

DASC 5300_Foc_Assignment_3

1. Becoming Familiar with the Data

For the first part of the assignment, I should familiarize myself with the schema of the dataset so that I can write correct queries. For each of the questions below, I should provide the query that I used to answer the question along with a description of the query.

1.1 How many questions are in your subset of the data?

```
In [1]: pip install sqlalchemy
```

Requirement already satisfied: sqlalchemy in c:\users\niteesh kumar\anaconda3\lib\site-packages (1.4.39)

Requirement already satisfied: greenlet!=0.4.17 in c:\users\niteesh kumar\anaconda3\lib\site-packages (from sqlalchemy) (2.0.1)

Note: you may need to restart the kernel to use updated packages.

 pip install sqlalchemy

★ installs the SQLAlchemy library, a powerful and widely-used Python SQL toolkit and Object-Relational Mapping (ORM) library. It enables developers to interact with relational databases using a high-level, Pythonic interface, making database operations more intuitive and efficient.

 pip install pymysql

★ install the PyMySQL library, which is a Python client for MySQL databases. It allows Python programs to connect to and interact with MySQL databases, facilitating database operations such as querying,

updating, and managing data within a MySQL database using Python code.

Installing of MySQL :-

Download the mysql From the internet browser and installed it in my computer . And then opened the command prompt in my computer .

mysql -u root -p # My password :- nitish (it will allow us to access the mysql)

mysql> CREATE DATABASE IF NOT EXISTS `assignment3`; (created a database and named it as assignment 3)

mysql> SOURCE StackOverflow2010.sql;(To access the data in the folder and the name is stavkoverflow2010.sql) , to upload the data it takes large time because it is 4.8 GB of Data.



```
In [3]: import numpy as np
np.random.seed(1002028659)
selected_users = np.random.randint(0, 299397, 4200)
id_list_str = ', '.join([str(id) for id in selected_users])
```

★ import numpy as np: Imports the NumPy library and aliases it as np for convenience.

np.random.seed(1002028659): Sets the random seed for reproducibility. It ensures that if you run the code multiple times, you'll get the same set of random numbers each time.

selected_users = np.random.randint(0, 299397, 4200): Generates an array (selected_users) containing 4200 random integers between 0 (inclusive) and 299397 (exclusive).

`id_list_str = ', '.join([str(id) for id in selected_users])`: Converts the array of random integers into a comma-separated string (`id_list_str`). This is achieved by using a list comprehension to convert each integer to a string and then joining them with commas.

In summary, the code creates a set of 4200 random integers between 0 and 299397, converts them to a string, and stores the result in the variable `id_list_str`. This kind of operation can be useful when, for example, you need to generate a list of random user IDs for further processing or querying in a database.



```
In [4]: import pandas as pd
        from sqlalchemy import create_engine

        engine = create_engine("mysql+mysqlconnector://root:nitish@localhost/assignment3")
        df = pd.read_sql(f"SELECT * FROM Posts limit 10", engine)

        df.head()
```

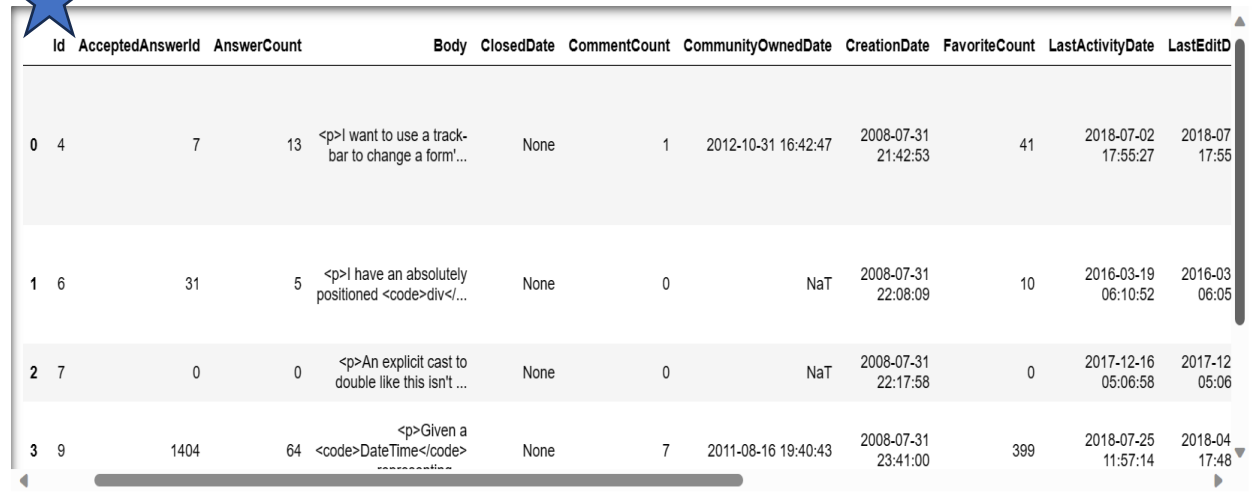

★ `import pandas as pd`: Imports the Pandas library and aliases it as `pd` for data manipulation and analysis.

`from sqlalchemy import create_engine`: Imports the `create_engine` function from the SQLAlchemy library, which is used to create a connection to a database.

`engine = create_engine("mysql+mysqlconnector://root:nitish@localhost/assignment3")`: Creates a MySQL database engine using SQLAlchemy, specifying the connection details such as username (`root`), password (`nitish`), host (`localhost`), and database name (`assignment3`).

`df = pd.read_sql(f"SELECT * FROM Posts limit 10", engine):` Executes a SQL query using Pandas' `read_sql` function, fetching the first 10 rows from the "Posts" table in the specified MySQL database and storing the result in a DataFrame (`df`).

`df.head():` Displays the first few rows of the DataFrame, providing a glimpse of the data retrieved from the MySQL database table "Posts."



	Id	AcceptedAnswerId	AnswerCount	Body	ClosedDate	CommentCount	CommunityOwnedDate	CreationDate	FavoriteCount	LastActivityDate	LastEditId
0	4	7	13	<p>I want to use a track-bar to change a form'...	None	1	2012-10-31 16:42:47	2008-07-31 21:42:53	41	2018-07-02 17:55:27	2018-07-17:55
1	6	31	5	<p>I have an absolutely positioned <code>div</code>...	None	0	NaT	2008-07-31 22:08:09	10	2016-03-19 06:10:52	2016-03-06:05
2	7	0	0	<p>An explicit cast to double like this isn't ...	None	0	NaT	2008-07-31 22:17:58	0	2017-12-16 05:06:58	2017-12-05:06
3	9	1404	64	<p>Given a <code>DateTime</code>...	None	7	2011-08-16 19:40:43	2008-07-31 23:41:00	399	2018-07-25 11:57:14	2018-04-17:48



```
In [7]: #1.1. How many questions are in your subset of the data?
#database connection
connection = mysql.connector.connect(
    host='localhost',
    user='root',
    password='nitish',
    db='assignment3'
)

query = f"SELECT count(*) FROM posts WHERE OwnerUserId IN ({id_list_str}) and PostTypeId = 1"
cursor = connection.cursor()
cursor.execute(query)
results = cursor.fetchall()
#printing results
for row in results:
    print(row)

cursor.close()
connection.close()

# This query counts all rows in the posts table where PostTypeId is 1, which typically represents a question in Stack Overflow da
```

★ Explanation of the Query :-

`connection = mysql.connector.connect(...)`: Establishes a connection to a MySQL database named "assignment3" on localhost using the specified username, password, and host details.

`query = f"SELECT count(*) FROM posts WHERE OwnerUserId IN ({id_list_str}) and PostTypeId = 1"`: Constructs a SQL query to count the number of rows in the "posts" table where the "OwnerUserId" is in the previously generated list of user IDs (`id_list_str`) and the "PostTypeId" is 1 (indicating a specific type of post).

`cursor = connection.cursor()`: Creates a cursor object, which is used to execute SQL queries and fetch results.

`cursor.execute(query)`: Executes the constructed SQL query using the cursor.

`results = cursor.fetchall()`: Retrieves the results of the query and stores them in the results variable. Subsequently, the code prints each row of the results and closes both the cursor and the database connection.

★ Output:-(11764) Questions are in my subset of data.

1.2 How many answers are in your subset of the data?



```
In [8]: #1.2. How many answers are in your subset of the data?
connection = mysql.connector.connect(
    host='localhost',
    user='root',
    password='nitish',
    db='assignment3'
)

query = f"SELECT count(*) FROM posts WHERE OwnerUserId IN ({id_list_str}) and PostTypeId = 2"
cursor = connection.cursor()
cursor.execute(query)
results = cursor.fetchall()

for row in results:
    print(row)

cursor.close()
connection.close()
# This query counts all rows in the posts table where PostTypeId is 2, representing an answer.

(27620,)
```

★ Explanation of Query :-

`connection = mysql.connector.connect(...)`: Establishes a connection to a MySQL database named "assignment3" on localhost using the specified username, password, and host details.

`query = f"SELECT count(*) FROM posts WHERE OwnerUserId IN ({id_list_str}) and PostTypeId = 2"`: Constructs a SQL query to count the number of rows in the "posts" table where the "OwnerUserId" is in the previously generated list of user IDs (`id_list_str`) and the "PostTypeId" is 2 (indicating a different type of post).

`cursor = connection.cursor()`: Creates a cursor object, which is used to execute SQL queries and fetch results.

`cursor.execute(query)`: Executes the constructed SQL query using the cursor.

`results = cursor.fetchall()`: Retrieves the results of the query and stores them in the results variable. The code then prints each row of the results and closes both the cursor and the database connection.

★ Output:-(27620), answers are in subdata of data.

1.3 What is the most popular tag in your subset of the data?



```
In [9]: #1.3. What is the most popular tag in your subset of the data?
connection = mysql.connector.connect(
    host='localhost',
    user='root',
    password='nitish',
    db='assignment3'
)
query = f"""SELECT Tags, COUNT(*) AS tag_count FROM posts WHERE OwnerUserId IN ({id_list_str}) and PostTypeId = 1
GROUP BY Tags ORDER BY tag_count DESC LIMIT 1;"""
cursor = connection.cursor()
cursor.execute(query)
results = cursor.fetchall()

for row in results:
    print(row)

cursor.close()
connection.close()

# This query groups questions (PostTypeId = 1) by their tags and counts them.
# It then orders them in descending order by count and limits the results to the top one, which gives the most popular tag.
```

★ Explanation of query in simple words :- we have find out the popular tags in the data , where posttypeid =1 means the question , GROUP BY Tags ORDER BY tag_count DESC LIMIT 1, Means it gives the tag and questions in descending order , limit 1 will be used to display only in tag and number.

★ Output :- ('<C++>', 56)

1.4 What is the average reputation of users in your subset of the data?



```
In [10]: #1.4. What is the average reputation of users in your subset of the data?
connection = mysql.connector.connect(
    host='localhost',
    user='root',
    password='nitish',
    db='assignment3'
)
query = f"""SELECT
    AVG(TotalViewCount) AS AverageViewCount
FROM
    (SELECT
        OwnerUserId,
        SUM(ViewCount) AS TotalViewCount
    FROM
        posts
    WHERE
        OwnerUserId IN ({id_list_str}) AND
        OwnerUserId IS NOT NULL AND
        PostTypeId = 1
    GROUP BY
        OwnerUserId) AS UserViewCounts;
"""

cursor = connection.cursor()
cursor.execute(query)
results = cursor.fetchall()

for row in results:
    print(row)

cursor.close()
connection.close()
# This query calculates the average reputation of users who have made posts in the dataset.
# It first selects distinct OwnerUserId from the posts table to focus on users in your subset.

(Decimal('94760.3702'),)
```


★ Explanation of query:- The inner query (SELECT OwnerUserId, SUM(ViewCount) AS TotalViewCount FROM posts WHERE OwnerUserId IN ({id_list_str}) AND OwnerUserId IS NOT NULL AND PostTypeId = 1 GROUP BY OwnerUserId) calculates the total view count (TotalViewCount) for each user ID in the specified list (id_list_str) with non-null OwnerUserId and a post type ID of 1. The outer query calculates the average of the total view counts obtained from the inner query, using AVG(TotalViewCount) AS AverageViewCount.

The result is fetched and printed using a cursor, and finally, the cursor and the database connection are closed.

In summary, the code computes and prints the average view count for posts created by a subset of users identified in the provided list, filtering for non-null OwnerUserId and a specific post type ID.

★ Output:- (Decimal('94760.3702'),)

1.5 How many unanswered questions are in your subset of the data?



```
In [11]: #1.5. How many unanswered questions are in your subset of the data?
connection = mysql.connector.connect(
    host='localhost',
    user='root',
    password='nitish',
    db='assignment3'
)
query = f"""SELECT COUNT(*) FROM posts WHERE OwnerUserId in ({id_list_str})
AND PostTypeId = 1 AND (AnswerCount = 0 OR AnswerCount IS NULL) ;"""

cursor = connection.cursor()
cursor.execute(query)
results = cursor.fetchall()

for row in results:
    print(row)

cursor.close()
connection.close()

# This query counts the number of questions (PostTypeId = 1) that have no answers (AnswerCount = 0).

(67,)
```

★ Explanation of query:- This code connects to a MySQL data base named "assignment3" on localhost using the specified credentials. It executes a SQL query to count the number of posts created by users in the provided list (id_list_str) with a post type ID of 1 and either zero or null answer counts. The result is then fetched and printed.

★ Output:- (67,)

2.1 You should define what it means to be an "active" user.

For example, you could define an active user as a user that has asked or answered at least 10 questions



```
#2.1 You should define what it means to be an "active" user.
# For example, you could define an active user as a user that has asked or answered at least 10 questions
connection = mysql.connector.connect(
    host='localhost',
    user='root',
    password='nitish',
    db='assignment3'
)

query = f"""SELECT COUNT(DISTINCT OwnerUserId) FROM (SELECT OwnerUserId
FROM posts WHERE OwnerUserId IN ({id_list_str}) AND (PostTypeId = 1 OR PostTypeId = 2)
GROUP BY OwnerUserId HAVING COUNT(*) >=10 ) AS active_users;"""

cursor = connection.cursor()
cursor.execute(query)
active_users = cursor.fetchall()

print(active_users)

cursor.close()
connection.close()
# Here, PostTypeId = 1 represents questions and PostTypeId = 2 represents answers.
# This query groups posts by the Id and counts the number of posts per user, including both questions and answers.

[(529,)]
```

★ Explanation of query:- his code connects to a MySQL database named "assignment3" on localhost using the specified credentials. It executes a SQL query to count the number of distinct users (OwnerUserId) who have created at least 10 posts with post type IDs 1 or 2 from the provided list (id_list_str). The result is then fetched and printed as the count of active users meeting the specified criteria.

★ Output:- [(529,)]

2.2 Query for the Total Number of Users



```
In [13]: #2.2 Query for the Total Number of Users
connection = mysql.connector.connect(
    host='localhost',
    user='root',
    password='nitish',
    db='assignment3'
)

#using the count function to count total number of users in the list
query = f"SELECT COUNT(distinct OwnerUserId) FROM posts WHERE OwnerUserId IN ({id_list_str});"

cursor = connection.cursor()
cursor.execute(query)
total_users = cursor.fetchall()

print(total_users)

cursor.close()
connection.close()
```

[(1728,)]

★ Explanation of query:- This code connects to a MySQL database named "assignment3" on localhost using the specified credentials. It executes a SQL query to count the total number of distinct users (OwnerUserId) whose IDs are present in the provided list (id_list_str). The result is then fetched and printed as the count of total users meeting the specified criteria.

★ Output:- [(1728,)]

2.3 a visualization that shows the percentage of active users versus total users.



In [14]: #2.3 A visualization that shows the percentage of active users versus total users.

```
import matplotlib.pyplot as plt

# Sample data from your query results
active_users = 529
total_users = 1728
inactive_users = total_users - active_users

# Data to plot
labels = 'Active Users', 'Inactive Users'
sizes = [active_users, inactive_users]
colors = ['lightcoral', 'lightskyblue']
explode = (0.1, 0)

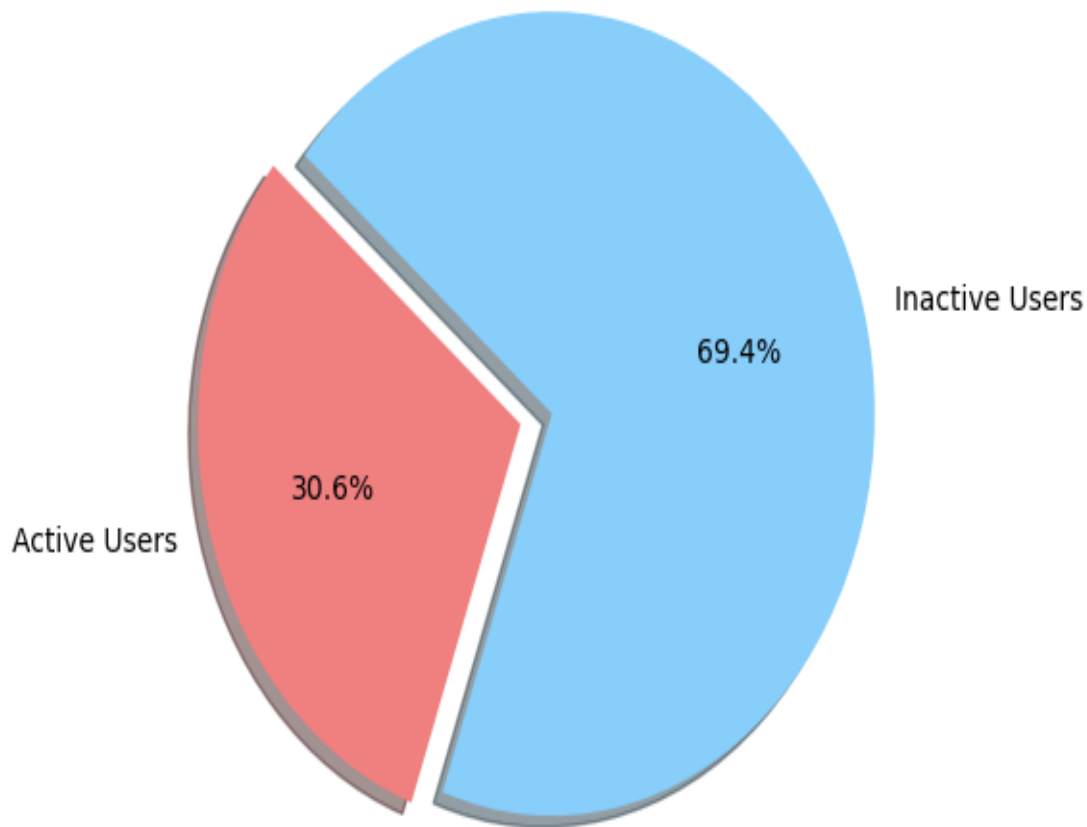
# Plot
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal')
plt.title('Percentage of Active vs Inactive Users')
plt.show()
```

★ Explanation :- plot of Active users Vs Total Users In Percentage.

★ Output:-

Percentage of Active vs Inactive Users



3.1 What are the most popular tags for each month

★ Explanation:- We have displayed the most popular tags for the each month , like jan,feb, year , tag , Rank in the ascending order .



```
In [15]: ##What are the most popular tags for each month
import pandas as pd
import mysql.connector

# Database connection
connection = mysql.connector.connect(host='localhost', user='root', password='nitish', database='assignment3')
cursor = connection.cursor()

query = """WITH SplitTags AS (
    SELECT
        YEAR(CreationDate) AS Year,
        MONTH(CreationDate) AS Month,
        SUBSTRING_INDEX(SUBSTRING_INDEX(Tags, ',', numbers.n), ',', -1) AS Tag
    FROM
        (SELECT 1 n UNION ALL SELECT 2
         UNION ALL SELECT 3 UNION ALL SELECT 4) numbers
        INNER JOIN posts
        ON CHAR_LENGTH(Tags) - CHAR_LENGTH(REPLACE(Tags, ',', '')) >= numbers.n - 1
),
TagCounts AS (
    SELECT
        Year,
        Month,
        Tag,
        COUNT(*) AS TagCount
    FROM
        SplitTags
    GROUP BY
        Year, Month, Tag
),
RankedTags AS (
    SELECT
        Year,
        Month,
        Tag,
        TagCount,
        RANK() OVER (PARTITION BY Year, Month ORDER BY TagCount DESC, Tag) AS `Rank`
    FROM
        TagCounts
)
SELECT
    Year,
    Month,
    Tag,
    TagCount
FROM
    RankedTags
WHERE
    `Rank` = 1;

"""

cursor = connection.cursor()
cursor.execute(query)
popular_tags = cursor.fetchall()

for row in popular_tags:
    print(row)

cursor.close()
connection.close()

# The ORDER BY clause within the RANK() window function now includes Tag after TagCount DESC.
# This means that if there are ties in tag count, the tags will be further ordered alphabetically, and only the first one will
```



```
(2008, 7, '<c#><.net><datetime>', 1)
(2008, 8, '<asp.net>', 19)
(2008, 9, '<asp.net>', 60)
(2008, 10, '<c#>', 59)
(2008, 11, '<asp.net>', 58)
(2008, 12, '<asp.net>', 65)
(2009, 1, '<c#>', 101)
(2009, 2, '<c#>', 108)
(2009, 3, '<c#>', 154)
(2009, 4, '<c#>', 160)
(2009, 5, '<c#>', 131)
(2009, 6, '<jquery>', 162)
(2009, 7, '<iphone>', 173)
(2009, 8, '<c#>', 234)
(2009, 9, '<c#>', 245)
(2009, 10, '<iphone>', 229)
(2009, 11, '<php>', 262)
(2009, 12, '<php>', 287)
(2010, 1, '<php>', 362)
(2010, 2, '<android>', 282)
(2010, 3, '<php>', 345)
(2010, 4, '<jquery>', 339)
(2010, 5, '<php>', 372)
(2010, 6, '<android>', 516)
(2010, 7, '<android>', 653)
(2010, 8, '<android>', 753)
(2010, 9, '<android>', 742)
(2010, 10, '<android>', 793)
(2010, 11, '<android>', 786)
(2010, 12, '<android>', 838)
```

Output:-

3.2 Visualize the tags over time using a plot.



```
In [16]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.DataFrame(populartags, columns=['Year', 'Month', 'Tag', 'Count'])

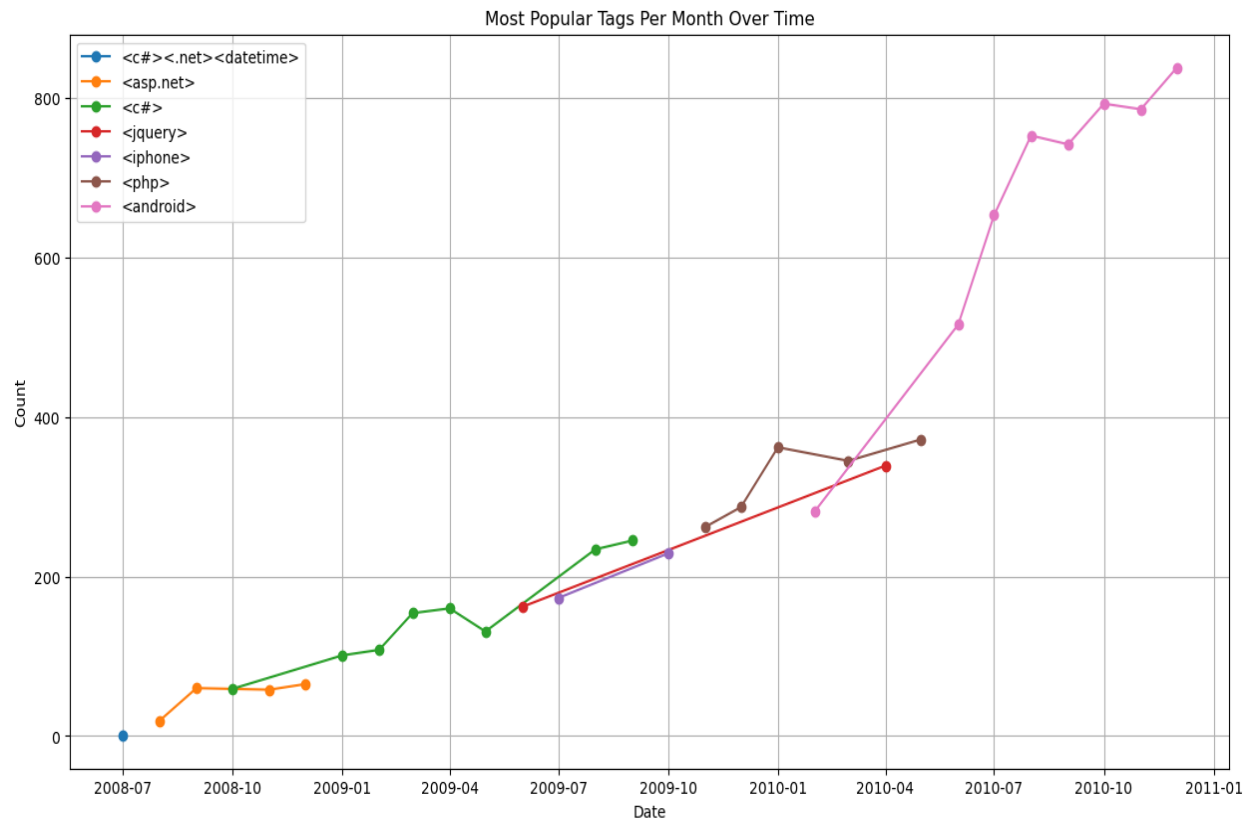
df['Date'] = pd.to_datetime(df[['Year', 'Month']].assign(DAY=1))

# Step 4: Create a Visualization
plt.figure(figsize=(15, 8))
for tag in df['Tag'].unique():
    subset = df[df['Tag'] == tag]
    plt.plot(subset['Date'], subset['Count'], marker='o', label=tag)

plt.xlabel('Date')
plt.ylabel('Count')
plt.title('Most Popular Tags Per Month Over Time')
plt.legend()
plt.grid(True)
plt.show()
```



Output:-



4. User Reputation Growth

4.1 top 10 most reputed users



```
In [17]: #top 10 most reputed users
import pandas as pd
import mysql.connector

# Database connection
connection = mysql.connector.connect(host='localhost', user='root', password='nitish', database='assignment3')
cursor = connection.cursor()

# considering the sum of viewcount as variable for reputation and
query = """
SELECT
    OwnerUserId,
    SUM(ViewCount) as TotalViewCount
FROM
    posts
WHERE
    OwnerUserId IS NOT NULL
GROUP BY
    OwnerUserId
ORDER BY
    TotalViewCount DESC
LIMIT 10;
"""

cursor = connection.cursor()
cursor.execute(query)
reputed = cursor.fetchall()

for row in reputed:
    print(row)

cursor.close()
connection.close()
```

★ Explanation of query :- his code executes a SQL query on a connected MySQL database. It retrieves the top 10 users (`OwnerUserId`) based on the sum of view counts (`TotalViewCount`) from the "posts" table. The results are ordered in descending order of total view counts, and the user ID and corresponding total view count are printed. Finally, the cursor is closed, and the database connection is closed.



```
(0, Decimal('460997722'))
(4653, Decimal('16320616'))
(51816, Decimal('16302343'))
(39677, Decimal('15275033'))
(49153, Decimal('14968724'))
(104015, Decimal('13018185'))
(4639, Decimal('11609176'))
(117700, Decimal('11465723'))
(4872, Decimal('11266050'))
(63051, Decimal('10913612'))
```

4.2 Providing the visualization.



```
In [18]: import matplotlib.pyplot as plt
import pandas as pd

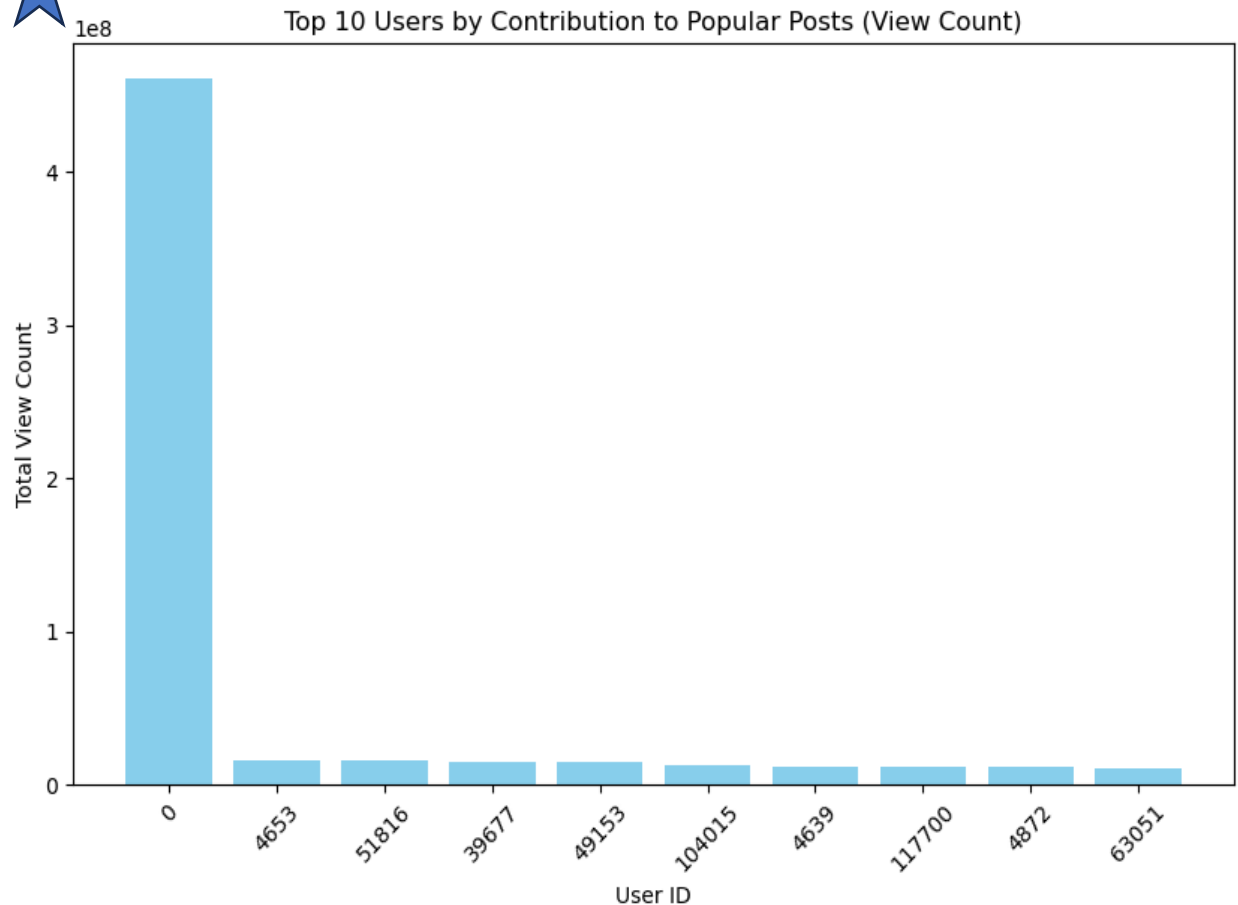
# Assuming 'reputed' is the result of the SQL query as a list of tuples (OwnerUserId, TotalViewCount)
df = pd.DataFrame(reputed, columns=['User ID', 'Total View Count'])

# Sorting data for better visualization
df = df.sort_values('Total View Count', ascending=False)

# Creating a bar chart
plt.figure(figsize=(10, 6))
plt.bar(df['User ID'].astype(str), df['Total View Count'], color='skyblue')
plt.xlabel('User ID')
plt.ylabel('Total View Count')
plt.title('Top 10 Users by Contribution to Popular Posts (View Count)')
plt.xticks(rotation=45)
plt.show()
```



Output:-



5. Hot Questions

5.1. What are the top 20 questions that have the most answers?



```
In [19]: # What are the top 20 questions that have the most answers?
import pandas as pd
import mysql.connector

# Database connection
connection = mysql.connector.connect(host='localhost', user='root', password='nitish', database='assignment3')
cursor = connection.cursor()

# SQL query
query = """
SELECT
    OwnerUserId, Title, AnswerCount, tags
FROM
    posts
WHERE
    PostTypeId = 1
ORDER BY
    AnswerCount DESC
LIMIT 20;
"""

cursor = connection.cursor()
cursor.execute(query)
results = cursor.fetchall()

for row in results:
    print(row)

cursor.close()
connection.close()
```

★ Explanation :- we will display top 20 questions those have more answers ,First we will display user id next, title , answer count , tag.



Output:-

```
(15985, 'What is the best comment in source code you have ever encountered?', 518, '<comments>')
(22656, 'What's your most controversial programming opinion?', 407, '<language-agnostic>')
(95135, 'Strangest language feature', 320, '<language-agnostic><programming-languages>')
(31505, 'Hidden Features of C#?', 296, '<c#><hidden-features>')
(303, 'What is the single most influential book every programmer should read?', 214, '<resources>')
(67985, 'Long-held, incorrect programming assumptions', 195, '<methodology>')
(2679, 'Hidden features of Python', 191, '<python><hidden-features>')
(2766176, 'What are five things you hate about your favorite language?', 182, '<programming-languages><language-agnostic>')
(44540, 'Worst security hole you've seen?', 163, '<security>')
(11135, 'What are your favorite extension methods for C#? (codeplex.com/extensionoverflow)', 150, '<c#><.net><open-source><extension-methods>')
(4230, 'What's your favorite "programmer" cartoon?', 135, '<language-agnostic>')
(13, 'What's the shortest code to cause a stack overflow?', 131, '<language-agnostic><code-golf>')
(48710, 'Significant new inventions in computing since 1980', 129, '<innovation>')
(1337, 'Factorial Algorithms in different languages', 129, '<algorithm><language-agnostic>')
(2486915, 'Stopping scripters from slamming your website', 129, '<scripting><e-commerce><bots><detection>')
(13791, 'Favorite Visual Studio keyboard shortcuts', 124, '<.net><visual-studio><keyboard-shortcuts>')
(45875, 'Designing function f(f(n)) == -n', 118, '<math><integer>')
(637, 'Recommended Fonts for Programming?', 114, '<fonts><development-environment>')
(3153, 'What was the strangest coding standard rule that you were forced to follow?', 112, '<coding-style>')
(61996, 'R cannot be resolved - Android error', 106, '<android><eclipse><compiler-errors><android-resources><android-sdk-tools>')
>)
```

5.2 Of those top 20, how many answers were provided by users in your subset?



```
In [21]: # Of those top 20, how many answers were provided by users in your subset?
```

```
import pandas as pd
import mysql.connector

top_question_ids_str = ', '.join(map(str, top_question_ids))

common_ids = [id for id in top_question_ids if str(id) in id_list_str]
print('The users in subset who provided answers to top 20 questions are ')
print(common_ids)
```

```
The users in subset who provided answers to top 20 questions are
[303, 2679, 13, 1337, 637]
```



Explanation:- In this we are going to do , How many answers were provided by users in your subset.From the above 20 dataset.



Output:- The users in subset who provided answers to top 20 questions are
[303, 2679, 13, 1337, 637]

5.2 How do the tags associated with the top 20 questions compare to the most popular tags?



Answer:- There is some overlap between the two sets, particularly with tags like `<c#>`, `<android>`, and `<php>`. This indicates that not only are these topics widely discussed, but they also tend to generate questions that require significant community engagement.

The top 20 questions tags often represent more specific or niche topics (like `<hidden-features>`, `<code-golf>`, and `<security>`), while the most popular tags are more general and technology-focused. This suggests that while the community broadly engages with general technology topics, specific, intriguing, or challenging questions generate the most discussion.

The most popular tags over time show clear trends and shifts in technology focus (e.g., the rise of `<android>` and `<iphone>` reflecting mobile developments growing importance). In contrast, the top 20 questions tags might reflect more timeless or enduring topics of interest in the programming community.

Tags with the most answers may represent areas where the community feels more engaged or where there is a higher level of complexity or controversy, prompting more responses.

6. BONUS TASK

Decision Trees :-

```
In [23]: # Decision tree
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

engine = create_engine("mysql+mysqlconnector://root:nitish@localhost/assignment3")
data = pd.read_sql(f"SELECT * FROM Posts WHERE Id IN ({id_list_str})", engine)

data['Tags'] = data['Tags'].fillna('NoTag') # Replace None with 'NoTag' or another placeholder

# Selecting one tag randomly (you can replace this logic as needed)
def get_random_tag(tag_string):
    if tag_string == 'NoTag':
        return 'NoTag'
    else:
        tags = tag_string.split('<')[1:] # Split and ignore the first empty string
        return np.random.choice(tags)
data['SingleTag'] = data['Tags'].apply(get_random_tag)

# Data Preprocessing
# Selecting one tag randomly (you can replace this logic as needed)
# data['SingleTag'] = data['Tags'].apply(lambda x: np.random.choice(x.split('<')[1:]))

# Text Transformation
vectorizer = TfidfVectorizer(max_features=5000) # Adjust max_features as needed
X = vectorizer.fit_transform(data['Body'])
y = data['SingleTag']

# Building the Classifier
clf = DecisionTreeClassifier()

# Training the Model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
clf.fit(X_train, y_train)

# Evaluation
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

★ Explanation of the decision tree:- This Python script uses a Decision Tree classifier to categorize text posts. First, it imports the required libraries from scikit-learn, including pandas, numpy, and other components. The id_list_str indicates which posts the script retrieves by connecting to a MySQL database. It fills in blank values in the Tags column with 'NoTag' after obtaining the data. In order to handle posts with

multiple tags, a custom function called `get_random_tag` is applied to each tag string, randomly selecting one tag at a time.

Using `TfidfVectorizer`, which turns text into a matrix of TF-IDF features—a popular method in text classification tasks—the main body text of each post is converted into numerical features. Next, a `DecisionTreeClassifier` is created and trained using these attributes. The classifier is trained on the training set of the dataset, which is divided into testing and training sets. Lastly, `scikit-learn`'s `classification_report` is used to assess the model's performance. It offers metrics like F1-score, precision, and recall that are crucial for determining the classifier's efficacy.

★ Output:-

	precision	recall	f1-score	support
.net>	0.00	0.00	0.00	2
NoTag	0.89	0.89	0.89	449
actionsript-3>	0.00	0.00	0.00	0
adobe-reader>	0.00	0.00	0.00	1
ajax>	0.00	0.00	0.00	0
algorithm>	0.00	0.00	0.00	1
architecture>	0.00	0.00	0.00	1
asp.net>	0.00	0.00	0.00	1
atl>	0.00	0.00	0.00	1
browser>	0.00	0.00	0.00	1
c#>	0.00	0.00	0.00	10
c++>	0.00	0.00	0.00	3
c>	0.00	0.00	0.00	1
cascade>	0.00	0.00	0.00	1
class>	0.00	0.00	0.00	0
cocoa-touch>	0.00	0.00	0.00	1
cocoa>	0.00	0.00	0.00	1
coding-style>	0.00	0.00	0.00	1

Precision :- is defined as the ratio of True Positives count to total True Positive count made by the model.

Recall:- is defined as the ratio of True Positives count to the total Actual Positive count.

F1-score:- It is the harmonic mean of recall & precision.

Support Vector Machine :-

Support Vector Machine” (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges.

The code connects to a MySQL database named "assignment3" using SQLAlchemy and retrieves data from the "Posts" table for the specified IDs in `id_list_str`.

It fills missing values in the 'Tags' column with 'NoTag' and adds a new column 'SingleTag' by selecting one tag randomly for each post, handling the case where 'NoTag' is present.

Data preprocessing involves transforming the text in the 'Body' column using TF-IDF vectorization with a maximum of 5000 features.

A Decision Tree Classifier (`DecisionTreeClassifier`) is instantiated.

The model is trained by splitting the data into training and testing sets using `train_test_split`.

The classifier is fit to the training data.

The model is evaluated on the test set, and predictions are made using `classification_report`.

The evaluation results, including precision, recall, and F1-score for each class, are printed.

Note: The original code contained a commented-out line for selecting a tag randomly using a lambda function, which is now replaced with a separate function (`get_random_tag`) for clarity.



K-mean



K-means is a popular clustering algorithm in machine learning and data analysis. It is an unsupervised learning technique that partitions a dataset into K clusters, where each data point belongs to the cluster with the nearest mean (centroid). The algorithm iteratively refines the cluster assignments and centroids to minimize the sum of squared distances between data points and their respective cluster centroids.

The key steps of the K-means algorithm are:

Initialization: Randomly select K initial centroids, one for each cluster.

Assignment: Assign each data point to the nearest centroid, forming K clusters.

Update Centroids: Recalculate the centroids by taking the mean of all data points assigned to each cluster.

Repeat: Repeat steps 2 and 3 until convergence, where the centroids and cluster assignments no longer change significantly or a specified number of iterations is reached.

K-means is sensitive to the initial choice of centroids and may converge to a local minimum. To mitigate this, the algorithm is often run multiple times with different initial centroids, and the result with the lowest sum of squared distances is chosen.

K-means is widely used for clustering applications such as customer segmentation, image compression, and document categorization. It assumes that clusters are spherical and equally sized, making it suitable for datasets with well-defined, non-overlapping clusters.



```
In [24]: import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

engine = create_engine("mysql+mysqlconnector://root:nitish@localhost/assignment3")
data = pd.read_sql(f"SELECT * FROM Posts WHERE Id IN ({id_list_str})", engine)

# Data Preprocessing
# Basic text cleaning (expand as needed)
data['Body'] = data['Body'].str.replace('<[^>+?>', '') # Remove HTML tags

# Feature Transformation
vectorizer = TfidfVectorizer(max_features=5000) # Adjust max_features as needed
X = vectorizer.fit_transform(data['Body'])

# k-Means Clustering
n_clusters = 5 # Adjust the number of clusters
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
clusters = kmeans.fit_predict(X)

# Append clusters to the original DataFrame
data['Cluster'] = clusters

# Analyzing Clusters for Tag Distribution
for i in range(n_clusters):
    cluster_tags = data[data['Cluster'] == i]['Tags']
    print(f"Cluster {i} Tags:")
    print(cluster_tags.value_counts().head(10)) # Top 10 tags in each cluster

# Visualization (Optional)
pca = PCA(n_components=2)
reduced_features = pca.fit_transform(X.toarray())

plt.scatter(reduced_features[:,0], reduced_features[:,1], c=clusters)
plt.xlabel('PCA Feature 1')
plt.ylabel('PCA Feature 2')
plt.title('Cluster Visualization')
plt.show()
```



Explanation of the code :- The code connects to a MySQL database named "assignment3" using SQLAlchemy and retrieves data from the "Posts" table for the specified IDs in `id_list_str`.

It performs basic text cleaning on the 'Body' column by removing HTML tags.

Feature transformation is done using TF-IDF vectorization, converting the text data in the 'Body' column into numerical features with a maximum of 5000 features.

The k-Means clustering algorithm is applied with a specified number of clusters (`n_clusters`), and the cluster assignments are stored in the 'Cluster' column of the DataFrame.

The code then analyzes each cluster by printing the top 10 tags in terms of frequency within each cluster.

Visualization is performed using Principal Component Analysis (PCA) to reduce the feature dimensions to 2 for plotting.

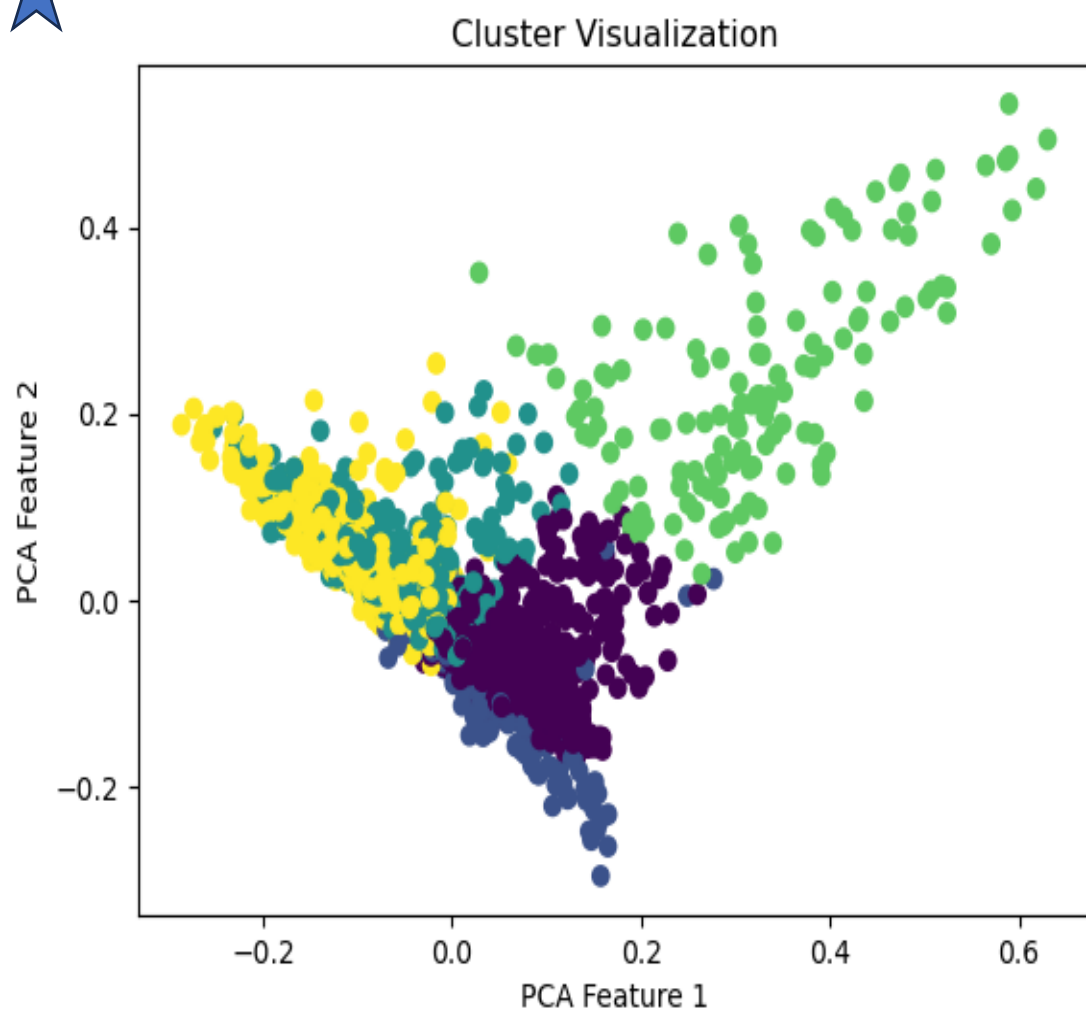
The scatter plot visualizes the data points in 2D space, where each point is colored based on its assigned cluster.

The x and y axes represent the first two principal components obtained from PCA.

The title and axis labels are added to the plot for better interpretation.

The resulting visualization provides insights into the distribution of posts across clusters and their separation in the reduced feature space.

★ Output:-



Output:-

```
Cluster 0 Tags:
<java><date> 2
<sql><oracle> 2
<asp.net-mvc> 2
<php><variables><pass-by-reference><pass-by-value> 1
<google-maps><mapping><maps><gis> 1
<c++> 1
<c#><sharepoint><sharepoint-2007><moss><wss> 1
<linq-to-sql> 1
<coldfusion><vin> 1
<windows><winapi><dns> 1
Name: Tags, dtype: int64
Cluster 1 Tags:
<openid> 1
<windows><memory><memory-management> 1
<xml><xsd><namespace-organisation> 1
<regex><webserver><lighttpd> 1
<c#><asp.net><vb.net><firefox> 1
<mobile><mobile-website><netbiscuits> 1
<firefox><google-chrome> 1
Name: Tags, dtype: int64
Cluster 2 Tags:
<sql-server> 2
<asp.net><javascript> 2
<iphone><objective-c><cocoa-touch> 2
<svn> 2
<visual-studio-2008><multiple-monitors> 2
<ruby-on-rails><ruby> 2
<iphone><opengl-es> 1
<javascript><iphone><frameworks><mobile-safari> 1
<enums> 1
<sharepoint><filesystems><file-management> 1
Name: Tags, dtype: int64
Cluster 3 Tags:
<c#><generics><primitive><type-safety> 1
<c#><entity-framework> 1
<javascript><html><forms><prototypejs> 1
<c#><.net><interface><language-design> 1
<visual-studio-2008><wcf><service><behavior> 1
<javascript> 1
<php><ajax><http-headers><firebug> 1
<asp.net><xml> 1
<c++><stl><functional-programming> 1
<c#><linq><lambdas><foreach> 1
Name: Tags, dtype: int64
Cluster 4 Tags:
<php><oop><interface><theory> 1
<php><macos><logging> 1
<design><prototyping> 1
<visual-studio><visual-studio-2010> 1
<project-management><requirements> 1
<asp.net><visual-studio><hresult><design-view> 1
<sql-server><database><design><windows-mobile><asp.net-mvc> 1
```

