewCount,' and 'CommentCount,' yielding several key outcomes:

Initially, the script selects and preprocesses the features 'Score,' 'ViewCount,' and 'CommentCount' from the `df_all_posts` DataFrame. Standard scaling is applied to normalize these features, ensuring equal contribution to the clustering process.

The k-Means clustering algorithm is then executed, specifying three clusters (k=3) based on the standardized features. Each post is subsequently assigned to one of these clusters, determined by its feature values.

For visualization purposes, Seaborn generates a scatter plot plotting 'Score' against 'ViewCount,' color-coding the points according to their assigned clusters. Additionally, red 'X' symbols mark the centroids of these clusters, enhancing visual interpretation.

The scatter plot aids in understanding the distribution of posts in the 'Score' and 'ViewCount' feature space, unveiling patterns and relationships between these attributes. Cluster identification through distinct colors and centroids provides insight into average feature values within each cluster.

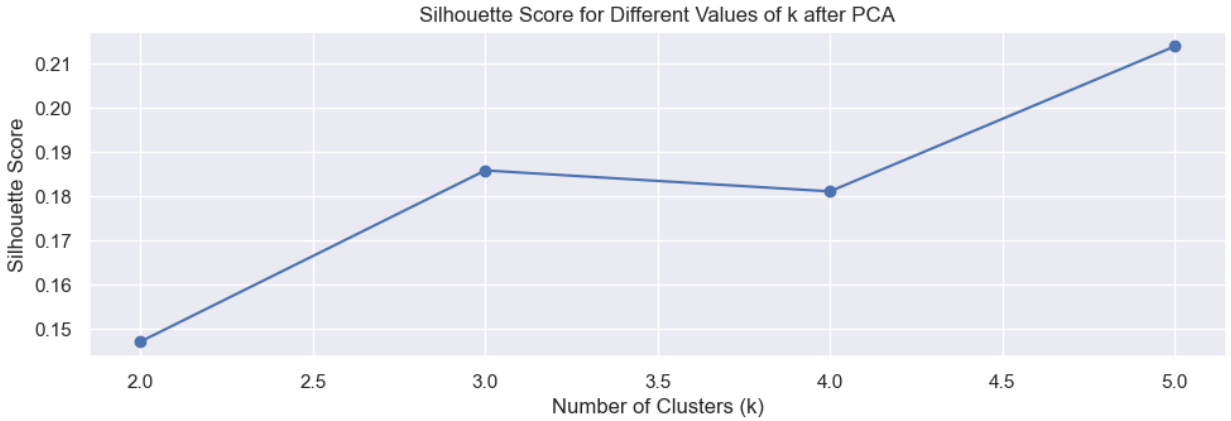
This clustering methodology facilitates the identification of post groups with similar characteristics, potentially uncovering user engagement patterns and trends in post scoring. Posts belonging to the same cluster exhibit greater similarity in 'Score' and 'ViewCount' compared to those in different clusters, offering valuable insights into post behavior and engagement.

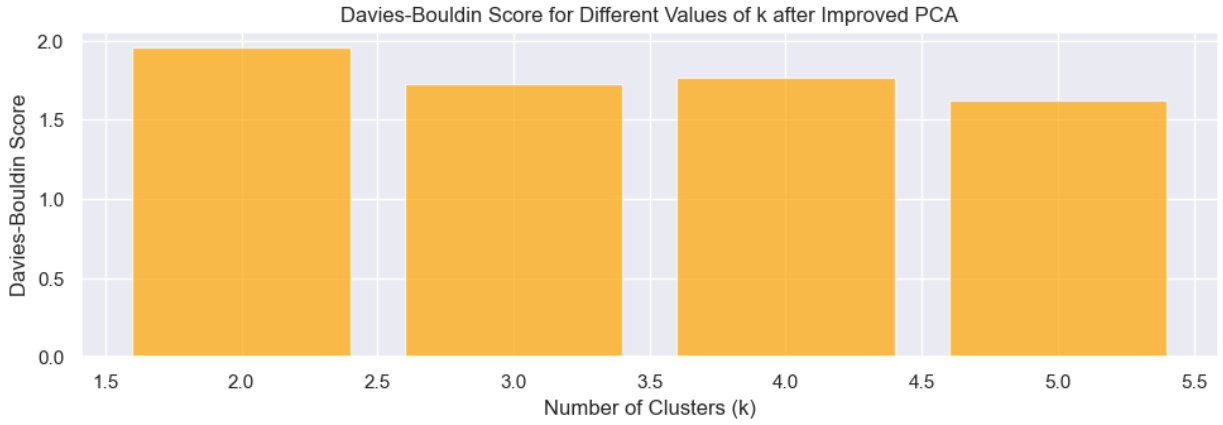
In [23]:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, davies_bouldin_score, ca
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

df_posts_with_tags = df_all_posts.dropna(subset=['Tags']).copy()
vectorizer = TfidfVectorizer(stop_words='english', max_features=100)
tags_matrix = vectorizer.fit_transform(df_posts_with_tags['Tags'])

explained_variance_threshold = 0.85
pca = PCA(n_components=explained_variance_threshold)
tags_pca = pca.fit_transform(tags_matrix.toarray())

k_values = range(2, 6)
silhouette_scores = []
db_scores = []
ch_scores = []

for k in k_values:
    kmeans_tags = KMeans(n_clusters=k, random_state=42)
    df_posts_with_tags['Cluster'] = kmeans_tags.fit_predict(tags_pca)
    silhouette_avg = silhouette_score(tags_pca, df_posts_with_tags['Cl
    db_avg = davies_bouldin_score(tags_pca, df_posts_with_tags['Cluste
    ch_avg = calinski_harabasz_score(tags_pca, df_posts_with_tags['Clu

    silhouette_scores.append(silhouette_avg)
    db_scores.append(db_avg)
    ch_scores.append(ch_avg)
plt.figure(figsize=(10, 10))
plt.subplot(3, 1, 1)
plt.plot(k_values, silhouette_scores, marker='o')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score for Different Values of k after PCA')
plt.subplot(3, 1, 2)
plt.bar(k_values, db_scores, color='orange', alpha=0.7)
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Davies-Bouldin Score')
plt.title('Davies-Bouldin Score for Different Values of k after Improv


plt.tight_layout()
plt.show()
```

```
C:\Users\haric\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py
:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\haric\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py
:1412: FutureWarning: The default value of `n_init` will change from
```

```
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\haric\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py
:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\haric\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py
:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

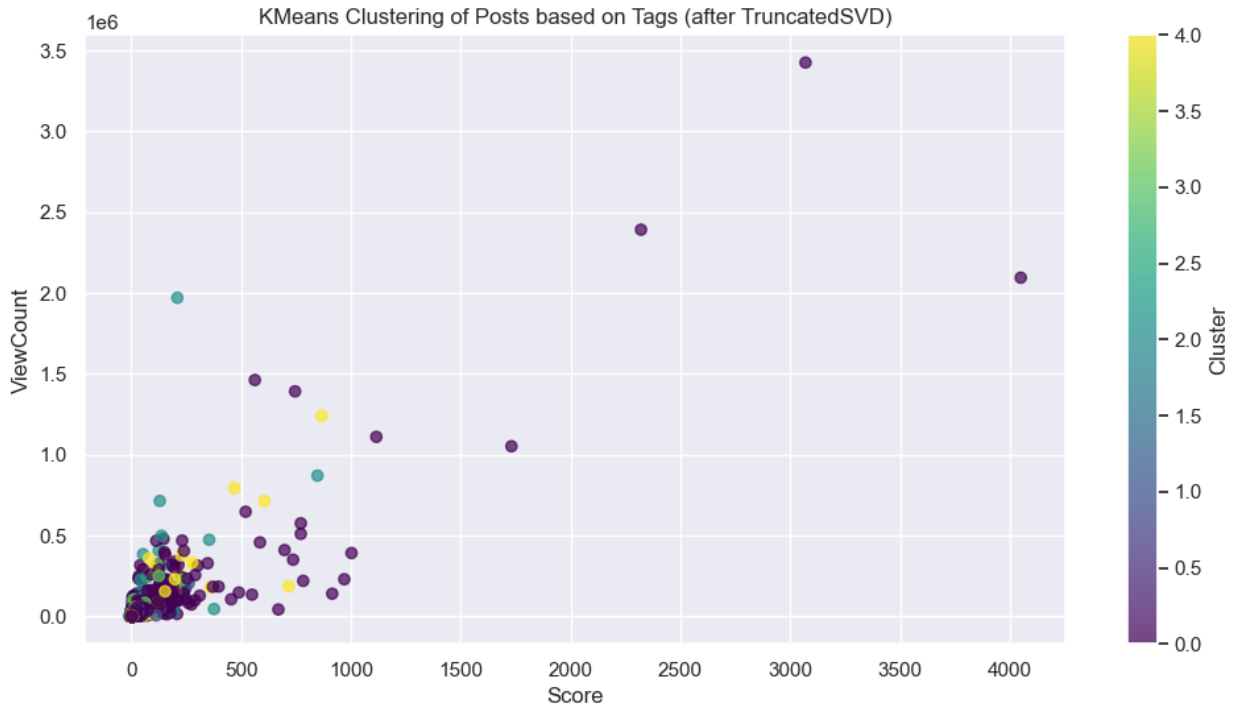




The code commences by excluding posts with missing or None tags and generates a copy of the dataframe for further processing. Utilizing TfidfVectorizer, the text data within the 'Tags' column is transformed into numerical features, limiting the maximum features considered to 100. Subsequently, Principal Component Analysis (PCA) is employed on the TF-IDF matrix of tags to reduce dimensionality while retaining 85% of the explained variance. The script iterates through a range of cluster values from 2 to 5, evaluating clustering performance using metrics such as silhouette scores, Davies-Bouldin scores, and Calinski-Harabasz scores for each k. The k-Means clustering analysis on the posts dataset, incorporating TF-IDF vectorization and PCA dimensionality reduction, yields insights into determining the optimal number of clusters (k) and assessing clustering quality. Key takeaways include understanding silhouette scores' role in cluster definition and separation, lower Davies-Bouldin scores indicating compactness, higher Calinski-Harabasz scores reflecting cluster cohesion, and the significance of visualizing plotted scores to identify an optimal k value for clustering.

In [24]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
df_posts_with_tags = df_all_posts.dropna(subset=['Tags']).copy()
vectorizer = TfidfVectorizer(stop_words='english', max_features=100)
tags_matrix = vectorizer.fit_transform(df_posts_with_tags['Tags'])
svd = TruncatedSVD(n_components=10)
tags_matrix_svd = svd.fit_transform(tags_matrix)
kmeans_tags = KMeans(n_clusters=5, random_state=42)
df_posts_with_tags['Cluster'] = kmeans_tags.fit_predict(tags_matrix_sv
silhouette_avg = silhouette_score(tags_matrix_svd, df_posts_with_tags[
print(f"Silhouette Score: {silhouette_avg}")
plt.figure(figsize=(12, 6))
scatter = plt.scatter(df_posts_with_tags['Score'], df_posts_with_tags[
plt.xlabel('Score')
plt.ylabel('ViewCount')
plt.title('KMeans Clustering of Posts based on Tags (after TruncatedSV
plt.colorbar(scatter, label='Cluster')
plt.show()
```

```
C:\Users\haric\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py
:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)

Silhouette Score: 0.49541260585844077
```



The code snippet provided involves several steps in clustering analysis and visualization based on textual tags within the 'Tags' column of a DataFrame, ensuring effective feature extraction, dimensionality reduction, and subsequent clustering. Firstly, it utilizes the TfidfVectorizer to convert the textual tags into a numerical matrix format, specifically a TF-IDF representation, which quantifies the importance of each term in relation to the documents it appears in. This processed matrix is then subjected to dimensionality reduction using TruncatedSVD, retaining 10 components to capture crucial patterns within the data.

Subsequently, DBSCAN clustering is applied to the reduced matrix, where parameters like 'eps' and 'min_samples' govern the density-based clustering process. The assigned cluster labels are added as a new column named 'Cluster' in the DataFrame, aiding in post-grouping based on their tags. A key metric, the silhouette score, is calculated to gauge the quality of the clustering, offering insight into how well objects fit within their assigned clusters in comparison to other clusters.

The script also includes a visualization component, generating a scatter plot showcasing the clustering outcomes concerning the 'Score' and 'ViewCount' features. Each point in the plot is color-coded based on its assigned cluster, offering a clear visual representation of how posts are grouped concerning these two features after applying DBSCAN clustering following TruncatedSVD dimensionality reduction.

The code concludes by printing the silhouette score and providing a visual representation of post grouping based on their tags, enabling an assessment of clustering quality and the effectiveness of the chosen parameters. Fine-tuning parameters such as 'eps' and 'min_samples' might be necessary depending on the inherent characteristics of the dataset for optimal clustering results.

In [25]:

```python
import pandas as pd
import plotly.express as px
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import MiniBatchKMeans, KMeans
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
df_posts_with_tags = df_all_posts.dropna(subset=['Tags']).copy()
vectorizer = TfidfVectorizer(stop_words='english', max_features=100)
tags_matrix = vectorizer.fit_transform(df_posts_with_tags['Tags'])
k_values = range(2, 11)
silhouette_scores = []

for k in k_values:
    kmeans_tags = MiniBatchKMeans(n_clusters=k, random_state=42)
    df_posts_with_tags['Cluster'] = kmeans_tags.fit_predict(tags_matri
    silhouette_avg = silhouette_score(tags_matrix, df_posts_with_tags[
    silhouette_scores.append(silhouette_avg)
fig = px.line(x=k_values, y=silhouette_scores, labels={'x': 'Number of
              title='Silhouette Score for Different Values of k using
fig.show()
best_k = k_values[silhouette_scores.index(max(silhouette_scores))]
print(f"Best k based on Silhouette Score: {best_k}")
svd = TruncatedSVD(n_components=int(0.85 * tags_matrix.shape[1]))
tags_matrix_svd = svd.fit_transform(tags_matrix)
kmeans_tags_best_k = KMeans(n_clusters=best_k, random_state=42)
df_posts_with_tags['Cluster'] = kmeans_tags_best_k.fit_predict(tags_ma
fig = px.scatter(x=tags_matrix_svd[:, 0], y=tags_matrix_svd[:, 1], col
                 title=f'k-Means Clustering of Posts (k={best_k}) usin
                 labels={'x': 'Component 1', 'y': 'Component 2', 'colo
fig.show()
fig = px.scatter(x=tags_matrix_svd[:, 0], y=tags_matrix_svd[:, 1], col
                 title=f'k-Means Clustering of Posts (k={best_k}) usin
                 labels={'x': 'Component 1', 'y': 'Component 2', 'colo
fig.show()
```

```
C:\Users\haric\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py
:1930: FutureWarning: The default value of `n_init` will change from
3 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=3)
C:\Users\haric\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py
:1930: FutureWarning: The default value of `n_init` will change from
3 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=3)
C:\Users\haric\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py
:1930: FutureWarning: The default value of `n_init` will change from
3 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=3)
C:\Users\haric\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py
:1930: FutureWarning: The default value of `n_init` will change from
```

```
3 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
```

The provided code executes k-Means clustering on a dataset of posts, focusing on the content present in the 'Tags' column and producing a comprehensive outcome. Initially, the script initiates data preparation by eliminating posts with missing or None tags, ensuring a clean dataset, and subsequently creating a copy of the DataFrame for further processing. Using TfidfVectorizer, the text data within the 'Tags' column undergoes conversion into numerical features, limiting the consideration to a maximum of 100 features.

Hyperparameter tuning becomes a pivotal step in the process, where MiniBatchKMeans is employed to explore a range of k values from 2 to 10. Silhouette scores are computed for each k value to ascertain the most suitable number of clusters that best represent the data's inherent structure. A silhouette score plot is generated, visually presenting the silhouette scores across different k values. The peak within this plot indicates the optimal number of clusters, offering a clear insight into the most appropriate k value.

Subsequently, the script selects the best k based on the highest silhouette score, printing this optimal value in the console. Further dimensionality reduction is implemented using TruncatedSVD, reducing the data's dimensionality while retaining 85% of its variance. The reduced-dimensional features are then utilized in executing the final k-Means clustering with the determined optimal k, effectively grouping the posts based on their tag similarities.

The code proceeds to create two distinct visualizations for comprehensive understanding. A 2D scatter plot employing Plotly showcases the clustered posts in the reduced feature space, providing a clear visual representation of how posts are grouped based on their tag similarities. Additionally, an alternative 2D scatter plot is generated using Matplotlib, offering another perspective with color-coded points to visualize the clustered posts.

Overall, these visualizations furnish valuable insights into the grouping patterns of posts, facilitating a better understanding of how posts are categorized based on their tag similarities, thereby aiding in comprehensive data analysis and interpretation.