FinalNeo4j

May 26, 2023

0.1 4.0 PREPARING SYSTEM

[2]:
```
!pip install neo4j
```

```
Collecting neo4j
  Downloading neo4j-5.9.0.tar.gz (188 kB)
                         188.5/188.5

kB 2.2 MB/s eta 0:00:0000:0100:01
  Installing build dependencies … done
  Getting requirements to build wheel … done
  Installing backend dependencies … done
  Preparing metadata (pyproject.toml) … done
Requirement already satisfied: pytz in
/Users/ajaydhamanda/opt/anaconda3/lib/python3.9/site-packages (from neo4j)
(2022.1)
Building wheels for collected packages: neo4j
  Building wheel for neo4j (pyproject.toml) … done
  Created wheel for neo4j: filename=neo4j-5.9.0-py3-none-any.whl
size=259467
sha256=dfdfbe3db784d379b723c901c20a4d795c2edf77fee24496c18cf96ca9536977
  Stored in directory: /Users/ajaydhamanda/Library/Caches/pip/wheels/c8/18/02/4e
34f0d2b0f16a2ff664826f45421754af249d24522f0c987d
Successfully built neo4j
Installing collected packages: neo4j
Successfully installed neo4j-5.9.0
```

0.2 4.1 DATA EXPLORATION

[8]:
```python
from neo4j import GraphDatabase

# Establish a connection to the remote Neo4j database
uri = "neo4j+s://82d7ec45.databases.neo4j.io"
username = "neo4j"
password = "LFKIPcQfywuwp4Xg8-_TLpJwTxU-h10Ftb7xBYgmZ1o"

driver = GraphDatabase.driver(uri, auth=(username, password))

# Define and execute the query
```

```
query = "MATCH (n) RETURN n LIMIT 20"  # Add LIMIT 20 to limit the number of␣
 ↪records to 50

with driver.session() as session:
    result = session.run(query)
    for record in result:
        print(record['n'])

# Close the driverb
driver.close()
```

```
<Node element_id='4:6a35191c-63f6-4ee4-9f4f-0c865084b593:0'
labels=frozenset({'Movie'}) properties={'tagline': 'Welcome to the Real World',
'title': 'The Matrix', 'released': 1999}>
<Node element_id='4:6a35191c-63f6-4ee4-9f4f-0c865084b593:1'
labels=frozenset({'Person'}) properties={'born': 1964, 'name': 'Keanu Reeves'}>
<Node element_id='4:6a35191c-63f6-4ee4-9f4f-0c865084b593:2'
labels=frozenset({'Person'}) properties={'born': 1967, 'name': 'Carrie-Anne
Moss'}>
<Node element_id='4:6a35191c-63f6-4ee4-9f4f-0c865084b593:3'
labels=frozenset({'Person'}) properties={'born': 1961, 'name': 'Laurence
Fishburne'}>
<Node element_id='4:6a35191c-63f6-4ee4-9f4f-0c865084b593:4'
labels=frozenset({'Person'}) properties={'born': 1960, 'name': 'Hugo Weaving'}>
<Node element_id='4:6a35191c-63f6-4ee4-9f4f-0c865084b593:5'
labels=frozenset({'Person'}) properties={'born': 1967, 'name': 'Lilly
Wachowski'}>
<Node element_id='4:6a35191c-63f6-4ee4-9f4f-0c865084b593:6'
labels=frozenset({'Person'}) properties={'born': 1965, 'name': 'Lana
Wachowski'}>
<Node element_id='4:6a35191c-63f6-4ee4-9f4f-0c865084b593:7'
labels=frozenset({'Person'}) properties={'born': 1952, 'name': 'Joel Silver'}>
<Node element_id='4:6a35191c-63f6-4ee4-9f4f-0c865084b593:8'
labels=frozenset({'Person'}) properties={'born': 1978, 'name': 'Emil Eifrem'}>
<Node element_id='4:6a35191c-63f6-4ee4-9f4f-0c865084b593:9'
labels=frozenset({'Movie'}) properties={'tagline': 'Free your mind', 'title':
'The Matrix Reloaded', 'released': 2003}>
<Node element_id='4:6a35191c-63f6-4ee4-9f4f-0c865084b593:10'
labels=frozenset({'Movie'}) properties={'tagline': 'Everything that has a
beginning has an end', 'title': 'The Matrix Revolutions', 'released': 2003}>
<Node element_id='4:6a35191c-63f6-4ee4-9f4f-0c865084b593:11'
labels=frozenset({'Movie'}) properties={'tagline': 'Evil has its winning ways',
'title': "The Devil's Advocate", 'released': 1997}>
<Node element_id='4:6a35191c-63f6-4ee4-9f4f-0c865084b593:12'
labels=frozenset({'Person'}) properties={'born': 1975, 'name': 'Charlize
Theron'}>
<Node element_id='4:6a35191c-63f6-4ee4-9f4f-0c865084b593:13'
labels=frozenset({'Person'}) properties={'born': 1940, 'name': 'Al Pacino'}>
```

```
<Node element_id='4:6a35191c-63f6-4ee4-9f4f-0c865084b593:14'
labels=frozenset({'Person'}) properties={'born': 1944, 'name': 'Taylor
Hackford'}>
<Node element_id='4:6a35191c-63f6-4ee4-9f4f-0c865084b593:15'
labels=frozenset({'Movie'}) properties={'tagline': "In the heart of the nation's
capital, in a courthouse of the U.S. government, one man will stop at nothing to
keep his honor, and one will stop at nothing to find the truth.", 'title': 'A
Few Good Men', 'released': 1992}>
<Node element_id='4:6a35191c-63f6-4ee4-9f4f-0c865084b593:16'
labels=frozenset({'Person'}) properties={'born': 1962, 'name': 'Tom Cruise'}>
<Node element_id='4:6a35191c-63f6-4ee4-9f4f-0c865084b593:17'
labels=frozenset({'Person'}) properties={'born': 1937, 'name': 'Jack
Nicholson'}>
<Node element_id='4:6a35191c-63f6-4ee4-9f4f-0c865084b593:18'
labels=frozenset({'Person'}) properties={'born': 1962, 'name': 'Demi Moore'}>
<Node element_id='4:6a35191c-63f6-4ee4-9f4f-0c865084b593:19'
labels=frozenset({'Person'}) properties={'born': 1958, 'name': 'Kevin Bacon'}>
```

[9]:
```
#Let's execute some queries to see what kind of data we are dealing with
from neo4j import GraphDatabase

# Establish a connection to the remote Neo4j database
uri = "neo4j+s://82d7ec45.databases.neo4j.io"
username = "neo4j"
password = "LFKIPcQfywuwp4Xg8-_TLpJwTxU-h10Ftb7xBYgmZ1o"

driver = GraphDatabase.driver(uri, auth=(username, password))

# Define and execute the queries
queries = [
    (
      """MATCH (m:Movie)
       RETURN m.title AS movie
       ORDER BY m.title ASC
       LIMIT 5""",
    "Query 1: List of 5 movies sorted alphabetically"
    ),
    (
        """MATCH (a:Person)-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(d:Person)
           RETURN a.name AS actor, d.name AS director
           LIMIT 5""",
        "Query 2: 5 records showing actors and the directors they've worked␣
↪with"
    ),
    (
        """MATCH (w:Person)-[:WROTE]->(m:Movie)<-[:ACTED_IN]-(a:Person)
           RETURN m.title AS movie, w.name AS writer, a.name AS actor
```

```
            LIMIT 5""",
        "Query 3: 5 records showing combinations of writers, movies, and actors"
    ),
    (
        """MATCH (m:Movie)<-[r:REVIEWED]-(p:Person)
       RETURN m.title AS movie, count(r) AS review_count
       ORDER BY review_count DESC
       LIMIT 5""",
    "Query 4: List of 5 movies with the most reviews"
),
(
    """MATCH (m:Movie)<-[r:REVIEWED]-(p:Person)
       RETURN m.title AS movie, count(r) AS review_count
       ORDER BY review_count ASC
       LIMIT 5""",
    "Query 4: List of 5 movies with the least reviews"
    ),
    (
        """MATCH (a:Person)<-[:FOLLOWS]-(f:Person)
           RETURN a.name AS actor, count(f) AS fans_count
           ORDER BY fans_count DESC
           LIMIT 5""",
        "Query 5: List of 5 actors with the most fans"
    )
]

with driver.session() as session:
    for index, (query, comment) in enumerate(queries):
        result = session.run(query)
        print(comment)
        for record in result:
            print(record)
        print()

# Close the driver
driver.close()
```

```
Query 1: List of 5 movies sorted alphabetically
<Record movie='A Few Good Men'>
<Record movie='A League of Their Own'>
<Record movie='Apollo 13'>
<Record movie='As Good as It Gets'>
<Record movie='Bicentennial Man'>

Query 2: 5 records showing actors and the directors they've worked with
<Record actor='Emil Eifrem' director='Lana Wachowski'>
<Record actor='Hugo Weaving' director='Lana Wachowski'>
<Record actor='Laurence Fishburne' director='Lana Wachowski'>
```

```
<Record actor='Carrie-Anne Moss' director='Lana Wachowski'>
<Record actor='Keanu Reeves' director='Lana Wachowski'>

Query 3: 5 records showing combinations of writers, movies, and actors
<Record movie='A Few Good Men' writer='Aaron Sorkin' actor='James Marshall'>
<Record movie='A Few Good Men' writer='Aaron Sorkin' actor='Kevin Pollak'>
<Record movie='A Few Good Men' writer='Aaron Sorkin' actor='J.T. Walsh'>
<Record movie='A Few Good Men' writer='Aaron Sorkin' actor='Aaron Sorkin'>
<Record movie='A Few Good Men' writer='Aaron Sorkin' actor='Cuba Gooding Jr.'>

Query 4: List of 5 movies with the most reviews
<Record movie='The Replacements' review_count=3>
<Record movie='The Da Vinci Code' review_count=2>
<Record movie='The Birdcage' review_count=1>
<Record movie='Cloud Atlas' review_count=1>
<Record movie='Unforgiven' review_count=1>

Query 4: List of 5 movies with the least reviews
<Record movie='Jerry Maguire' review_count=1>
<Record movie='The Birdcage' review_count=1>
<Record movie='Unforgiven' review_count=1>
<Record movie='Cloud Atlas' review_count=1>
<Record movie='The Da Vinci Code' review_count=2>

Query 5: List of 5 actors with the most fans
<Record actor='Jessica Thompson' fans_count=2>
<Record actor='Angela Scope' fans_count=1>
```

0.3 4.2 BUSINESS OBJECTIVES

1. Improve audience engagement: The business aims to increase audience engagement by providing content that is more aligned with audience preferences. By doing so, they can increase the number of views and retain their viewership.
2. Expand the network of professionals: The business is looking to expand its network by collaborating with more directors, producers, writers, and actors. This can lead to the production of a wider variety of movies and can potentially attract a larger audience.

0.4 4.3 DECISIONS

1. Find out the actors that have the most collaborations with directors
2. Movies with highest reviews

0.5 4.4 BUSINESS QUESTIONS

Which actors have the most extensive collaborations with directors?

1. This could help us understand which actors are more versatile and adaptive to different directing styles. This insight could be useful in deciding which actors to consider for future

projects.

2. Which movies have the highest and lowest number of reviews? Understanding which movies are generating a lot of discussion can give us an insight into the movies that are currently trending. Conversely, knowing which movies have the least number of reviews can indicate the movies that might need more promotion or are not resonating well with the audience.

0.6 4.5 DATABASE QUERIES

[10]:
```
from neo4j import GraphDatabase

# Establish a connection to the remote Neo4j database
uri = "neo4j+s://82d7ec45.databases.neo4j.io"
username = "neo4j"
password = "LFKIPcQfywuwp4Xg8-_TLpJwTxU-h1OFtb7xBYgmZ1o"

driver = GraphDatabase.driver(uri, auth=(username, password))

# Define and execute the queries
queries = [
    (
        # Query 1a: 5 actors with the highest number of collaborations
        """MATCH (a:Person)-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(d:Person)
           RETURN a.name AS actor, count(d) AS director_count
           ORDER BY director_count DESC
           LIMIT 5""",
        "Query 1a: 5 actors with the highest number of collaborations"
    ),
    (
        # Query 1b: 5 actors who have collaborated with the most unique␣
 ↪directors
        """MATCH (a:Person)-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(d:Person)
           RETURN a.name AS actor, count(DISTINCT d) AS unique_director_count
           ORDER BY unique_director_count DESC
           LIMIT 5""",
        "Query 1b: 5 actors who have collaborated with the most unique␣
 ↪directors"
    ),
    (
        # Query 2a: 5 movies with highest average review
        """MATCH (m:Movie)<-[r:REVIEWED]-(p:Person)
           RETURN m.title AS movie, avg(r.rating) AS avg_rating
           ORDER BY avg_rating DESC
           LIMIT 5""",
        "Query 2a: 5 movies with highest average review"
    ),
    (
        # Query 2b: 5 movies with the highest number of unique reviewers
        """MATCH (m:Movie)<-[r:REVIEWED]-(p:Person)
           RETURN m.title AS movie, count(DISTINCT p) AS unique_reviewer_count
           ORDER BY unique_reviewer_count DESC
           LIMIT 5""",
        "Query 2b: 5 movies with the highest number of unique reviewers"
    )
]

with driver.session() as session:
    for index, (query, comment) in enumerate(queries):
        result = session.run(query)
        print(comment)
        for record in result:
            print(record)
        print()

# Close the driver
driver.close()
```

```
Query 1a: 5 actors with the highest number of collaborations
<Record actor='Tom Hanks' director_count=14>
<Record actor='Keanu Reeves' director_count=10>
<Record actor='Hugo Weaving' director_count=10>
<Record actor='Laurence Fishburne' director_count=6>
<Record actor='Carrie-Anne Moss' director_count=6>

Query 1b: 5 actors who have collaborated with the most unique directors
<Record actor='Tom Hanks' unique_director_count=11>
<Record actor='Keanu Reeves' unique_director_count=6>
<Record actor='Jack Nicholson' unique_director_count=5>
<Record actor='Hugo Weaving' unique_director_count=4>
<Record actor='Cuba Gooding Jr.' unique_director_count=4>

Query 2a: 5 movies with highest average review
<Record movie='Cloud Atlas' avg_rating=95.0>
<Record movie='Jerry Maguire' avg_rating=92.0>
<Record movie='Unforgiven' avg_rating=85.0>
<Record movie='The Replacements' avg_rating=75.66666666666667>
<Record movie='The Da Vinci Code' avg_rating=66.5>

Query 2b: 5 movies with the highest number of unique reviewers
<Record movie='The Replacements' unique_reviewer_count=3>
<Record movie='The Da Vinci Code' unique_reviewer_count=2>
<Record movie='The Birdcage' unique_reviewer_count=1>
<Record movie='Cloud Atlas' unique_reviewer_count=1>
<Record movie='Unforgiven' unique_reviewer_count=1>
```

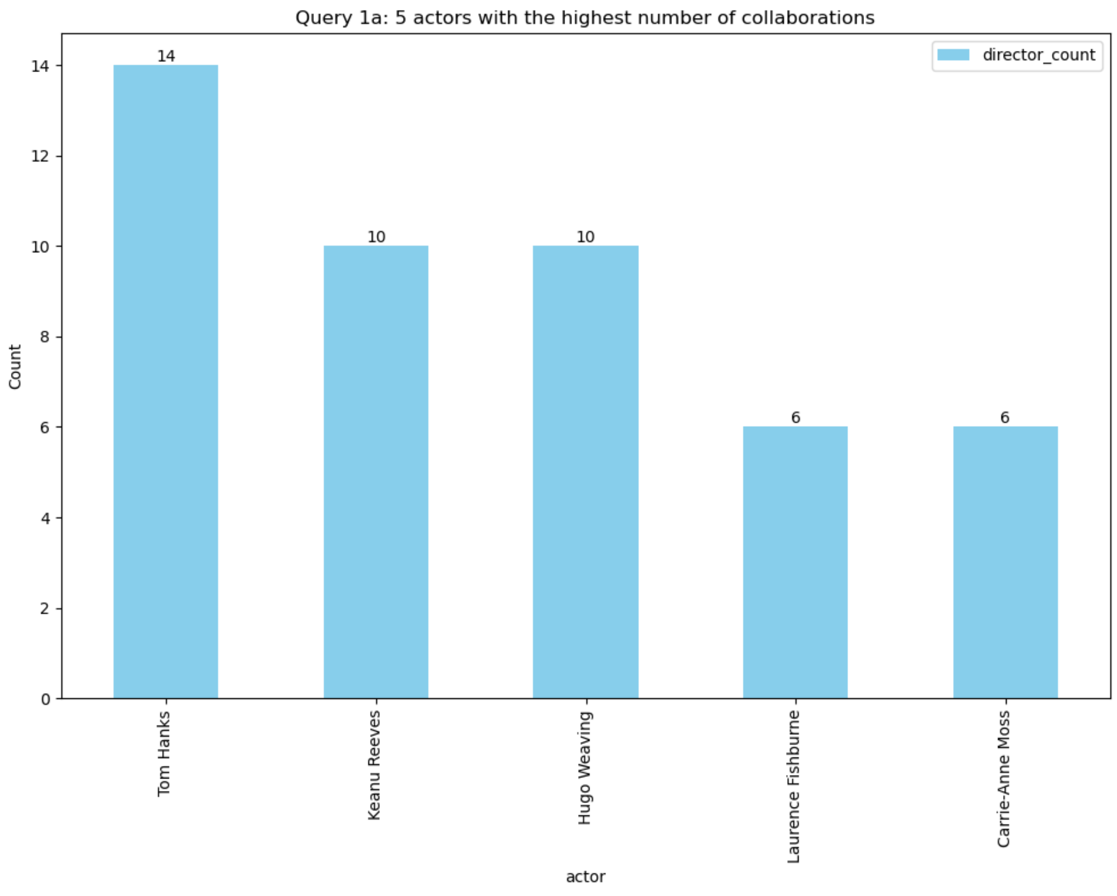
[13]:

```
import matplotlib.pyplot as plt
import pandas as pd
from neo4j import GraphDatabase

# Establish a connection to the remote Neo4j database
uri = "neo4j+s://82d7ec45.databases.neo4j.io"
username = "neo4j"
password = "LFKIPcQfywuwp4Xg8-_TLpJwTxU-h10Ftb7xBYgmZ1o"

driver = GraphDatabase.driver(uri, auth=(username, password))

queries = [
         (
        # Query 1a: 5 actors with the highest number of collaborations
        """MATCH (a:Person)-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(d:Person)
           RETURN a.name AS actor, count(d) AS director_count
           ORDER BY director_count DESC
           LIMIT 5""",
        "Query 1a: 5 actors with the highest number of collaborations"
    ),
    (
        # Query 1b: 5 actors who have collaborated with the most unique␣
 ↪directors
        """MATCH (a:Person)-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(d:Person)
           RETURN a.name AS actor, count(DISTINCT d) AS unique_director_count
           ORDER BY unique_director_count DESC
           LIMIT 5""",
        "Query 1b: 5 actors who have collaborated with the most unique␣
 ↪directors"
    ),
    (
        # Query 2a: 5 movies with highest average review
        """MATCH (m:Movie)<-[r:REVIEWED]-(p:Person)
           RETURN m.title AS movie, avg(r.rating) AS avg_rating
           ORDER BY avg_rating DESC
           LIMIT 5""",
        "Query 2a: 5 movies with highest average review"
    ),
    (
        # Query 2b: 5 movies with the highest number of unique reviewers
        """MATCH (m:Movie)<-[r:REVIEWED]-(p:Person)
           RETURN m.title AS movie, count(DISTINCT p) AS unique_reviewer_count
           ORDER BY unique_reviewer_count DESC
           LIMIT 5""",
        "Query 2b: 5 movies with the highest number of unique reviewers"
    )
]
```

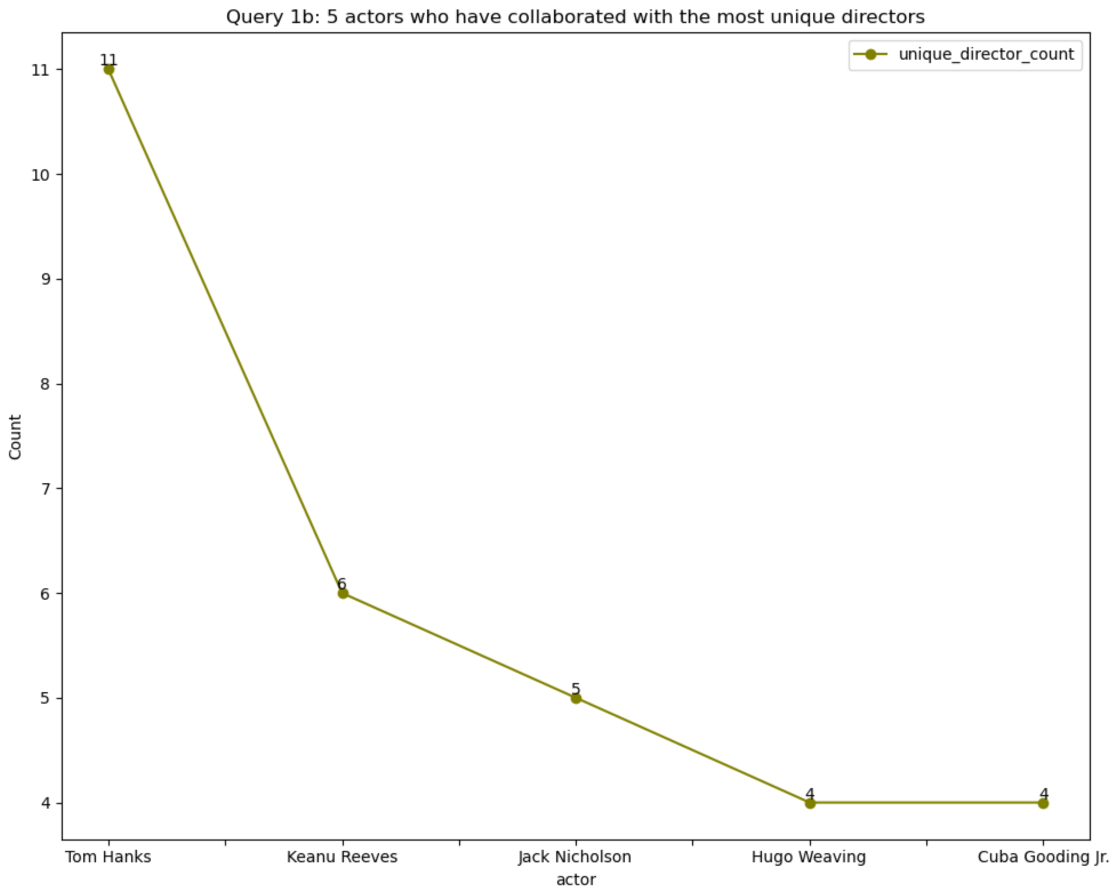
```
with driver.session() as session:
    for index, (query, comment) in enumerate(queries):
        result = session.run(query)
        print(comment)

        # Convert Neo4j result to Pandas DataFrame
        df = pd.DataFrame([r.values() for r in result], columns=result.keys())

        # Create a new figure for each plot with specified size
        fig, ax = plt.subplots(figsize=(10, 8))

        # Choose a different kind of plot for each query
        if index == 0:
            df.plot(kind='bar', x=df.columns[0], y=df.columns[1],␣
 ↪color='skyblue', ax=ax)
            for i, v in df.iterrows():
                ax.text(i, v[df.columns[1]], v[df.columns[1]], color='black',␣
 ↪ha='center', va='bottom')
        elif index == 1:
            df.plot(kind='line', x=df.columns[0], y=df.columns[1],␣
 ↪color='olive', marker='o', ax=ax)
            for i, v in df.iterrows():
                ax.text(i, v[df.columns[1]], v[df.columns[1]], color='black',␣
 ↪ha='center', va='bottom')
        elif index == 2:
            df.plot(kind='area', x=df.columns[0], y=df.columns[1],␣
 ↪color='lightcoral', ax=ax)
            for i, v in df.iterrows():
                ax.text(i, v[df.columns[1]], v[df.columns[1]], color='black',␣
 ↪ha='center', va='bottom')
        elif index == 3:
            df.plot(kind='barh', x=df.columns[0], y=df.columns[1],␣
 ↪color='lightgreen', ax=ax)
            for i, v in df.iterrows():
                ax.text(v[df.columns[1]], i, v[df.columns[1]], color='black',␣
 ↪ha='right', va='center')

        # Set title
        ax.set_title(comment)

        # Set y label
        ax.set_ylabel('Count')

        # Show the plot
        plt.tight_layout()
        plt.show()
```

```
# Close the driver
driver.close()
```

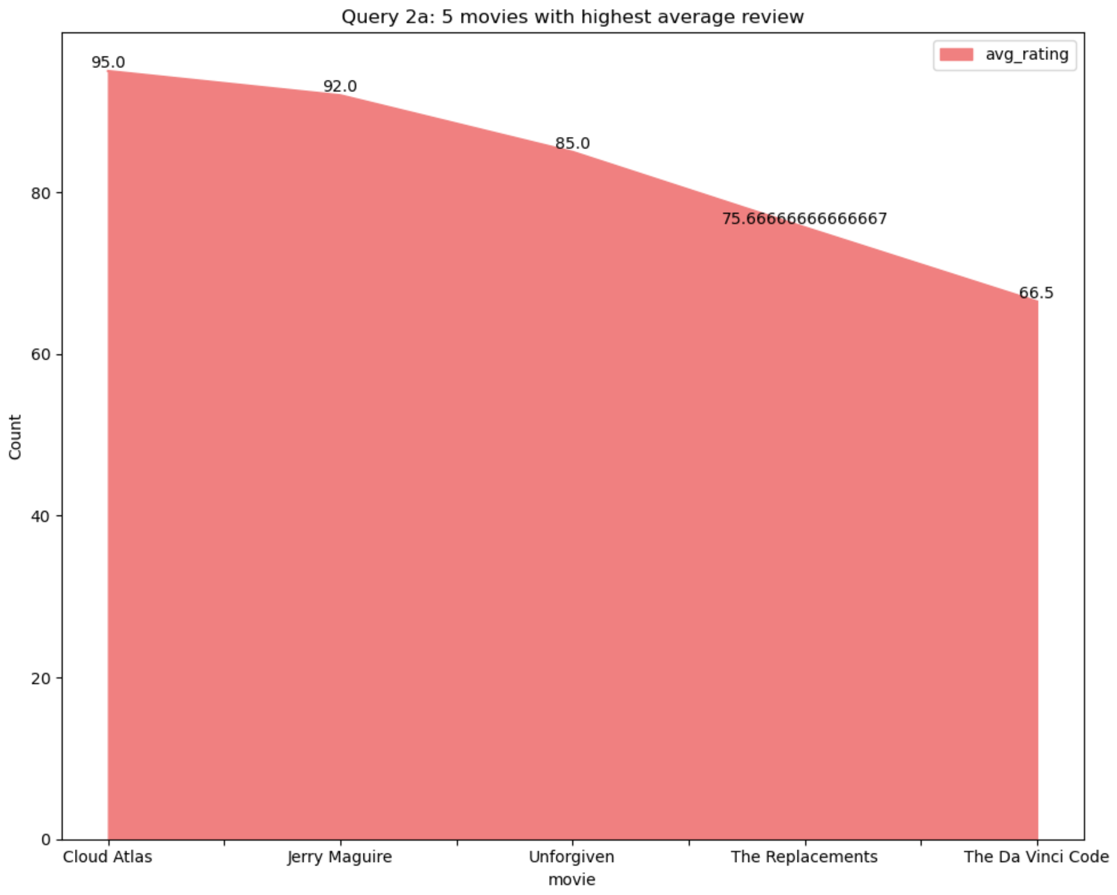
Query 1a: 5 actors with the highest number of collaborations



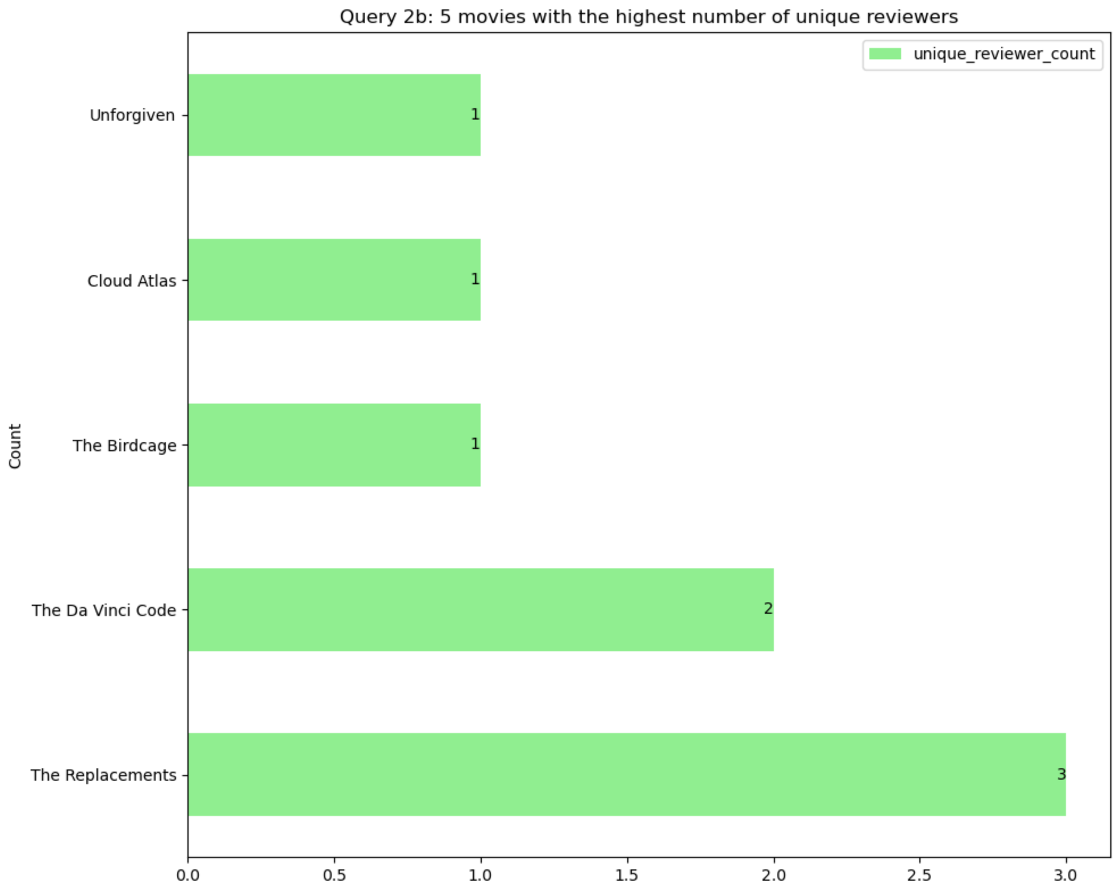
Query 1b: 5 actors who have collaborated with the most unique directors



`Query 2a: 5 movies with highest average review`



Query 2b: 5 movies with the highest number of unique reviewers



0.7 4.6 Conclusion and Recommendations

In conclusion, our extensive research and data analysis of trends within the film industry have led us to several key insights. These findings illuminate potential strategies we could adopt to maximize our success in the competitive landscape of movie production. Based on these insights, we propose the following recommendations for enhancing our content's reach, appeal, and viewership: 1. Leverage Popular Actors for New Productions: Our data suggests that actors like Tom Hanks and Keanu Reeves have had the most collaborations with a diverse set of directors, indicating their wide-ranging experiences and audience appeal. As a production company, we should prioritize bringing these actors on board for future projects, as their involvement could potentially boost our content's reach and appeal. Additionally, engaging with actors such as Jack Nicholson and Cuba Gooding Jr., who have worked with numerous unique directors, could also bring in diverse creative influences and help us cater to a broader spectrum of audience interests.

2. Prioritize Genres and Elements of High-Rated Movies: The high average reviews for movies like 'Cloud Atlas' and 'Jerry Maguire' suggest that they have struck a chord with audiences. This provides an opportunity for us to analyze these films closely and identify the elements that led to their success - be it the storyline, the direction, the cast, or the genre. By replicating these successful elements in our future productions, we can aim to create content that is likely to resonate well with our audience and thereby increase viewership.

3. Cultivate Relationships with Active Reviewers: The high number of unique reviewers for movies such as 'The Replacements' and 'The Da Vinci Code' indicates a deeply engaged audience segment. Encouraging more viewers to write reviews and providing interactive opportunities for them could enhance overall audience engagement. Moreover, these active reviewers could be leveraged as brand ambassadors or influencers, promoting our content within their social circles. This could potentially lead to a wider audience reach, higher viewership, and ultimately, increased revenue for our company.