MSR2025_RQ2

November 24, 2024

1 Mining Software Repositories(MSR) 2025 - Mining Challenge

2 Reaserch Question #2 (RQ2)

How can we assess and identify high-risk clusters of artifacts using a composite risk score derived from various risk metrics?

2.0.1 Risk factors to consider

- **Total CVEs per Cluster:**
- **Average CVE Severity Score (1-4 scale: LOW=1, MODERATE=2, HIGH=3, CRITICAL=4):**
- **Average Time Since Last Update:**
- **Average Freshness Score in Days:**
- **Average Popularity Score:**
- **Average Release Interval in Days:**
- **Calculated Risk Score (From above metrics)**

3 Graph Setup

[3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
import json
import os
import re
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from datetime import datetime
```

[90]:
```
from neo4j import GraphDatabase

#connect to the database
uri = "bolt://localhost:7687"
driver = GraphDatabase.driver(uri, auth=("neo4j", "Password1"))
```

[92]:
```
def execute_query(query, parameters=None):
    with driver.session() as session:
        result = session.run(query, parameters or {})
        return result.data()
```

[94]:
```
gds_version = """
CALL gds.version()
"""
print(execute_query(gds_version))
```

```
[{'gdsVersion': '2.6.8'}]
```

3.1 Goblin Weaver

NOTE: Used goblin weaver to update the AddedValues in the Database. Getting updated CVE information from the Open Source Vulnerability (OSV) database. Running Weaver does the following.

- **Automatic Deletion:** Automatically deletes the CVE entries from the neo4J data. Leaves FRESHNESS, POPULARITY_1_YEAR, SPEED as they are internal data.

Goblin Weaver Not Used The database came with added metrics that were current up through 8/30/20204. This will be good enough for my analysis. I found that when trying to use the goblin weaver, it deleted my CVE from added values with the idea that it would be added back by querying updated databases at OSV. I had a hard time getting the weaver to add values back.

3.2 Added Value Nodes

3.2.1 Verify AddedValues Nodes Exist

[14]:
```
# Fetch all node property keys from the database
verify_nodes_query = """
MATCH (n)
RETURN DISTINCT labels(n) AS nodeLabels
"""
result = execute_query(verify_nodes_query)
print(f"Nodes Available: {result}")
```

```
Nodes Available: [{'nodeLabels': ['Artifact']}, {'nodeLabels': ['Release']},
{'nodeLabels': ['AddedValue']}]
```

3.2.2 Find AddedValue Properties

[16]:
```
test_AV_query = """
MATCH (n:AddedValue)
RETURN keys(n) AS properties, COUNT(*) AS count
LIMIT 1
    """
results = execute_query(test_AV_query)
results
```

[16]:
```
[{'properties': ['value', 'id', 'type'], 'count': 28772855}]
```

3.2.3 Find the Possible Types

[19]:
```
test_AV_query = """
MATCH (n:AddedValue)
RETURN DISTINCT n.type AS types
    """
results = execute_query(test_AV_query)
results
```

[19]:
```
[{'types': 'CVE'},
 {'types': 'FRESHNESS'},
 {'types': 'POPULARITY_1_YEAR'},
 {'types': 'SPEED'}]
```

3.2.4 Find One Artifact/Release That Has a CVE Property

[22]:
```
# Helper function to clean and unescape the JSON string
def clean_json_string(json_str):
    # Remove the escaping of quotes if necessary (unescape the string)
    json_str = json_str.replace('\\"', '"')

    # Remove additional escaping if present (optional based on your data)
    json_str = re.sub(r'\\+', '', json_str)

    return json_str
```

[24]:
```
cve_node = """
MATCH (a:Artifact)-[rel]-(r:Release)-[rel2]-(av:AddedValue)
WHERE av.type='CVE' AND av.value CONTAINS 'CVE-'
RETURN a.id AS artifact_id, r.id AS release_id, av.id AS added_value_id, av.
 ↪value AS cve_value
LIMIT 1
"""

results = execute_query(cve_node)

# Process the results
data = []
for record in results:
    cve_value = record['cve_value']

    try:
        # Clean the JSON string before parsing
        cleaned_cve_value = clean_json_string(cve_value)
```

```
        # Attempt to parse the cleaned cve_value JSON string
        cve_data = json.loads(cleaned_cve_value)

        # Extract relevant information from the cve_data (if available)
        cve_entry = cve_data.get("cve", [])[0]  # Assuming we are interested in␣
 ↪the first CVE entry
        cwe = cve_entry.get('cwe', 'N/A')       # Extract CWE
        severity = cve_entry.get('severity', 'N/A')  # Extract severity
        cve_name = cve_entry.get('name', 'N/A')  # Extract CVE name

    except (json.JSONDecodeError, IndexError, KeyError) as e:
        # Handle and log JSON parsing errors or missing fields
        print(f"Error parsing JSON for record: {cve_value}, Error: {e}")
        cwe = 'N/A'
        severity = 'N/A'
        cve_name = 'N/A'

    # Append the processed data to the list
    data.append({
        'Artifact ID': record['artifact_id'],
        'Release ID': record['release_id'],
        'Added Value ID': record['added_value_id'],
        'CVE Name': cve_name,
        'CWE': cwe,
        'Severity': severity
    })

# Create a pandas DataFrame
df = pd.DataFrame(data)

# Display the DataFrame
print(df)
```

```
   Artifact ID                              Release ID  \
0  junit:junit  org.keycloak:keycloak-core:3.4.1.Final

                              Added Value ID        CVE Name        CWE  \
0  org.keycloak:keycloak-core:3.4.1.Final:CVE  CVE-2019-10170  [CWE-267]

  Severity
0     HIGH
```

4 Risk Metrics

4.1 CVE

Total number of known vulnerabilities associated with the artifacts in each cluster. This is a direct measure of potential security risks.
* Identify the artifacts and their releases in each cluster * Cross-reference these artifacts with a CVE database. * Sum the CVEs for each cluster

4.1.1 Retrieve CVE Data for All Relevant Artifacts

[155]:
```
cve_node_query = """
MATCH (a:Artifact)-[rel]->(r:Release)-[rel2]->(av:AddedValue)
WHERE av.type = 'CVE' AND av.value CONTAINS 'CVE-'
RETURN a.id AS artifact_id, r.id AS release_id, av.id AS added_value_id, av.
 ↪value AS cve_value, a.communityId AS communityId
"""

results = execute_query(cve_node_query)
```

4.1.2 Clean and Parse JSON

[157]:
```
def clean_json_string(json_str):
    # Remove any unwanted characters or patterns
    cleaned_str = json_str.replace('\\', '')
    cleaned_str = re.sub(r'"\s*{\s*"', '{"', cleaned_str)
    cleaned_str = re.sub(r'"\s*}\s*"', '"}', cleaned_str)
    return cleaned_str
```

[160]:
```
data = []
for record in results:
    cve_value = record['cve_value']

    try:
        # Clean and parse the JSON string
        cleaned_cve_value = clean_json_string(cve_value)
        cve_data = json.loads(cleaned_cve_value)

        # Extract relevant information from the cve_data
        cve_entries = cve_data.get("cve", [])
        for cve_entry in cve_entries:
            cwe = cve_entry.get('cwe', 'N/A')
            severity = cve_entry.get('severity', 'N/A')
            cve_name = cve_entry.get('name', 'N/A')

            # Append the processed data to the list
            data.append({
                'Artifact ID': record['artifact_id'],
```

```
                'Release ID': record['release_id'],
                'Added Value ID': record['added_value_id'],
                'CVE Name': cve_name,
                'CWE': cwe,
                'Severity': severity,
                'Community ID': record['communityId']
            })

    except (json.JSONDecodeError, IndexError, KeyError, TypeError) as e:
        # Handle and log JSON parsing errors or missing fields
        print(f"Error parsing JSON for record: {cve_value}, Error: {e}")
        # Optionally, append incomplete data or skip
        continue

# Create a pandas DataFrame
df = pd.DataFrame(data)

# Export
df.to_csv('CVE_data.csv', index=False)
```

[162]:
```
print(df.head(5))
```

```
                  Artifact ID                                 Release ID  \
0  org.keycloak:keycloak-core  org.keycloak:keycloak-core:3.4.1.Final
1  org.keycloak:keycloak-core  org.keycloak:keycloak-core:3.4.1.Final
2  org.keycloak:keycloak-core  org.keycloak:keycloak-core:3.4.1.Final
3  org.keycloak:keycloak-core  org.keycloak:keycloak-core:3.4.1.Final
4  org.keycloak:keycloak-core  org.keycloak:keycloak-core:3.4.1.Final

                               Added Value ID        CVE Name  \
0  org.keycloak:keycloak-core:3.4.1.Final:CVE  CVE-2019-10170
1  org.keycloak:keycloak-core:3.4.1.Final:CVE   CVE-2022-0225
2  org.keycloak:keycloak-core:3.4.1.Final:CVE   CVE-2020-1697
3  org.keycloak:keycloak-core:3.4.1.Final:CVE  CVE-2019-14837
4  org.keycloak:keycloak-core:3.4.1.Final:CVE  CVE-2021-20262

                 CWE  Severity  Community ID
0          [CWE-267]      HIGH         10471
1           [CWE-79]  MODERATE         10471
2           [CWE-79]  MODERATE         10471
3  [CWE-547,CWE-798]  CRITICAL         10471
4          [CWE-306]  MODERATE         10471
```

4.1.3 Associate CVE Data with Clusters

[168]:
```
# Group the CVE data by 'Community ID'
cve_by_cluster = df.groupby('Community ID')

# Calculate the total number of CVEs per cluster
total_cves_per_cluster = cve_by_cluster['CVE Name'].count().reset_index()
total_cves_per_cluster = total_cves_per_cluster.rename(columns={'CVE Name':␣
 ↪'Total CVEs'})

# Calculate the total number of unique releases per cluster as this is what the␣
 ↪CVE is tied to.
num_artifacts_per_cluster = df.groupby('Community ID')['Release ID'].nunique().
 ↪reset_index()
num_artifacts_per_cluster = num_artifacts_per_cluster.rename(columns={'Release␣
 ↪ID': 'Num_Releases'})

# List all CVEs per cluster
cves_list_per_cluster = cve_by_cluster['CVE Name'].apply(list).reset_index()
cves_list_per_cluster = cves_list_per_cluster.rename(columns={'CVE Name': 'CVE␣
 ↪List'})

# Merge the total CVEs, CVE lists, and number of artifacts
cluster_cve_data = pd.merge(total_cves_per_cluster, cves_list_per_cluster,␣
 ↪on='Community ID')
cluster_cve_data = pd.merge(cluster_cve_data, num_artifacts_per_cluster,␣
 ↪on='Community ID')

# Export
cluster_cve_data.to_csv('cluster_cve_data.csv', index=False)

# Display the merged data
print("CVE Data per Cluster:")
print(cluster_cve_data.head())
```

```
CVE Data per Cluster:
   Community ID  Total CVEs  \
0           436         121
1           657         401
2          1434          10
3          2682          28
4          2702          30

                                              CVE List  Num_Releases
0  [CVE-2022-25873, CVE-2022-25873, CVE-2022-2587…           121
1  [CVE-2021-32013, CVE-2021-32012, CVE-2021-3201…           171
2  [CVE-2017-3200, CVE-2017-3200, CVE-2017-3200, …            10
3  [CVE-2016-9177, CVE-2018-9159, CVE-2016-9177, …            18
```

4 [CVE-2023-33546, CVE-2023-33546, CVE-2023-3354… 30

4.1.4 Calculate CVE Metrics per Cluster

Verify Severity Scores Exist

[172]:
```
severity_levels = df['Severity'].unique()

print("Different Severity Levels Present in the Data:")
print(severity_levels)
```

```
Different Severity Levels Present in the Data:
['HIGH' 'MODERATE' 'CRITICAL' 'LOW' 'UNKNOWN']
```

Convert Severity Levels to Numerical Scores

[175]:
```
severity_mapping = {
    'LOW': 1,
    'MODERATE': 2,
    'HIGH': 3,
    'CRITICAL': 4,
    'UNKNOWN': 0
}

# Map the severity levels to numerical scores
df['Severity Score'] = df['Severity'].map(severity_mapping)
```

Handle Missing Severity Scores

[178]:
```
# Replace any missing severity scores with 0 or an appropriate default
df['Severity Score'] = df['Severity Score'].fillna(0)
```

Calculate Average Severity per Cluster

[194]:
```
# Calculate the average severity score per cluster
average_severity_score_per_cluster = df.groupby('Community ID')['Severity␣
 ↪Score'].mean().reset_index()
average_severity_score_per_cluster = average_severity_score_per_cluster.
 ↪rename(columns={'Severity Score': 'Average Severity Score'})

# Merge the average severity score into the existing cluster_cve_data DataFrame
cluster_cve_data = pd.merge(cluster_cve_data,␣
 ↪average_severity_score_per_cluster, on='Community ID', how='left')

# Export the updated DataFrame to a CSV file
cluster_cve_data.to_csv('cluster_cve_data.csv', index=False)

# Display the updated DataFrame
print("Updated Cluster CVE Data with Average Severity Score:")
print(cluster_cve_data.head())
```

```
Updated Cluster CVE Data with Average Severity Score:
   Community ID  Total CVEs  \
0           436         121
1           657         401
2          1434          10
3          2682          28
4          2702          30

                                              CVE List  Num_Releases  \
0  [CVE-2022-25873, CVE-2022-25873, CVE-2022-2587…           121
1  [CVE-2021-32013, CVE-2021-32012, CVE-2021-3201…           171
2  [CVE-2017-3200, CVE-2017-3200, CVE-2017-3200, …            10
3  [CVE-2016-9177, CVE-2018-9159, CVE-2016-9177, …            18
4  [CVE-2023-33546, CVE-2023-33546, CVE-2023-3354…            30

   Average Severity Score
0                2.000000
1                2.349127
2                3.000000
3                2.357143
4                2.000000
```

Normalize the Total CVEs

Number of CVEs per release (on average) per cluster

[203]:
```
# Add the normalized Total CVEs as a new column
cluster_cve_data['Normalized Total CVEs'] = (
    cluster_cve_data['Total CVEs'] / cluster_cve_data['Num_Releases']
)

# Export the updated DataFrame to a CSV file
cluster_cve_data.to_csv('cluster_cve_data.csv', index=False)
```

Display the Metrics

[206]:
```
print(cluster_cve_data.sort_values(by='Average Severity Score',␣
 ↪ascending=False).head(20))
```

```
     Community ID  Total CVEs  \
75          36717           6
79          39037           4
25          11504          53
59          30752           6
121         57273           4
127         61493          15
101         50022           9
50          27120           3
97          46633           8
81          39418          31
```

135	65986	5
87	41352	3
122	58239	11
54	28969	2270
34	16657	107
136	66039	5
82	39655	998
130	62299	27
118	56718	4
112	53957	45

	CVE List	Num_Releases
75	[CVE-2020-17531, CVE-2020-17531, CVE-2020-1753…	6
79	[CVE-2023-51084, CVE-2023-51084, CVE-2023-5108…	4
25	[CVE-2023-27162, CVE-2023-27162, CVE-2023-2716…	53
59	[CVE-2021-43090, CVE-2021-43090, CVE-2021-4309…	6
121	[CVE-2016-11023, CVE-2016-11024, CVE-2016-1102…	2
127	[CVE-2019-19899, CVE-2019-19899, CVE-2019-1989…	15
101	[CVE-2017-12620, CVE-2017-12620, CVE-2017-1262…	9
50	[CVE-2015-7501, CVE-2015-7501, CVE-2015-7501]	3
97	[CVE-2014-3579, CVE-2014-3579, CVE-2014-3579, …	8
81	[CVE-2016-5019, CVE-2016-5019, CVE-2016-5019, …	31
135	[CVE-2021-23369, CVE-2021-23369, CVE-2021-2336…	5
87	[CVE-2018-1000836, CVE-2018-1000836, CVE-2018-…	3
122	[CVE-2023-0869, CVE-2018-1000823, CVE-2018-100…	11
54	[CVE-2019-5312, CVE-2019-5312, CVE-2019-5312, …	1341
34	[CVE-2021-39239, CVE-2021-39239, CVE-2021-3923…	107
136	[CVE-2022-23302, CVE-2021-4104, CVE-2019-17571…	1
82	[CVE-2024-25603, CVE-2024-25601, CVE-2024-2514…	252
130	[CVE-2022-26112, CVE-2022-23974, CVE-2022-2611…	14
118	[CVE-2021-44585, CVE-2021-46089, CVE-2022-2288…	1
112	[CVE-2021-38555, CVE-2022-25312, CVE-2023-3415…	13

	Average Severity Score	Normalized Total CVEs
75	4.000000	1.000000
79	4.000000	1.000000
25	4.000000	1.000000
59	4.000000	1.000000
121	4.000000	2.000000
127	4.000000	1.000000
101	4.000000	1.000000
50	4.000000	1.000000
97	4.000000	1.000000
81	4.000000	1.000000
135	4.000000	1.000000
87	4.000000	1.000000
122	3.818182	1.000000
54	3.691189	1.692767

```
34                  3.663551                1.000000
136                 3.600000                5.000000
82                  3.596192                3.960317
130                 3.518519                1.928571
118                 3.500000                4.000000
112                 3.422222                3.461538
```

[208]:
```python
# Count the number of CVEs per severity level
severity_counts = df['Severity'].value_counts()

print("Number of CVEs per Severity Level:")
print(severity_counts)
```

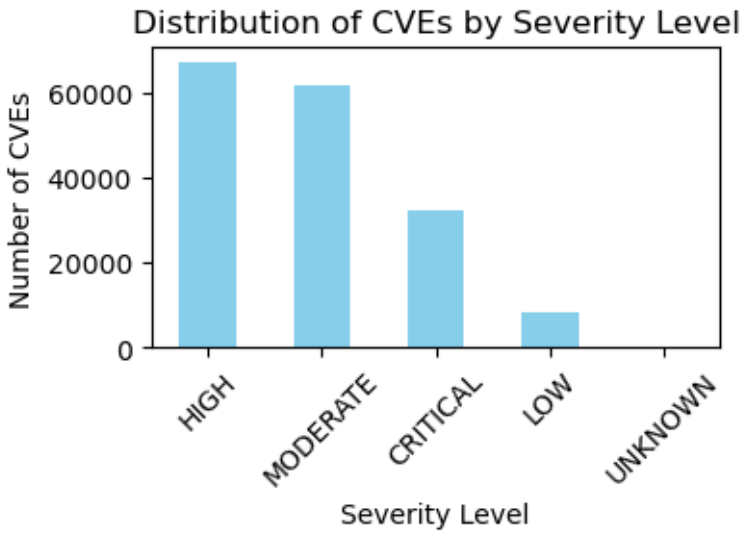
```
Number of CVEs per Severity Level:
Severity
HIGH        67235
MODERATE    61643
CRITICAL    32304
LOW          8008
UNKNOWN         3
Name: count, dtype: int64
```

Visualizing Severity Distributions

[211]:
```python
# Plot the number of CVEs per severity level
plt.figure(figsize=(4, 3))
severity_counts.plot(kind='bar', color='skyblue')
plt.xlabel('Severity Level')
plt.ylabel('Number of CVEs')
plt.title('Distribution of CVEs by Severity Level')
plt.xticks(rotation=45)
plt.tight_layout()

# Export Plot
plt.savefig('cve_severity_distribution.pdf', bbox_inches='tight')

# Display
plt.show()
```



Create Cluster_Risk_Metrics External File

[213]:
```
cluster_risk_metrics = cluster_cve_data
```

[215]:
```
# Save the DataFrame to a Pickle file (more efficient for large data)
cluster_risk_metrics.to_pickle('cluster_cve_metrics.pkl')
print("Data saved to 'cluster_cve_metrics.pkl'")
```

```
Data saved to 'cluster_cve_metrics.pkl'
```

Load pickle file for verification

[218]:
```
# Load the DataFrame from the pickle file
cluster_risk_metrics = pd.read_pickle('cluster_cve_metrics.pkl')

print("DataFrame loaded from 'cluster_cve_metrics.pkl'")

# Print the first few lines of the DataFrame
print(cluster_risk_metrics.head())
```

```
DataFrame loaded from 'cluster_cve_metrics.pkl'
   Community ID  Total CVEs  \
0           436         121
1           657         401
2          1434          10
3          2682          28
4          2702          30

                                              CVE List  Num_Releases  \
0  [CVE-2022-25873, CVE-2022-25873, CVE-2022-2587…           121
1  [CVE-2021-32013, CVE-2021-32012, CVE-2021-3201…           171
```

```
2  [CVE-2017-3200, CVE-2017-3200, CVE-2017-3200, …            10
3  [CVE-2016-9177, CVE-2018-9159, CVE-2016-9177, …            18
4  [CVE-2023-33546, CVE-2023-33546, CVE-2023-3354…            30

   Average Severity Score  Normalized Total CVEs
0                2.000000               1.000000
1                2.349127               2.345029
2                3.000000               1.000000
3                2.357143               1.555556
4                2.000000               1.000000
```

[220]:
```python
# Assuming 'df' is your DataFrame
num_rows = cluster_risk_metrics.shape[0]
num_cols = cluster_risk_metrics.shape[1]
print(f"The DataFrame has {num_rows} rows and {num_cols} columns.")
```

```
The DataFrame has 139 rows and 6 columns.
```

4.2 UPDATE

Get total Artifact Records

[106]:
```python
def get_total_update_artifact_records():
    query = """
    MATCH (a:Artifact)
    RETURN count(a) AS total_records
    """
    result = execute_query(query)
    return result[0]['total_records']

total_records = get_total_update_artifact_records()
print(f"Total artifact records: {total_records}")
```

```
Total artifact records: 658078
```

[108]:
```python
def get_artifact_update_data_batch(skip, limit):
    query = f"""
    MATCH (a:Artifact)-[:relationship_AR]->(r:Release)
    WITH a, max(r.timestamp) AS last_update_timestamp
    RETURN a.id AS artifact_id, a.communityId AS community_id,␣
 ↪last_update_timestamp
    SKIP {skip}
    LIMIT {limit}
    """
    results = execute_query(query)
    return pd.DataFrame(results)
```

Get Data In Batches

[111]:
```
batch_size = 100000  # Adjust as needed
processed_batches = []
for skip in range(0, total_records, batch_size):
    print(f"Processing records {skip} to {min(skip + batch_size - 1,␣
 ↪total_records - 1)}")
    # Fetch the batch
    batch_df = get_artifact_update_data_batch(skip, batch_size)

    if batch_df.empty:
        continue

    # Convert 'last_update_timestamp' to datetime
    batch_df['Last Update Date'] = pd.
 ↪to_datetime(batch_df['last_update_timestamp'], unit='ms')

    # Calculate 'Days Since Last Update'
    current_date = datetime.now()
    batch_df['Days Since Last Update'] = (current_date - batch_df['Last Update␣
 ↪Date']).dt.days

    # Append to the list of processed batches
    processed_batches.append(batch_df[['artifact_id', 'community_id', 'Days␣
 ↪Since Last Update']])
```

```
Processing records 0 to 99999
Processing records 100000 to 199999
Processing records 200000 to 299999
Processing records 300000 to 399999
Processing records 400000 to 499999
Processing records 500000 to 599999
Processing records 600000 to 658077
```

Combine Batches into Single DataFrame

[116]:
```
# Combine batches into a single DataFrame
release_df = pd.concat(processed_batches, ignore_index=True)

# Export
release_df.to_csv('Update_data.csv', index=False)
```

Aggregate data per cluster (Community ID)

[119]:
```
cluster_update_metrics = release_df.groupby('community_id').agg({
    'Days Since Last Update': 'mean'
}).reset_index()
```

Rename Columns for Clarity

[122]:
```
# Rename columns for clarity
cluster_update_metrics = cluster_update_metrics.rename(columns={
    'community_id': 'Community ID',
    'Days Since Last Update': 'Average Days Since Last Update'
})
```

Display Top Clusters

[125]:
```
# Display the top clusters with the highest average days since last update
print(cluster_update_metrics.sort_values(by='Average Days Since Last Update',␣
 ↪ascending=False).head(10))
```

```
       Community ID  Average Days Since Last Update
64178         64216                          7559.0
13969         13973                          7234.0
40951         40974                          7234.0
59586         59624                          7168.0
27326         27338                          7168.0
28401         28415                          7168.0
60156         60194                          7168.0
54252         54287                          7145.0
50636         50669                          7145.0
54684         54719                          7145.0
```

Save to External File

[127]:
```
cluster_update_metrics.to_csv('cluster_update_data.csv', index=False)

cluster_update_metrics.to_pickle('cluster_update_metrics.pkl')
print("Data saved to 'cluster_update_metrics.pkl'")
```

```
Data saved to 'cluster_update_metrics.pkl'
```

Load External File For Testing

[131]:
```
# Load the DataFrame from the pickle file
cluster_update_metrics = pd.read_pickle('cluster_update_metrics.pkl')

print("DataFrame loaded from 'cluster_update_metrics.pkl'")

# Print the first few lines of the DataFrame
print(cluster_update_metrics.head())
```

```
DataFrame loaded from 'cluster_update_metrics.pkl'
   Community ID  Average Days Since Last Update
0             0                          3212.0
1             1                           838.0
2             2                           492.0
3             3                          2346.0
4             4                           438.0
```

4.3 FRESHNESS

4.3.1 Retrieve Freshness Data

Find number of Nodes

[16]:
```
# Query to count the total number of records
def get_total_freshness_records():
    query = """
    MATCH (a:Artifact)-[rel]->(r:Release)-[rel2]->(av:AddedValue)
    WHERE av.type = 'FRESHNESS'
    RETURN count(*) AS total_records
    """
    result = execute_query(query)
    return result[0]['total_records']

total_records = get_total_freshness_records()
print(f"Total FRESHNESS records: {total_records}")
```

```
Total FRESHNESS records: 14459139
```

Get one result to verify filetype

[19]:
```
query = f"""
MATCH (a:Artifact)-[rel]->(r:Release)-[rel2]-(av:AddedValue)
WHERE av.type = 'FRESHNESS'
RETURN a.id AS artifact_id, a.communityId AS community_id, av.value AS VALUE
LIMIT 1
"""
results = execute_query(query)
print(pd.DataFrame(results))
```

```
                                    artifact_id  community_id  \
0  com.softwaremill.sttp.client:core_sjs0.6_2.13         26959

                                               VALUE
0  {\"freshness\":{\"numberMissedRelease\":\"7\",…
```

Fetch Data

[22]:
```
def get_freshness_data_batch(skip, limit):
    query = f"""
    MATCH (a:Artifact)-[rel]->(r:Release)-[rel2]->(av:AddedValue)
    WHERE av.type = 'FRESHNESS' AND av.value IS NOT NULL
    RETURN a.id AS artifact_id, a.communityId AS community_id, av.value AS␣
 ↪freshness_json
    SKIP {skip}
    LIMIT {limit}
    """
    results = execute_query(query)
```

```
    return pd.DataFrame(results)
```

[24]:
```
def parse_freshness(json_str):
    try:
        # Clean the JSON string if necessary
        json_str = json_str.replace('\\', '')
        data = json.loads(json_str)
        freshness = data.get('freshness', {})
        number_missed_release = int(freshness.get('numberMissedRelease', '0'))
        outdated_time_ms = int(freshness.get('outdatedTimeInMs', '0'))
        return number_missed_release, outdated_time_ms
    except (json.JSONDecodeError, ValueError, TypeError) as e:
        return None, None
```

[26]:
```
def process_freshness_batch(batch_df):
    # Apply the parsing function to extract the metrics
    batch_df[['number_missed_release', 'outdated_time_ms']] =␣
 ↪batch_df['freshness_json'].apply(
        lambda x: pd.Series(parse_freshness(x))
    )

    # Ensure numeric types
    batch_df['number_missed_release'] = pd.
 ↪to_numeric(batch_df['number_missed_release'], errors='coerce')
    batch_df['outdated_time_ms'] = pd.to_numeric(batch_df['outdated_time_ms'],␣
 ↪errors='coerce')

    # Drop rows with missing 'outdated_time_ms'
    batch_df = batch_df.dropna(subset=['outdated_time_ms'])

    # Convert milliseconds to days
    batch_df['outdated_time_days'] = batch_df['outdated_time_ms'] / (1000 * 60␣
 ↪* 60 * 24)

    # Return the processed batch
    return batch_df
```

Fetch and Aggregate Data - THIS TAKES A LONG TIME (USE SAVED FILE IF POSSIBLE)

[30]:
```
batch_size = 100000  # Adjust as needed
total_records = get_total_freshness_records()
print(f"Total FRESHNESS records: {total_records}")

processed_batches = []
for skip in range(0, total_records, batch_size):
    print(f"Fetching records {skip} to {min(skip + batch_size - 1,␣
 ↪total_records)}")
```

```
    # Fetch the batch
    batch_df = get_freshness_data_batch(skip, batch_size)

    # Process and parse the batch
    processed_batch = process_freshness_batch(batch_df)

    # Append to the list of processed batches
    processed_batches.append(processed_batch)

    # Clear variables to free up memory
    del batch_df
    del processed_batch

# Concatenate all processed batches into a single DataFrame
freshness_df = pd.concat(processed_batches, ignore_index=True)

# Display the first few rows
print("Processed FRESHNESS Data:")
print(freshness_df.head())
```

```
Total FRESHNESS records: 14459139
Fetching records 0 to 99999
Fetching records 100000 to 199999
Fetching records 200000 to 299999
Fetching records 300000 to 399999
Fetching records 400000 to 499999
Fetching records 500000 to 599999
Fetching records 600000 to 699999
Fetching records 700000 to 799999
Fetching records 800000 to 899999
Fetching records 900000 to 999999
Fetching records 1000000 to 1099999
Fetching records 1100000 to 1199999
Fetching records 1200000 to 1299999
Fetching records 1300000 to 1399999
Fetching records 1400000 to 1499999
Fetching records 1500000 to 1599999
Fetching records 1600000 to 1699999
Fetching records 1700000 to 1799999
Fetching records 1800000 to 1899999
Fetching records 1900000 to 1999999
Fetching records 2000000 to 2099999
Fetching records 2100000 to 2199999
Fetching records 2200000 to 2299999
Fetching records 2300000 to 2399999
Fetching records 2400000 to 2499999
Fetching records 2500000 to 2599999
Fetching records 2600000 to 2699999
```

```
Fetching records 2700000 to 2799999
Fetching records 2800000 to 2899999
Fetching records 2900000 to 2999999
Fetching records 3000000 to 3099999
Fetching records 3100000 to 3199999
Fetching records 3200000 to 3299999
Fetching records 3300000 to 3399999
Fetching records 3400000 to 3499999
Fetching records 3500000 to 3599999
Fetching records 3600000 to 3699999
Fetching records 3700000 to 3799999
Fetching records 3800000 to 3899999
Fetching records 3900000 to 3999999
Fetching records 4000000 to 4099999
Fetching records 4100000 to 4199999
Fetching records 4200000 to 4299999
Fetching records 4300000 to 4399999
Fetching records 4400000 to 4499999
Fetching records 4500000 to 4599999
Fetching records 4600000 to 4699999
Fetching records 4700000 to 4799999
Fetching records 4800000 to 4899999
Fetching records 4900000 to 4999999
Fetching records 5000000 to 5099999
Fetching records 5100000 to 5199999
Fetching records 5200000 to 5299999
Fetching records 5300000 to 5399999
Fetching records 5400000 to 5499999
Fetching records 5500000 to 5599999
Fetching records 5600000 to 5699999
Fetching records 5700000 to 5799999
Fetching records 5800000 to 5899999
Fetching records 5900000 to 5999999
Fetching records 6000000 to 6099999
Fetching records 6100000 to 6199999
Fetching records 6200000 to 6299999
Fetching records 6300000 to 6399999
Fetching records 6400000 to 6499999
Fetching records 6500000 to 6599999
Fetching records 6600000 to 6699999
Fetching records 6700000 to 6799999
Fetching records 6800000 to 6899999
Fetching records 6900000 to 6999999
Fetching records 7000000 to 7099999
Fetching records 7100000 to 7199999
Fetching records 7200000 to 7299999
Fetching records 7300000 to 7399999
Fetching records 7400000 to 7499999
```

```
Fetching records 7500000 to 7599999
Fetching records 7600000 to 7699999
Fetching records 7700000 to 7799999
Fetching records 7800000 to 7899999
Fetching records 7900000 to 7999999
Fetching records 8000000 to 8099999
Fetching records 8100000 to 8199999
Fetching records 8200000 to 8299999
Fetching records 8300000 to 8399999
Fetching records 8400000 to 8499999
Fetching records 8500000 to 8599999
Fetching records 8600000 to 8699999
Fetching records 8700000 to 8799999
Fetching records 8800000 to 8899999
Fetching records 8900000 to 8999999
Fetching records 9000000 to 9099999
Fetching records 9100000 to 9199999
Fetching records 9200000 to 9299999
Fetching records 9300000 to 9399999
Fetching records 9400000 to 9499999
Fetching records 9500000 to 9599999
Fetching records 9600000 to 9699999
Fetching records 9700000 to 9799999
Fetching records 9800000 to 9899999
Fetching records 9900000 to 9999999
Fetching records 10000000 to 10099999
Fetching records 10100000 to 10199999
Fetching records 10200000 to 10299999
Fetching records 10300000 to 10399999
Fetching records 10400000 to 10499999
Fetching records 10500000 to 10599999
Fetching records 10600000 to 10699999
Fetching records 10700000 to 10799999
Fetching records 10800000 to 10899999
Fetching records 10900000 to 10999999
Fetching records 11000000 to 11099999
Fetching records 11100000 to 11199999
Fetching records 11200000 to 11299999
Fetching records 11300000 to 11399999
Fetching records 11400000 to 11499999
Fetching records 11500000 to 11599999
Fetching records 11600000 to 11699999
Fetching records 11700000 to 11799999
Fetching records 11800000 to 11899999
Fetching records 11900000 to 11999999
Fetching records 12000000 to 12099999
Fetching records 12100000 to 12199999
Fetching records 12200000 to 12299999
```

```
Fetching records 12300000 to 12399999
Fetching records 12400000 to 12499999
Fetching records 12500000 to 12599999
Fetching records 12600000 to 12699999
Fetching records 12700000 to 12799999
Fetching records 12800000 to 12899999
Fetching records 12900000 to 12999999
Fetching records 13000000 to 13099999
Fetching records 13100000 to 13199999
Fetching records 13200000 to 13299999
Fetching records 13300000 to 13399999
Fetching records 13400000 to 13499999
Fetching records 13500000 to 13599999
Fetching records 13600000 to 13699999
Fetching records 13700000 to 13799999
Fetching records 13800000 to 13899999
Fetching records 13900000 to 13999999
Fetching records 14000000 to 14099999
Fetching records 14100000 to 14199999
Fetching records 14200000 to 14299999
Fetching records 14300000 to 14399999
Fetching records 14400000 to 14459139
Processed FRESHNESS Data:
                                    artifact_id  community_id  \
0  com.softwaremill.sttp.client:core_sjs0.6_2.13         26959
1                          com.ibeetl:act-sample         29418
2  com.softwaremill.sttp.client:core_sjs0.6_2.13         26959
3                    com.lihaoyi:ammonite_2.12.1         26959
4                  com.yahoo.vespa:container-disc         50022

                                      freshness_json  number_missed_release  \
0  {\"freshness\":{\"numberMissedRelease\":\"7\",…                       7
1  {\"freshness\":{\"numberMissedRelease\":\"2\",…                       2
2  {\"freshness\":{\"numberMissedRelease\":\"9\",…                       9
3  {\"freshness\":{\"numberMissedRelease\":\"367\…                     367
4  {\"freshness\":{\"numberMissedRelease\":\"448\…                     448

   outdated_time_ms  outdated_time_days
0        3795765000           43.932465
1       11941344000          138.210000
2        4685281000           54.227789
3      142773884000         1652.475509
4      105191360000         1217.492593
```

[32]:
```python
# Display the columns and first few rows
print("Columns in freshness_df after modifying the query:")
print(freshness_df.columns)
```

```
print("First few rows of freshness_df:")
print(freshness_df.head())
```

```
Columns in freshness_df after modifying the query:
Index(['artifact_id', 'community_id', 'freshness_json',
       'number_missed_release', 'outdated_time_ms', 'outdated_time_days'],
      dtype='object')
First few rows of freshness_df:
                                    artifact_id  community_id  \
0  com.softwaremill.sttp.client:core_sjs0.6_2.13         26959
1                          com.ibeetl:act-sample         29418
2  com.softwaremill.sttp.client:core_sjs0.6_2.13         26959
3                   com.lihaoyi:ammonite_2.12.1         26959
4                com.yahoo.vespa:container-disc         50022

                                      freshness_json  number_missed_release  \
0  {\"freshness\":{\"numberMissedRelease\":\"7\",…                       7
1  {\"freshness\":{\"numberMissedRelease\":\"2\",…                       2
2  {\"freshness\":{\"numberMissedRelease\":\"9\",…                       9
3  {\"freshness\":{\"numberMissedRelease\":\"367\…                     367
4  {\"freshness\":{\"numberMissedRelease\":\"448\…                     448

   outdated_time_ms  outdated_time_days
0        3795765000           43.932465
1       11941344000          138.210000
2        4685281000           54.227789
3      142773884000         1652.475509
4      105191360000         1217.492593
```

[151]:
```
# Export Data
freshness_df.to_csv('Freshness_data.csv', index=False)
```

4.3.2 Data Analysis - Freshness

Aggregating FRESHNESS Metrics per Cluster

[36]:
```
# Group by community ID and calculate the average freshness (in days) per␣
 ↪cluster
cluster_freshness = freshness_df.groupby('community_id').agg({
    'outdated_time_days': 'mean'
}).reset_index()

# Rename columns for clarity
cluster_freshness = cluster_freshness.rename(columns={
    'community_id': 'Community ID',
    'outdated_time_days': 'Average Freshness (Days)'
})
```

```
# Display the first few rows
print("Average Freshness per Cluster:")
print(cluster_freshness.head(10))
```

```
Average Freshness per Cluster:
   Community ID  Average Freshness (Days)
0             0                 31.300386
1             1                195.021071
2             2                  0.000000
3             3                  0.000000
4             4                  0.000000
5             5                  0.000000
6             6                322.837211
7             7                  0.000000
8             8                  7.664687
9             9                  0.000000
```

Save Results Locally - As Pickle Structure

[38]:
```
# Save the DataFrame to a Pickle file (more efficient for large data)
cluster_freshness.to_pickle('cluster_freshness_metrics.pkl')
print("Data saved to 'cluster_freshness_metrics.pkl'")
```

```
Data saved to 'cluster_freshness_metrics.pkl'
```

[40]:
```
# Export as csv
cluster_freshness.to_csv('cluster_freshness_data.csv', index=False)
```

Load Data

[42]:
```
# Load the DataFrame from the Pickle file
cluster_freshness = pd.read_pickle('cluster_freshness_metrics.pkl')
print("Data loaded from 'cluster_freshness_metrics.pkl'")

# Print the first few lines of the DataFrame
print(freshness_df.head())
```

```
Data loaded from 'cluster_freshness_metrics.pkl'
                                    artifact_id  community_id  \
0  com.softwaremill.sttp.client:core_sjs0.6_2.13         26959
1                          com.ibeetl:act-sample         29418
2  com.softwaremill.sttp.client:core_sjs0.6_2.13         26959
3                    com.lihaoyi:ammonite_2.12.1         26959
4                  com.yahoo.vespa:container-disc         50022

                                      freshness_json  number_missed_release  \
0  {\"freshness\":{\"numberMissedRelease\":\"7\",…                      7
1  {\"freshness\":{\"numberMissedRelease\":\"2\",…                      2
2  {\"freshness\":{\"numberMissedRelease\":\"9\",…                      9
3  {\"freshness\":{\"numberMissedRelease\":\"367\…                    367
```

```
4  {\"freshness\":{\"numberMissedRelease\":\"448\…                    448
```

	outdated_time_ms	outdated_time_days
0	3795765000	43.932465
1	11941344000	138.210000
2	4685281000	54.227789
3	142773884000	1652.475509
4	105191360000	1217.492593

4.4 POPULARITY

4.4.1 Fetch Popularity Data from the Database

Define the Query

[10]:
```python
def get_total_popularity_records():
    query = """
    MATCH (a:Artifact)-[rel]->(r:Release)-[rel2]->(av:AddedValue)
    WHERE av.type = 'POPULARITY_1_YEAR' AND av.value IS NOT NULL
    RETURN count(*) AS total_records
    """
    result = execute_query(query)
    return result[0]['total_records']

def get_popularity_data_batch(skip, limit):
    query = f"""
    MATCH (a:Artifact)-[rel]->(r:Release)-[rel2]->(av:AddedValue)
    WHERE av.type = 'POPULARITY_1_YEAR' AND av.value IS NOT NULL
    RETURN a.id AS artifact_id, a.communityId AS community_id, av.value AS␣
 ↪popularity_value
    SKIP {skip}
    LIMIT {limit}
    """
    results = execute_query(query)
    return pd.DataFrame(results)
```

Get Total Popularity Records

[13]:
```python
total_records = get_total_popularity_records()
print(f"Total POPULARITY records: {total_records}")
```

```
Total POPULARITY records: 14459139
```

Fetch Data in Batches

[15]:
```python
batch_size = 100000  # Adjust as needed

processed_batches = []
for skip in range(0, total_records, batch_size):
```

```
    print(f"Fetching records {skip} to {min(skip + batch_size - 1,␣
↪total_records)}")
    # Fetch the batch
    batch_df = get_popularity_data_batch(skip, batch_size)

    # Process the batch (if needed)
    processed_batches.append(batch_df)
```

```
Fetching records 0 to 99999
Fetching records 100000 to 199999
Fetching records 200000 to 299999
Fetching records 300000 to 399999
Fetching records 400000 to 499999
Fetching records 500000 to 599999
Fetching records 600000 to 699999
Fetching records 700000 to 799999
Fetching records 800000 to 899999
Fetching records 900000 to 999999
Fetching records 1000000 to 1099999
Fetching records 1100000 to 1199999
Fetching records 1200000 to 1299999
Fetching records 1300000 to 1399999
Fetching records 1400000 to 1499999
Fetching records 1500000 to 1599999
Fetching records 1600000 to 1699999
Fetching records 1700000 to 1799999
Fetching records 1800000 to 1899999
Fetching records 1900000 to 1999999
Fetching records 2000000 to 2099999
Fetching records 2100000 to 2199999
Fetching records 2200000 to 2299999
Fetching records 2300000 to 2399999
Fetching records 2400000 to 2499999
Fetching records 2500000 to 2599999
Fetching records 2600000 to 2699999
Fetching records 2700000 to 2799999
Fetching records 2800000 to 2899999
Fetching records 2900000 to 2999999
Fetching records 3000000 to 3099999
Fetching records 3100000 to 3199999
Fetching records 3200000 to 3299999
Fetching records 3300000 to 3399999
Fetching records 3400000 to 3499999
Fetching records 3500000 to 3599999
Fetching records 3600000 to 3699999
Fetching records 3700000 to 3799999
Fetching records 3800000 to 3899999
Fetching records 3900000 to 3999999
```

```
Fetching records 4000000 to 4099999
Fetching records 4100000 to 4199999
Fetching records 4200000 to 4299999
Fetching records 4300000 to 4399999
Fetching records 4400000 to 4499999
Fetching records 4500000 to 4599999
Fetching records 4600000 to 4699999
Fetching records 4700000 to 4799999
Fetching records 4800000 to 4899999
Fetching records 4900000 to 4999999
Fetching records 5000000 to 5099999
Fetching records 5100000 to 5199999
Fetching records 5200000 to 5299999
Fetching records 5300000 to 5399999
Fetching records 5400000 to 5499999
Fetching records 5500000 to 5599999
Fetching records 5600000 to 5699999
Fetching records 5700000 to 5799999
Fetching records 5800000 to 5899999
Fetching records 5900000 to 5999999
Fetching records 6000000 to 6099999
Fetching records 6100000 to 6199999
Fetching records 6200000 to 6299999
Fetching records 6300000 to 6399999
Fetching records 6400000 to 6499999
Fetching records 6500000 to 6599999
Fetching records 6600000 to 6699999
Fetching records 6700000 to 6799999
Fetching records 6800000 to 6899999
Fetching records 6900000 to 6999999
Fetching records 7000000 to 7099999
Fetching records 7100000 to 7199999
Fetching records 7200000 to 7299999
Fetching records 7300000 to 7399999
Fetching records 7400000 to 7499999
Fetching records 7500000 to 7599999
Fetching records 7600000 to 7699999
Fetching records 7700000 to 7799999
Fetching records 7800000 to 7899999
Fetching records 7900000 to 7999999
Fetching records 8000000 to 8099999
Fetching records 8100000 to 8199999
Fetching records 8200000 to 8299999
Fetching records 8300000 to 8399999
Fetching records 8400000 to 8499999
Fetching records 8500000 to 8599999
Fetching records 8600000 to 8699999
Fetching records 8700000 to 8799999
```

```
Fetching records 8800000 to 8899999
Fetching records 8900000 to 8999999
Fetching records 9000000 to 9099999
Fetching records 9100000 to 9199999
Fetching records 9200000 to 9299999
Fetching records 9300000 to 9399999
Fetching records 9400000 to 9499999
Fetching records 9500000 to 9599999
Fetching records 9600000 to 9699999
Fetching records 9700000 to 9799999
Fetching records 9800000 to 9899999
Fetching records 9900000 to 9999999
Fetching records 10000000 to 10099999
Fetching records 10100000 to 10199999
Fetching records 10200000 to 10299999
Fetching records 10300000 to 10399999
Fetching records 10400000 to 10499999
Fetching records 10500000 to 10599999
Fetching records 10600000 to 10699999
Fetching records 10700000 to 10799999
Fetching records 10800000 to 10899999
Fetching records 10900000 to 10999999
Fetching records 11000000 to 11099999
Fetching records 11100000 to 11199999
Fetching records 11200000 to 11299999
Fetching records 11300000 to 11399999
Fetching records 11400000 to 11499999
Fetching records 11500000 to 11599999
Fetching records 11600000 to 11699999
Fetching records 11700000 to 11799999
Fetching records 11800000 to 11899999
Fetching records 11900000 to 11999999
Fetching records 12000000 to 12099999
Fetching records 12100000 to 12199999
Fetching records 12200000 to 12299999
Fetching records 12300000 to 12399999
Fetching records 12400000 to 12499999
Fetching records 12500000 to 12599999
Fetching records 12600000 to 12699999
Fetching records 12700000 to 12799999
Fetching records 12800000 to 12899999
Fetching records 12900000 to 12999999
Fetching records 13000000 to 13099999
Fetching records 13100000 to 13199999
Fetching records 13200000 to 13299999
Fetching records 13300000 to 13399999
Fetching records 13400000 to 13499999
Fetching records 13500000 to 13599999
```

```
Fetching records 13600000 to 13699999
Fetching records 13700000 to 13799999
Fetching records 13800000 to 13899999
Fetching records 13900000 to 13999999
Fetching records 14000000 to 14099999
Fetching records 14100000 to 14199999
Fetching records 14200000 to 14299999
Fetching records 14300000 to 14399999
Fetching records 14400000 to 14459139
```

Combine the Batches

[18]:
```
# Concatenate all processed batches into a single DataFrame
popularity_df = pd.concat(processed_batches, ignore_index=True)

# Display the first few rows
print("Popularity Data:")
print(popularity_df.head())
```

```
Popularity Data:
                                      artifact_id  community_id  \
0  com.softwaremill.sttp.client:core_sjs0.6_2.13         26959
1                          com.ibeetl:act-sample         29418
2  com.softwaremill.sttp.client:core_sjs0.6_2.13         26959
3                    com.lihaoyi:ammonite_2.12.1         26959
4                  com.yahoo.vespa:container-disc         50022

   popularity_value
0                 0
1                 0
2                 0
3                 0
4                 0
```

[22]:
```
# Export Data
popularity_df.to_csv('Popularity_data.csv', index=False)
```

4.4.2 Process Popularity Data

Ensure Correct Data Types

[26]:
```
# Convert 'popularity_value' to numeric
popularity_df['popularity_value'] = pd.
 ↪to_numeric(popularity_df['popularity_value'], errors='coerce')

# Handle missing or invalid values
popularity_df = popularity_df.dropna(subset=['popularity_value'])
```

Aggregate Popularity per Cluster

[29]:
```
# Group by community ID and calculate the average popularity per cluster
cluster_popularity = popularity_df.groupby('community_id').agg({
    'popularity_value': 'mean'
}).reset_index()

# Rename columns for clarity
cluster_popularity = cluster_popularity.rename(columns={
    'community_id': 'Community ID',
    'popularity_value': 'Average Popularity'
})

# Display the first few rows
print("Average Popularity per Cluster:")
print(cluster_popularity.head())
```

```
Average Popularity per Cluster:
   Community ID  Average Popularity
0             0                 0.0
1             1                 0.0
2             2                 0.0
3             3                 0.0
4             4                 0.0
```

4.4.3 Save the Processed Popularity Data

[32]:
```
# Save the DataFrame to a Pickle file
cluster_popularity.to_pickle('cluster_popularity_metrics.pkl')
print("Popularity metrics saved to 'cluster_popularity_metrics.pkl'")
```

```
Popularity metrics saved to 'cluster_popularity_metrics.pkl'
```

[34]:
```
# Export Data
cluster_popularity.to_csv('cluster_popularity_data.csv', index=False)
```

Load for Verification

[37]:
```
# Load the DataFrame from the Pickle file
cluster_popularity = pd.read_pickle('cluster_popularity_metrics.pkl')
print("Data loaded from 'cluster_popularity_metrics.pkl'")

# Print the first few lines of the DataFrame
print(cluster_popularity.head())
```

```
Data loaded from 'cluster_popularity_metrics.pkl'
   Community ID  Average Popularity
0             0                 0.0
1             1                 0.0
2             2                 0.0
3             3                 0.0
```

```
4              4                  0.0
```

[39]:
```
print(cluster_popularity.sort_values(by='Average Popularity', ascending=False).
 ↪head(10))
```

```
       Community ID  Average Popularity
28933         28947            6.663125
49913         49945            5.809125
18237         18244            5.729167
22331         22341            5.729167
39668         39690            5.223476
4176           4176            5.205418
35888         35907            5.116409
19687         19694            3.497355
30225         30242            3.000000
29011         29025            2.933333
```

4.5 SPEED

4.5.1 Fetch SPEED Data from the Database

Define the Query

[44]:
```
def get_total_speed_records():
    query = """
    MATCH (a:Artifact)-[rel]->(av:AddedValue)
    WHERE av.type = 'SPEED' AND av.value IS NOT NULL
    RETURN count(*) AS total_records
    """
    result = execute_query(query)
    return result[0]['total_records']

def get_speed_data_batch(skip, limit):
    query = f"""
    MATCH (a:Artifact)-[rel]->(av:AddedValue)
    WHERE av.type = 'SPEED' AND av.value IS NOT NULL
    RETURN a.id AS artifact_id, a.communityId AS community_id, av.value AS␣
 ↪speed_value
    SKIP {skip}
    LIMIT {limit}
    """
    results = execute_query(query)
    return pd.DataFrame(results)
```

Get Total Records

[47]:
```
total_records = get_total_speed_records()
print(f"Total SPEED records: {total_records}")
```

```
Total SPEED records: 658078
```

Fetch Data in Batches

[50]:
```
batch_size = 100000  # Adjust as needed

processed_batches = []
for skip in range(0, total_records, batch_size):
    print(f"Fetching records {skip} to {min(skip + batch_size - 1,␣
 ↪total_records)}")
    # Fetch the batch
    batch_df = get_speed_data_batch(skip, batch_size)

    # Process the batch (if needed)
    processed_batches.append(batch_df)
```

```
Fetching records 0 to 99999
Fetching records 100000 to 199999
Fetching records 200000 to 299999
Fetching records 300000 to 399999
Fetching records 400000 to 499999
Fetching records 500000 to 599999
Fetching records 600000 to 658078
```

Combine Batches

[53]:
```
# Concatenate all processed batches into a single DataFrame
speed_df = pd.concat(processed_batches, ignore_index=True)

# Display the first few rows
print("SPEED Data:")
print(speed_df.head())
```

```
SPEED Data:
                                        artifact_id  community_id  \
0              com.splendo.kaluga:alerts-androidlib         28940
1  org.wso2.carbon.identity.framework:org.wso2.ca…         30270
2  org.apache.camel.quarkus:camel-quarkus-kotlin-…         43407
3    io.projectreactor:reactor-scala-extensions_2.11         48568
4                  com.lihaoyi:ammonite-shell_2.10.5         26959

            speed_value
0  0.014109347442680775
1    0.7062795408507765
2  0.038056206088992975
3   0.01782178217821782
4   0.44660194174757284
```

4.5.2 Process the SPEED Data

[56]:
```
# Export Data
speed_df.to_csv('Speed_data.csv', index = False)
```

Ensure Correct Data Types

[59]:
```
def parse_speed(value):
    try:
        return float(value)
    except (ValueError, TypeError):
        return None

speed_df['avg_release_interval_days'] = speed_df['speed_value'].
 ↪apply(parse_speed)

# Drop rows with missing 'avg_release_interval_days'
speed_df = speed_df.dropna(subset=['avg_release_interval_days'])
```

Aggregate SPEED per Cluster

[62]:
```
# Group by community ID and calculate the average speed per cluster
cluster_speed = speed_df.groupby('community_id').agg({
    'avg_release_interval_days': 'mean'
}).reset_index()

# Rename columns for clarity
cluster_speed = cluster_speed.rename(columns={
    'community_id': 'Community ID',
    'avg_release_interval_days': 'Average Speed (Days)'
})

# Display the first few rows
print("Average SPEED per Cluster:")
print(cluster_speed.head())
```

```
Average SPEED per Cluster:
   Community ID  Average Speed (Days)
0             0              0.003755
1             1              0.001709
2             2              0.000000
3             3              0.000000
4             4              0.000000
```

4.5.3 Save Processed SPEED Data

[65]:
```
# Save the DataFrame to a Pickle file
cluster_speed.to_pickle('cluster_speed_metrics.pkl')
print("SPEED metrics saved to 'cluster_speed_metrics.pkl'")
```

```
SPEED metrics saved to 'cluster_speed_metrics.pkl'
```

[67]:
```
# Export Data
cluster_speed.to_csv('cluster_speed_data.csv', index=False)
```

Load For Verification

[70]:
```
# Load the DataFrame from the Pickle file
cluster_speed = pd.read_pickle('cluster_speed_metrics.pkl')
print("Data loaded from 'cluster_speed_metrics.pkl'")

# Print the first few lines of the DataFrame
print(cluster_speed.head())
```

```
Data loaded from 'cluster_speed_metrics.pkl'
   Community ID  Average Speed (Days)
0             0              0.003755
1             1              0.001709
2             2              0.000000
3             3              0.000000
4             4              0.000000
```

[72]:
```
print(cluster_speed.sort_values(by='Average Speed (Days)', ascending=False).
  ↪head(10))
```

```
       Community ID  Average Speed (Days)
51949         51981                 140.0
9695           9697                  27.0
12256         12259                  22.0
4279           4279                  14.0
60768         60804                  14.0
31399         31416                  13.0
16669         16673                  12.0
46146         46171                  12.0
3865           3865                  11.0
18612         18619                  10.0
```

5 Combine Metrics

5.0.1 Load Individual Metrics DataFrames

[136]:
```
# Load individual metrics DataFrames
cluster_cve_metrics= pd.read_pickle('cluster_cve_metrics.pkl')
print("Loaded 'cluster_cve_metrics.pkl'")

# cluster_update_metrics = pd.read_pickle('cluster_update_metrics.pkl')
# print("Loaded 'cluster_update_metrics.pkl'")

cluster_freshness_metrics = pd.read_pickle('cluster_freshness_metrics.pkl')
print("Loaded 'cluster_freshness_metrics.pkl'")

cluster_popularity_metrics = pd.read_pickle('cluster_popularity_metrics.pkl')
print("Loaded 'cluster_popularity_metrics.pkl'")

cluster_speed_metrics = pd.read_pickle('cluster_speed_metrics.pkl')
print("Loaded 'cluster_speed_metrics.pkl'")
```

```
Loaded 'cluster_cve_metrics.pkl'
Loaded 'cluster_freshness_metrics.pkl'
Loaded 'cluster_popularity_metrics.pkl'
Loaded 'cluster_speed_metrics.pkl'
```

Verify

[140]:
```
print(cluster_cve_metrics.head(1))
# print(cluster_update_metrics.head(1))
print(cluster_freshness_metrics.head(1))
print(cluster_popularity_metrics.head(1))
print(cluster_speed_metrics.head(1))
```

```
   Community ID  Total CVEs  \
0           436         121

                                              CVE List  Num_Releases  \
0  [CVE-2022-25873, CVE-2022-25873, CVE-2022-2587…           121

   Average Severity Score  Normalized Total CVEs
0                     2.0                    1.0
   Community ID  Average Freshness (Days)
0             0                 31.300386
   Community ID  Average Popularity
0             0                  0.0
   Community ID  Average Speed (Days)
0             0              0.003755
```

5.0.2 Merge DataFrames into a Single DataFrame

Merge on the 'Community ID' Column

[146]:
```
# Start with the CVE metrics DataFrame
cluster_risk_metrics = cluster_cve_metrics.copy()

# # Merge with update metrics
# cluster_risk_metrics = pd.merge(
#     cluster_risk_metrics,
#     cluster_update_metrics,
#     on='Community ID',
#     how='outer'
# )

# Merge with freshness metrics
cluster_risk_metrics = pd.merge(
    cluster_risk_metrics,
    cluster_freshness_metrics,
    on='Community ID',
    how='outer'
)

# Merge with popularity metrics
cluster_risk_metrics = pd.merge(
    cluster_risk_metrics,
    cluster_popularity_metrics,
    on='Community ID',
    how='outer'
)

# Merge with speed metrics
cluster_risk_metrics = pd.merge(
    cluster_risk_metrics,
    cluster_speed_metrics,
    on='Community ID',
    how='outer'
)
```

5.0.3 Handle Missing Data

[149]:
```
# List of metrics
metrics = [
    'Normalized Total CVEs',
    'Average Severity Score',
    # 'Average Days Since Last Update',
    'Average Freshness (Days)',
    'Average Popularity',
```

```
    'Average Speed (Days)'
]

# Fill missing values with appropriate defaults
# For metrics where missing implies zero risk, fill with 0
cluster_risk_metrics[metrics] = cluster_risk_metrics[metrics].fillna(0)
```

5.0.4 Save Metrics

[152]:
```
# Save the DataFrame to a Pickle file
cluster_risk_metrics.to_pickle('cluster_risk_metrics.pkl')
print("Complete Risk metrics saved to 'cluster_risk_metrics.pkl'")
```

```
Complete Risk metrics saved to 'cluster_risk_metrics.pkl'
```

[154]:
```
# Export Data
cluster_risk_metrics.to_csv('cluster_risk_metrics.csv', index=False)
```

Verify Data

[159]:
```
cluster_risk_metrics = pd.read_pickle('cluster_risk_metrics.pkl')
print("Data loaded from 'cluster_risk_metrics.pkl'")

# Print the first few lines of the DataFrame
cluster_risk_metrics.head()
```

```
Data loaded from 'cluster_risk_metrics.pkl'
```

[159]:

	Community ID	Total CVEs	CVE List	Num_Releases	Average Severity Score	Normalized Total CVEs	Average Freshness (Days)	Average Popularity	Average Speed (Days)
0	0	NaN	NaN	NaN	0.0	0.0	31.300386	0.0	0.003755
1	1	NaN	NaN	NaN	0.0	0.0	195.021071	0.0	0.001709
2	2	NaN	NaN	NaN	0.0	0.0	0.000000	0.0	0.000000
3	3	NaN	NaN	NaN	0.0	0.0	0.000000	0.0	0.000000
4	4	NaN	NaN	NaN	0.0	0.0	0.000000	0.0	0.000000

6 Principal Component Analysis (PCA)

Objective: Use PCA to derive weights for your risk metrics and compute a Composite Risk Score.
Metrics to Include:
* Normalized Total CVEs * Average Severity Score * Average Days Since Last Update * Average Freshness (Days) * Average Popularity * Average Speed (Days)

6.1 Load and Prepare the Data

Load merged DataFrame

[161]:
```
# Load the final merged DataFrame
cluster_risk_metrics = pd.read_pickle('cluster_risk_metrics.pkl')
print("Data loaded from 'cluster_risk_metrics.pkl'")
cluster_risk_metrics.head()
```

```
Data loaded from 'cluster_risk_metrics.pkl'
```

[161]:

	Community ID	Total CVEs	CVE List	Num_Releases	Average Severity Score	\
0	0	NaN	NaN	NaN	0.0	
1	1	NaN	NaN	NaN	0.0	
2	2	NaN	NaN	NaN	0.0	
3	3	NaN	NaN	NaN	0.0	
4	4	NaN	NaN	NaN	0.0	

	Normalized Total CVEs	Average Freshness (Days)	Average Popularity	\
0	0.0	31.300386	0.0	
1	0.0	195.021071	0.0	
2	0.0	0.000000	0.0	
3	0.0	0.000000	0.0	
4	0.0	0.000000	0.0	

	Average Speed (Days)
0	0.003755
1	0.001709
2	0.000000
3	0.000000
4	0.000000

6.1.1 Standardize the Metrics Using StandardScaler

[163]:
```
# Remove CVE List from Data, This is not needed for PCA
cluster_risk_metrics = cluster_risk_metrics.drop('CVE List', axis=1)
cluster_risk_metrics.head()
```

[163]:

	Community ID	Total CVEs	Num_Releases	Average Severity Score	\
0	0	NaN	NaN	0.0	
1	1	NaN	NaN	0.0	

```
2             2         NaN           NaN                     0.0
3             3         NaN           NaN                     0.0
4             4         NaN           NaN                     0.0

   Normalized Total CVEs  Average Freshness (Days)  Average Popularity  \
0                    0.0                 31.300386                 0.0
1                    0.0                195.021071                 0.0
2                    0.0                  0.000000                 0.0
3                    0.0                  0.000000                 0.0
4                    0.0                  0.000000                 0.0

   Average Speed (Days)
0              0.003755
1              0.001709
2              0.000000
3              0.000000
4              0.000000
```

6.1.2 Perform PCA

[166]:
```
# Define metrics to include in PCA
metrics = [
    'Normalized Total CVEs',
    'Average Severity Score',
    # 'Average Days Since Last Update',
    'Average Freshness (Days)',
    'Average Popularity',
    'Average Speed (Days)'
]

# Ensure metrics are numeric and handle missing values
cluster_risk_metrics[metrics] = cluster_risk_metrics[metrics].apply(pd.
 ↪to_numeric, errors='coerce')
cluster_risk_metrics[metrics] = cluster_risk_metrics[metrics].fillna(0)

# Standardize the metrics
scaler = StandardScaler()
X = scaler.fit_transform(cluster_risk_metrics[metrics])

# Convert back to DataFrame for easier handling
X = pd.DataFrame(X, columns=metrics)

# Initialize and fit PCA
pca = PCA(n_components=len(metrics))
pca.fit(X)

# Extract the loadings for PC1
```

```
pc1_loadings = pd.Series(pca.components_[0], index=metrics)

# Take the absolute values of loadings as weights
weights = pc1_loadings.abs()

# Normalize the weights to sum to 1
weights = weights / weights.sum()

print("\nWeights derived from PC1 loadings:")
print(weights)
```

```
Weights derived from PC1 loadings:
Normalized Total CVEs       0.376538
Average Severity Score      0.383546
Average Freshness (Days)    0.068776
Average Popularity          0.165592
Average Speed (Days)        0.005547
dtype: float64
```

7 Composite Risk Score

7.1 Calculate Weights

7.1.1 Apply PCA Weights to Calculate the Composite Risk Score

Calculate Weighted Metrics

[172]:
```
# Multiply each standardized metric by its weight
weighted_metrics = X.multiply(weights, axis=1)
```

Calculate the Composite Risk Score

[176]:
```
# Sum the weighted metrics to get the risk score
cluster_risk_metrics_weighted = cluster_risk_metrics
cluster_risk_metrics_weighted['Risk Score'] = weighted_metrics.sum(axis=1)

# Export for viewing
cluster_risk_metrics_weighted.to_csv('temp1.csv', index=False)

# Display
cluster_risk_metrics_weighted.head()
```

[176]:

	Community ID	Total CVEs	Num_Releases	Average Severity Score	\
0	0	NaN	NaN	0.0	
1	1	NaN	NaN	0.0	
2	2	NaN	NaN	0.0	
3	3	NaN	NaN	0.0	
4	4	NaN	NaN	0.0	

```
   Normalized Total CVEs  Average Freshness (Days)  Average Popularity  \
0                    0.0                 31.300386                 0.0
1                    0.0                195.021071                 0.0
2                    0.0                  0.000000                 0.0
3                    0.0                  0.000000                 0.0
4                    0.0                  0.000000                 0.0

   Average Speed (Days)  Risk Score
0              0.003755   -0.072326
1              0.001709   -0.047453
2              0.000000   -0.077116
3              0.000000   -0.077116
4              0.000000   -0.077116
```

7.1.2 Rescale the Risk Scores

Make risk scores more intepreprable (between 0 and 1)

[103]:
```
# # Rescale Risk Scores to range [0, 1]
# scaler = MinMaxScaler()
# cluster_risk_metrics_weighted['Risk Score'] = scaler.
 ↪fit_transform(cluster_risk_metrics_weighted[['Risk Score']])
```

[178]:
```
# Export Data
cluster_risk_metrics_weighted.to_pickle('cluster_risk_metrics_weighted.pkl')
```

7.2 Analyze and Interpret the Results

Sort and Display Top High-Risk Clusters

[5]:
```
# Load Data and Display
cluster_risk_metrics_weighted = pd.read_pickle('cluster_risk_metrics_weighted.
 ↪pkl')
cluster_risk_metrics_weighted.head()
```

[5]:
```
   Community ID  Total CVEs  Num_Releases  Average Severity Score  \
0             0         NaN           NaN                     0.0
1             1         NaN           NaN                     0.0
2             2         NaN           NaN                     0.0
3             3         NaN           NaN                     0.0
4             4         NaN           NaN                     0.0

   Normalized Total CVEs  Average Freshness (Days)  Average Popularity  \
0                    0.0                 31.300386                 0.0
1                    0.0                195.021071                 0.0
2                    0.0                  0.000000                 0.0
3                    0.0                  0.000000                 0.0
```

```
4                    0.0                    0.000000                    0.0
```

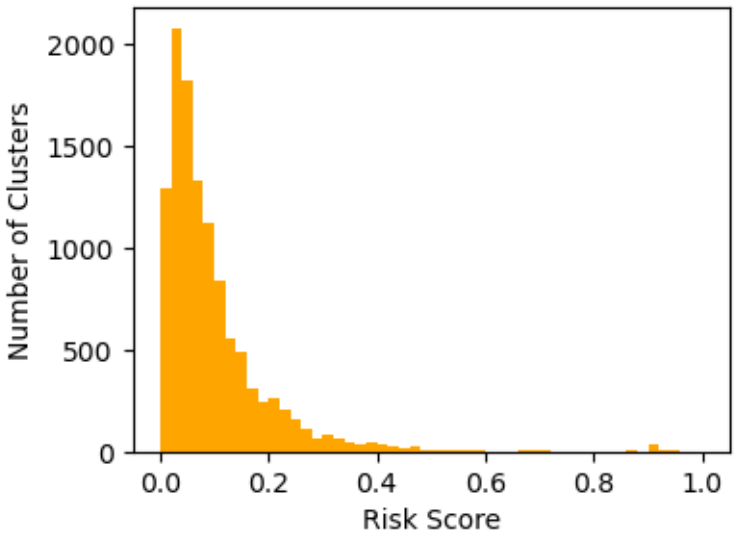
```
   Average Speed (Days)  Risk Score
0              0.003755   -0.072326
1              0.001709   -0.047453
2              0.000000   -0.077116
3              0.000000   -0.077116
4              0.000000   -0.077116
```

Visualize the Distribution of Risk Scores

[22]:
```
# Plot the distribution of Risk Scores

# Filter to focus on risk clusters (Risk Score > 0.5)
risk_clusters =␣
 ↪cluster_risk_metrics_weighted[cluster_risk_metrics_weighted['Risk Score']>0.
 ↪01]

plt.figure(figsize=(4, 3))
plt.hist(risk_clusters['Risk Score'], bins=50, range=(0, 1), color='orange')
plt.xlabel('Risk Score')
plt.ylabel('Number of Clusters')
# plt.title('Distribution of Risk Scores for High-Risk Clusters')
plt.savefig('cluster_risk_distribution.pdf', bbox_inches='tight')
plt.show()
```



Save DataFrame

[188]:
```
# Save the updated DataFrame
cluster_risk_metrics_weighted.to_pickle('cluster_risk_metrics_pca.pkl')
print("Updated data saved to 'cluster_risk_metrics_pca.pkl'")
```

```
Updated data saved to 'cluster_risk_metrics_pca.pkl'
```

[190]:
```
# Load Cluster Risk Metrics into DataFrame
cluster_risk_metrics_weighted = pd.read_pickle('cluster_risk_metrics_pca.pkl')
cluster_risk_metrics_weighted.head()
```

[190]:

	Community ID	Total CVEs	Num_Releases	Average Severity Score	Normalized Total CVEs	Average Freshness (Days)	Average Popularity	Average Speed (Days)	Risk Score
0	0	NaN	NaN	0.0	0.0	31.300386	0.0	0.003755	-0.072326
1	1	NaN	NaN	0.0	0.0	195.021071	0.0	0.001709	-0.047453
2	2	NaN	NaN	0.0	0.0	0.000000	0.0	0.000000	-0.077116
3	3	NaN	NaN	0.0	0.0	0.000000	0.0	0.000000	-0.077116
4	4	NaN	NaN	0.0	0.0	0.000000	0.0	0.000000	-0.077116

Get Cluster Size from Database then Save File for Future Work

[50]:
```
# Query to get cluster sizes
cluster_sizes_query = f"""
MATCH (n)
WHERE n.communityId IS NOT NULL
RETURN n.communityId AS communityId, count(*) AS size
ORDER BY size DESC
"""
cluster_sizes = execute_query(cluster_sizes_query)
cluster_sizes_df = pd.DataFrame(cluster_sizes)
cluster_sizes_df.rename(columns={'communityId':'Community ID'}, inplace=True)

# Display
print("Top 10 Clusters by Size:")
print(cluster_sizes_df.head())
```

```
# Save DataFrame to a pickle file
cluster_sizes_df.to_pickle('cluster_sizes.pkl')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[50], line 8
      1 # Query to get cluster sizes
      2 cluster_sizes_query = f"""
      3 MATCH (n)
      4 WHERE n.communityId IS NOT NULL
      5 RETURN n.communityId AS communityId, count(*) AS size
      6 ORDER BY size DESC
      7 """
----> 8 cluster_sizes = execute_query(cluster_sizes_query)
      9 cluster_sizes_df = pd.DataFrame(cluster_sizes)
     10 cluster_sizes_df.rename(columns={'communityId':'Community ID'},␣
 ↪inplace=True)

NameError: name 'execute_query' is not defined
```

Merge Cluster Size with DataFrame

[192]:
```
# Load Previous Files
cluster_total_metrics_temp = pd.read_pickle('cluster_risk_metrics_pca.pkl')
cluster_sizes_temp = pd.read_pickle('cluster_sizes.pkl')

# Merge Files and Save
cluster_total_metrics1 =  cluster_total_metrics_temp.merge(cluster_sizes_temp,␣
 ↪on='Community ID', how='left')
cluster_total_metrics1.to_pickle('cluster_total_metrics1.pkl')
cluster_total_metrics1.head()
```

[192]:

	Community ID	Total CVEs	Num_Releases	Average Severity Score	Normalized Total CVEs	Average Freshness (Days)	Average Popularity
0	0	NaN	NaN	0.0	0.0	31.300386	0.0
1	1	NaN	NaN	0.0	0.0	195.021071	0.0
2	2	NaN	NaN	0.0	0.0	0.000000	0.0
3	3	NaN	NaN	0.0	0.0	0.000000	0.0
4	4	NaN	NaN	0.0	0.0	0.000000	0.0

```
   Average Speed (Days)  Risk Score  size
0              0.003755   -0.072326    20
1              0.001709   -0.047453     5
2              0.000000   -0.077116    13
3              0.000000   -0.077116     3
4              0.000000   -0.077116     3
```

Inter Cluster Connections

[469]:
```
# Query to analyze inter-cluster relationships
inter_cluster_query = f"""
MATCH (a)-[r]->(b)
WHERE a.communityId <> b.communityId
RETURN a.communityId AS sourceCluster, b.communityId AS targetCluster, count(*)␣
 ↪AS connections
ORDER BY connections DESC
"""
inter_cluster_connections = execute_query(inter_cluster_query)
inter_cluster_df = pd.DataFrame(inter_cluster_connections)
inter_cluster_df.rename(columns={'sourceCluster':'Community ID'}, inplace=True)
inter_cluster_df.rename(columns={'targetCluster':'Target Cluster'},␣
 ↪inplace=True)

# Display
print("Top 10 Inter-Cluster Connections:")
print(inter_cluster_df.head())

# Save For Future Use
inter_cluster_df.to_pickle('inter_cluster_df.pkl')
```

```
Top 10 Inter-Cluster Connections:
   Community ID  Target Cluster  connections
0         48568           29418      1791786
1         29418           48568      1447774
2         34416           29418       837511
3         34416           28947       711388
4         29418           28947       699461
```

Merge Cluster Connections with DataFrame

[194]:
```
# Load Previous Files
cluster_total_metrics_temp = pd.read_pickle('cluster_total_metrics1.pkl')
cluster_inter_connections = pd.read_pickle('inter_cluster_df.pkl')

# Merge and Save
cluster_total_metrics2 =  cluster_total_metrics_temp.
 ↪merge(cluster_inter_connections, on='Community ID', how='left')
cluster_total_metrics2.to_pickle('cluster_total_metrics2.pkl')
```

```
cluster_total_metrics2.head()
```

[194]:

	Community ID	Total CVEs	Num_Releases	Average Severity Score	Normalized Total CVEs	Average Freshness (Days)	Average Popularity	Average Speed (Days)	Risk Score	size	Target Cluster	connections
0	0	NaN	NaN	0.0	0.0	31.300386	0.0	0.003755	-0.072326	20	NaN	NaN
1	1	NaN	NaN	0.0	0.0	195.021071	0.0	0.001709	-0.047453	5	NaN	NaN
2	2	NaN	NaN	0.0	0.0	0.000000	0.0	0.000000	-0.077116	13	NaN	NaN
3	3	NaN	NaN	0.0	0.0	0.000000	0.0	0.000000	-0.077116	3	NaN	NaN
4	4	NaN	NaN	0.0	0.0	0.000000	0.0	0.000000	-0.077116	3	NaN	NaN

Export for saving and viewing

[130]:
```
# Load Pickle File
cluster_total_metrics2 = pd.read_pickle('cluster_total_metrics2.pkl')
```

[198]:
```
# Sort the DataFrame by the 'Risk Score' column in descending order
cluster_metrics_sorted = cluster_metrics_final.sort_values(by='Risk Score',␣
 ↪ascending=False)

# Display the top rows of the sorted DataFrame
cluster_metrics_sorted.head()
```

[198]:

	Community ID	Total CVEs	Num_Releases	Average Severity Score	Normalized Total CVEs	Average Freshness (Days)	Average Popularity
39352	37666	2470.0	163.0	2.491498	15.153374	2028.301837	0.000000
45374	43639	1482.0	132.0	2.941970	11.227273	712.004092	0.000000
43632	41923	259.0	27.0	2.602317	9.592593	2044.736117	0.000000
11661	11268	71.0	8.0	2.000000	8.875000	914.076289	0.000000
30048	28947	827.0	292.0	2.585248	2.832192	878.906223	6.663125

	Average Speed (Days)	Risk Score	size	Target Cluster	connections
39352	0.045548	1.000000	179	NaN	NaN
45374	0.066108	0.802396	143	NaN	NaN
43632	0.010076	0.694845	29	NaN	NaN
11661	0.008397	0.616346	12	NaN	NaN
30048	0.100989	0.559408	561882	35373.0	24795.0

[134]:
```
cluster_metrics_final.to_csv('cluster_metrics_final.csv', index=False)
```

[ ]:
```

```