sc3topdf

May 27, 2023

0.0.1 CSDA 1120 - Group 4 - Sprint 2 - Scenario 3

1 1. Configuring Environment

Install pymongo –> python -m pip install pymongo==3.12.0 On AC prompt shell

Import the MongoClient class from the pymongo library, to interact with a MongoDB database from Python.

1.0.1 Install The Following Libraries

Pymongo, MongoClient, Pandas, Json, Pprint

[16]:
```
from pymongo import MongoClient
from pandas import json_normalize
import json
from pprint import pprint as pp
import pandas as pd
```

Connect to studio 3T mongoDB Database from python

[17]:
```
client = MongoClient ('mongodb://34.171.140.166:27017/Scenario3')
```

[18]:
```
db=client.Scenario3
print (db)
```

```
Database(MongoClient(host=['34.171.140.166:27017'], document_class=dict,
tz_aware=False, connect=True), 'Scenario3')
```

Use Mongoclient and cursor to connect to “restaurants.db”

[19]:
```
with client:
    db = client.Scenario3
    #getCollection("restaurants")
    qresult1 = db.restaurants.find()
    print(type(qresult1))
```

```
<class 'pymongo.cursor.Cursor'>
```

Query and create a dataframe from “restaurants.db” as “df”.

Normalize the dataframe into “df2”

[20]:
```
res=list(qresult1)

df = pd.DataFrame.from_records(res)

df2 = pd.DataFrame.from_records(json_normalize(res))
```

2 1.1 Exploratory Data Analysis

[8]:
```
df2.describe()
```

[8]:
```
                            _id    borough   cuisine grades    name  \
count                     25359      25359     25359  25359   25359
unique                    25359          6        85  23121   20470
top     646a22f720cec63db35433b0  Manhattan  American     []  Subway
freq                          1      10259      6183    738     421

       restaurant_id address.building                    address.coord  \
count          25359            25359                            25359
unique         25359             7990                            22103
top         30075445                1  [-73.77813909999999, 40.6413111]
freq               1              209                               84

       address.street address.zipcode
count           25359           25359
unique           2790             213
top          Broadway           10003
freq              928             686
```

[9]:
```
df.columns
```

[9]:
```
Index(['_id', 'address', 'borough', 'cuisine', 'grades', 'name',
       'restaurant_id'],
      dtype='object')
```

[10]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25359 entries, 0 to 25358
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   _id               25359 non-null  object
 1   borough           25359 non-null  object
 2   cuisine           25359 non-null  object
 3   grades            25359 non-null  object
 4   name              25359 non-null  object
 5   restaurant_id     25359 non-null  object
```

```
 6   address.building  25359 non-null  object
 7   address.coord     25359 non-null  object
 8   address.street    25359 non-null  object
 9   address.zipcode   25359 non-null  object
dtypes: object(10)
memory usage: 1.9+ MB
```

[164]:
```python
# Establish connection to MongoDB server and select "Scenario3" database
with client:
    db = client.Scenario3

    # Count occurrences of each borough
    borough_counts = db.restaurants.aggregate([
        {"$group": {"_id": "$borough", "count": {"$sum": 1}}},
        {"$project": {"borough": "$_id", "count": 1, "_id": 0}}
    ])

    # Print borough counts
    print("Borough Counts:")
    for borough in borough_counts:
        borough_name = borough["borough"]
        count = borough["count"]
        print(f"{borough_name}: {count}")
```

```
Borough Counts:
Bronx: 2338
Brooklyn: 6086
Queens: 5656
Staten Island: 969
Manhattan: 10259
Missing: 51
```

3 2. Business Objectives

1.Optimize current restaurant locations/market presence based on demand. #### 2. Expand restaurant locations

4 3. Decisions

1. Analyze the performance of specific cuisine categories in New York.
2. Develop targeted marketing campaigns that highlight the popular cuisine categories and their associated locations to attract customers. Implement current successful aspects, and find what is currently working in the market already.
3. Asses the New York restaurant market, to find potential opportunities for new restaurant locations.

5 4. Business Questions

1. What are the most and least prevelant cuisines categories in New York?
2. What are the top 3 restaurants names in each borough?
3. Which borough shows the most demand/lowest competition for a potential opening of a new restaurant of a specific cuisine category?

6 5. Database Queries

Q1: What are the most and least prevelant cuisines categories in New York? ### (Query 1 & 2)

[23]:
```
# Query 1

# Establish connection to MongoDB server and select "Scenario3" database
with client:
    db = client.Scenario3

    # List of boroughs
    boroughs = ["Manhattan", "Brooklyn", "Queens", "Bronx", "Staten Island"]

    for borough in boroughs:
        print(f"------ {borough} ------")

        # Aggregation pipeline query to find the top 5 cuisine categories with␣
 ↪highest count in the current borough
        top_5_pipeline = [
            {"$match": {"borough": borough}},
            {"$group": {"_id": "$cuisine", "count": {"$sum": 1}}},
            {"$sort": {"count": -1}},
            {"$limit": 5},
            {"$project": {"_id": 0, "cuisine": "$_id", "count": 1}}
        ]

        top_5_cuisines = db.restaurants.aggregate(top_5_pipeline)

        print("Top 5 cuisine categories:")
        for cuisine in top_5_cuisines:
            cuisine_name = cuisine["cuisine"]
            cuisine_count = cuisine["count"]
            print(f"Cuisine: {cuisine_name}, Count: {cuisine_count}")

        print("\n")

        ##Query2
```

```
        # Aggregation pipeline query to find the bottom 5 cuisine categories␣
↪with lowest count in the current borough
        bottom_5_pipeline = [
            {"$match": {"borough": borough}},
            {"$group": {"_id": "$cuisine", "count": {"$sum": 1}}},
            {"$sort": {"count": 1}},
            {"$limit": 5},
            {"$project": {"_id": 0, "cuisine": "$_id", "count": 1}}
        ]

        bottom_5_cuisines = db.restaurants.aggregate(bottom_5_pipeline)

        print("Bottom 5 cuisine categories:")
        for cuisine in bottom_5_cuisines:
            cuisine_name = cuisine["cuisine"]
            cuisine_count = cuisine["count"]
            print(f"Cuisine: {cuisine_name}, Count: {cuisine_count}")

        print("\n---------------------------------\n")
```

```
------ Manhattan ------
Top 5 cuisine categories:
Cuisine: American, Count: 3205
Cuisine: Café/Coffee/Tea, Count: 680
Cuisine: Italian, Count: 621
Cuisine: Chinese, Count: 510
Cuisine: Japanese, Count: 438


Bottom 5 cuisine categories:
Cuisine: Polynesian, Count: 1
Cuisine: Creole, Count: 1
Cuisine: Fruits/Vegetables, Count: 1
Cuisine: Czech, Count: 1
Cuisine: Californian, Count: 1

---------------------------------

------ Brooklyn ------
Top 5 cuisine categories:
Cuisine: American, Count: 1273
Cuisine: Chinese, Count: 763
Cuisine: Caribbean, Count: 314
Cuisine: Pizza, Count: 296
Cuisine: Café/Coffee/Tea, Count: 289


Bottom 5 cuisine categories:
```

```
Cuisine: Scandinavian, Count: 1
Cuisine: Creole/Cajun, Count: 1
Cuisine: Hawaiian, Count: 1
Cuisine: Southwestern, Count: 1
Cuisine: Afghan, Count: 1

---------------------------------

------ Queens ------
Top 5 cuisine categories:
Cuisine: American, Count: 1040
Cuisine: Chinese, Count: 728
Cuisine: Latin (Cuban, Dominican, Puerto Rican, South & Central American),
Count: 300
Cuisine: Pizza, Count: 277
Cuisine: Other, Count: 236


Bottom 5 cuisine categories:
Cuisine: Australian, Count: 1
Cuisine: Southwestern, Count: 1
Cuisine: Nuts/Confectionary, Count: 1
Cuisine: Soul Food, Count: 1
Cuisine: English, Count: 1

---------------------------------

------ Bronx ------
Top 5 cuisine categories:
Cuisine: American, Count: 411
Cuisine: Chinese, Count: 323
Cuisine: Pizza, Count: 197
Cuisine: Latin (Cuban, Dominican, Puerto Rican, South & Central American),
Count: 187
Cuisine: Spanish, Count: 127


Bottom 5 cuisine categories:
Cuisine: Soups & Sandwiches, Count: 1
Cuisine: French, Count: 1
Cuisine: Fruits/Vegetables, Count: 1
Cuisine: Salads, Count: 1
Cuisine: Hotdogs, Count: 1

---------------------------------

------ Staten Island ------
Top 5 cuisine categories:
```

```
Cuisine: American, Count: 244
Cuisine: Chinese, Count: 88
Cuisine: Italian, Count: 73
Cuisine: Pizza/Italian, Count: 58
Cuisine: Pizza, Count: 53


Bottom 5 cuisine categories:
Cuisine: Turkish, Count: 1
Cuisine: Continental, Count: 1
Cuisine: Not Listed/Not Applicable, Count: 1
Cuisine: Pancakes/Waffles, Count: 1
Cuisine: Steak, Count: 1

---------------------------------
```

[14]:
```python
import pandas as pd
from tabulate import tabulate

# Top 10 most prevalent cuisine categories
top_10_data = {
    'Top 10 Most Prevalent Cuisine Categories': [
        'American', 'Chinese', 'Café/Coffee/Tea', 'Pizza', 'Italian',
        'Latin', 'Japanese', 'Mexican', 'Bakery', 'Spanish'
    ],
    'Count': [6183, 2418, 1214, 1163, 1069, 850, 760, 754, 691, 637]
}

# Bottom 10 least prevalent cuisine categories
bottom_10_data = {
    'Bottom 10 Least Prevalent Cuisine Categories': [
        'Californian', 'Polynesian', 'Chilean', 'Creole/Cajun', 'Café/Coffee/
 ↪Tea',
        'Iranian', 'Hawaiian', 'Soups', 'Nuts/Confectionary', 'Czech'
    ],
    'Count': [1, 1, 1, 1, 2, 2, 3, 4, 6, 6]
}

top_df = pd.DataFrame(top_10_data)
bottom_df = pd.DataFrame(bottom_10_data)

# Set the display options for pandas
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)

# Format the tables as strings
```

```
top_table = tabulate(top_df, headers='keys', tablefmt='fancy_grid',␣
 ↪showindex=False)
bottom_table = tabulate(bottom_df, headers='keys', tablefmt='fancy_grid',␣
 ↪showindex=False)

# Print the horizontally aligned tables
print(tabulate([[top_table, bottom_table]], tablefmt='grid'))
```

```
+-----------------------------------------------------------+--------------------
-----------------------------------------+
|                                 |
                                 |
|   Top 10 Most Prevalent Cuisine Categories      Count   |   Bottom 10 Least
Prevalent Cuisine Categories      Count   |
|                                 |
                                 |
|   American                                       6183   |   Californian
       1   |
|                                 |
                                 |
|   Chinese                                        2418   |   Polynesian
       1   |
|                                 |
                                 |
|   Café/Coffee/Tea                                1214   |   Chilean
       1   |
|                                 |
                                 |
|   Pizza                                          1163   |   Creole/Cajun
       1   |
|                                 |
                                 |
|   Italian                                        1069   |   Café/Coffee/Tea
       2   |
|                                 |
                                 |
|   Latin                                           850   |   Iranian
       2   |
|                                 |
                                 |
|   Japanese                                        760   |   Hawaiian
       3   |
|                                 |
                                 |
|   Mexican                                         754   |   Soups
       4   |
|                                 |
```

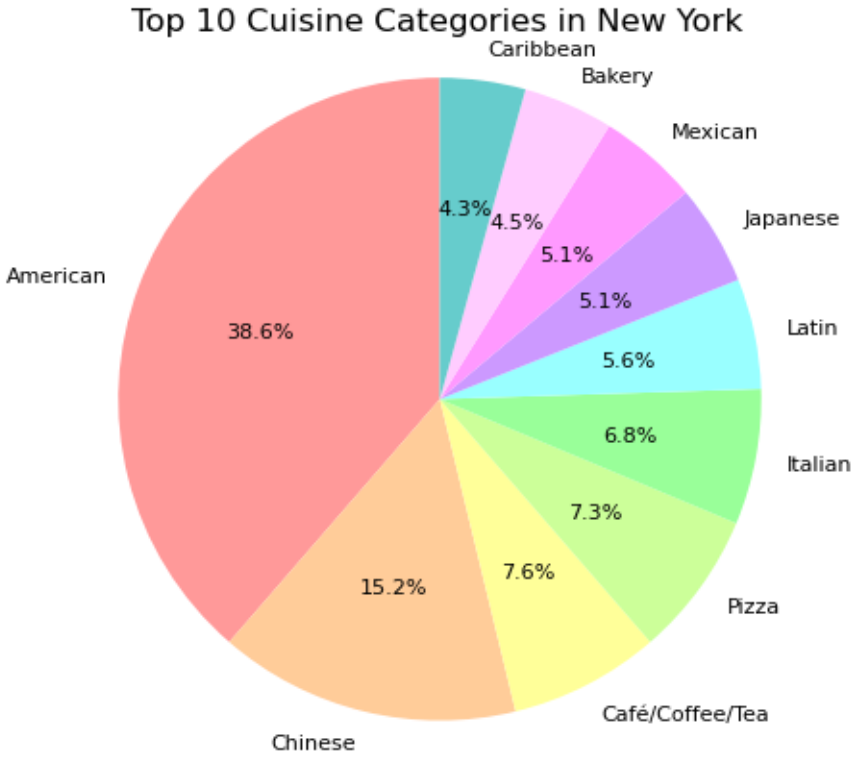
```
                                  |
|  Bakery                                                691   |
Nuts/Confectionary                                         6   |
|                                 |
                                  |
|  Spanish                                               637   |  Czech
       6   |
|                                 |
                                  |
+-------------------------------------------------------------+--------------------
----------------------------------------+
```

6.0.1 Top 5 cuisine categories in each borough

6.0.2 Top 10 cuisine categories based on market prevelance

[24]:
```
import matplotlib.pyplot as plt

# Function to plot pie chart
def plot_pie_chart(categories, percentages, title):
    # Create figure and axis
    fig, ax = plt.subplots()

    # Define colors for the pie chart
    colors = ['#FF9999', '#FFCC99', '#FFFF99', '#CCFF99', '#99FF99', '#99FFFF',␣
 ↪'#CC99FF', '#FF99FF', '#FFCCFF', '#66CCCC']

    # Plot the pie chart
    wedges, texts, autotexts = ax.pie(percentages, labels=categories,␣
 ↪colors=colors, startangle=90,
                                      autopct='%1.1f%%', textprops={'fontsize':␣
 ↪8})

    # Set title
    ax.set_title(title)

    # Equal aspect ratio ensures that pie is drawn as a circle
    ax.axis('equal')

    # Show the plot
    plt.show()

# Cuisine categories and their percentages
categories = [
    'American', 'Chinese', 'Café/Coffee/Tea', 'Pizza', 'Italian', 'Latin',
    'Japanese', 'Mexican', 'Bakery', 'Caribbean'
]
```

```
percentages = [
    15.3, 6.0, 3.0, 2.9, 2.7, 2.2, 2.0, 2.0, 1.8, 1.7
]

# Plot pie chart for cuisine categories
plot_pie_chart(categories, percentages, "Top 10 Cuisine Categories in New York")
```



Q2. What are the top 3 restaurant names in each borough? ### (Query 2)

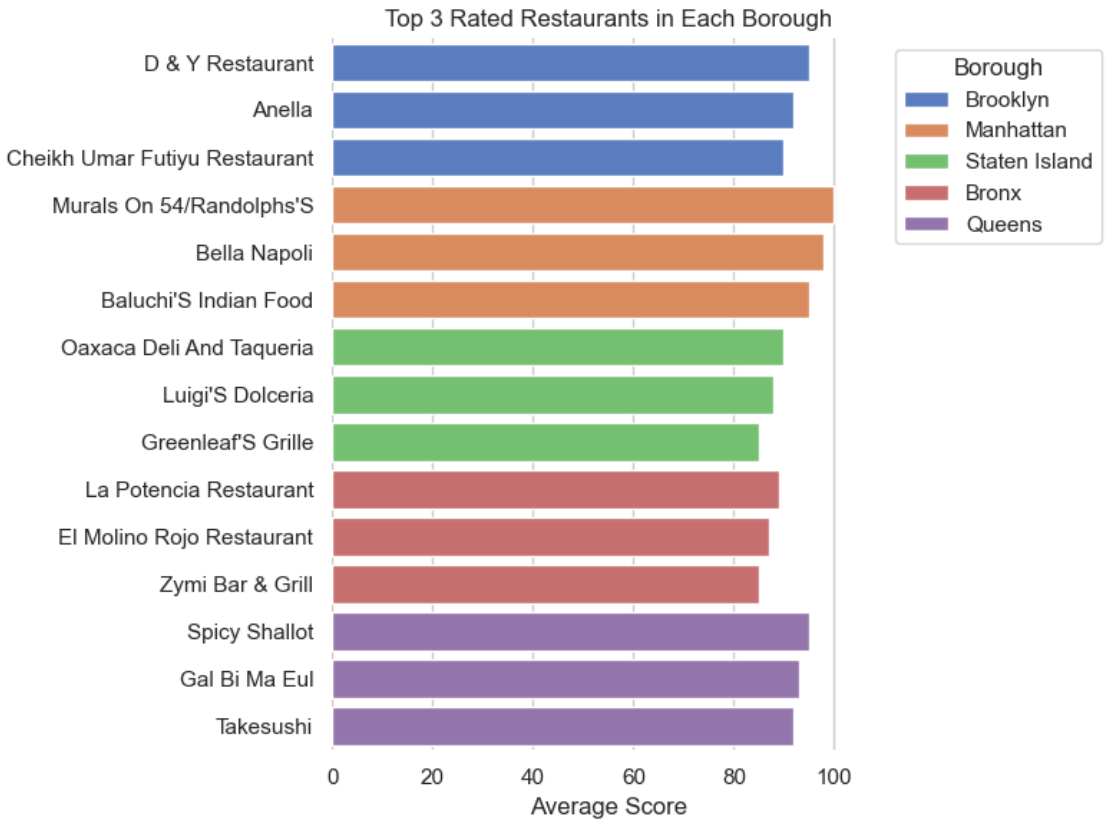
[6]:
```
## Query 2

# Establish connection to MongoDB server and select "Scenario3" database/ List␣
 ↪the boroughs
with client:
    db = client.Scenario3
    boroughs = ["Brooklyn", "Manhattan", "Staten Island", "Bronx", "Queens"]

# Aggregation pipeline query to retrieve top 3 rated restaurants for each␣
 ↪borough.
    for borough in boroughs:
        print(f"Top 3 rated restaurants in {borough}:")
        qresult = db.restaurants.aggregate([
```

```
            {"$match": {"borough": borough}},
            {"$unwind": "$grades"},
            {"$sort": {"grades.score": -1}},
            {"$limit": 3},
            {"$project": {"_id": 0, "name": 1, "score": "$grades.score"}}
        ])
# Print Query Result
        for result in qresult:
            restaurant_name = result['name']
            score = result['score']
            print(f"{restaurant_name} - Score: {score}")

        print("------------------------")
```

```
Top 3 rated restaurants in Brooklyn:
D & Y Restaurant - Score: 86
Anella - Score: 81
Cheikh Umar Futiyu Restaurant - Score: 78
------------------------
Top 3 rated restaurants in Manhattan:
Murals On 54/Randolphs'S - Score: 131
Baluchi'S Indian Food - Score: 98
Bella Napoli - Score: 98
------------------------
Top 3 rated restaurants in Staten Island:
Oaxaca Deli And Taqueria - Score: 68
Luigi'S Dolceria - Score: 65
Greenleaf'S Grille - Score: 60
------------------------
Top 3 rated restaurants in Bronx:
La Potencia Restaurant - Score: 82
El Molino Rojo Restaurant - Score: 76
Zymi Bar & Grill - Score: 75
------------------------
Top 3 rated restaurants in Queens:
Spicy Shallot - Score: 84
Gal Bi Ma Eul - Score: 78
Los Mismo Restaurant - Score: 73
------------------------
```

[39]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Data for the bar chart (example data)
boroughs = ['Brooklyn', 'Manhattan', 'Staten Island', 'Bronx', 'Queens']
restaurant_names = [
```

```
    ['D & Y Restaurant', 'Anella', 'Cheikh Umar Futiyu Restaurant'],
    ['Murals On 54/Randolphs\'S', 'Bella Napoli', 'Baluchi\'S Indian Food'],
    ['Oaxaca Deli And Taqueria', 'Luigi\'S Dolceria', 'Greenleaf\'S Grille'],
    ['La Potencia Restaurant', 'El Molino Rojo Restaurant', 'Zymi Bar & Grill'],
    ['Spicy Shallot', 'Gal Bi Ma Eul', 'Takesushi']
]
average_scores = [[95, 92, 90], [100, 98, 95], [90, 88, 85], [89, 87, 85], [95,␣
 ↪93, 92]]

# Set the style of the plot using Seaborn
sns.set(style="whitegrid")

# Create a horizontal bar chart using Seaborn
fig, ax = plt.subplots(figsize=(8, 6))

# Flatten the nested restaurant names and average scores for plotting
flat_names = [name for sublist in restaurant_names for name in sublist]
flat_scores = [score for sublist in average_scores for score in sublist]

# Create a color palette for the bars
colors = sns.color_palette("muted", n_colors=5)

# Plot the horizontal bars
sns.barplot(x=flat_scores, y=flat_names, hue=np.repeat(boroughs, 3),
            palette=colors, dodge=False, ax=ax)

# Customize the plot
ax.set(xlim=(0, max(flat_scores) + 10), ylabel="", xlabel="Average Score")
sns.despine(left=True, bottom=True)
ax.legend(title="Borough", bbox_to_anchor=(1, 1), loc="upper left")
ax.set_title("Top 3 Rated Restaurants in Each Borough")
# Show the plot
plt.tight_layout()
plt.show()
```



6.1 Q3. Which borough shows the most demand/lowest competition for a potential opening of a new restaurant of a specific cuisine category?

6.2 Data Validation

6.2.1 Total # of ratings in the dataset

[91]:
```
# Establish connection to MongoDB server and select "Scenario3" database
with client:
    db = client.Scenario3

    # Count the total number of ratings in the dataset
    total_ratings = db.restaurants.aggregate([
        {"$project": {"ratings_count": {"$size": "$grades"}}},
        {"$group": {"_id": None, "total_ratings": {"$sum": "$ratings_count"}}}
    ])

    # Extract the total number of ratings from the result
    total_ratings = list(total_ratings)[0]['total_ratings']

    print("Total number of ratings in the dataset:", total_ratings)
```

```
Total number of ratings in the dataset: 93463
```

6.2.2 Total number of ratings in each cuisine category / Top 15

[166]:
```python
# Establish connection to MongoDB server and select "Scenario3" database
with client:
    db = client.Scenario3

    # Aggregation pipeline to count the number of ratings for each cuisine␣
 ↪category
    pipeline = [
        {"$unwind": "$grades"},
        {"$group": {"_id": "$cuisine", "count": {"$sum": 1}}},
        {"$sort": {"count": -1}},
        {"$limit": 15}
    ]

    # Retrieve the top 15 cuisine categories with the highest number of ratings
    top_cuisines = list(db.restaurants.aggregate(pipeline))

    # Update the pipeline to sort in ascending order for the bottom 10
    pipeline[2]["$sort"] = {"count": 1}

    # Retrieve the bottom 10 cuisine categories with the lowest number of␣
 ↪ratings
    bottom_cuisines = list(db.restaurants.aggregate(pipeline))

    # Print the total number of ratings in each cuisine category
    print("Top 15 Cuisine Categories by Ratings:")
    for i, cuisine in enumerate(top_cuisines, start=1):
        cuisine_name = cuisine["_id"]
        rating_count = cuisine["count"]
        print(f"{i}. Cuisine: {cuisine_name}, Ratings: {rating_count}")

    print("\n-----------------------------------\n")
```

```
Top 15 Cuisine Categories by Ratings:
1. Cuisine: American, Ratings: 23491
2. Cuisine: Chinese, Ratings: 9349
3. Cuisine: Pizza, Ratings: 4698
4. Cuisine: Italian, Ratings: 4340
5. Cuisine: Café/Coffee/Tea, Ratings: 4047
6. Cuisine: Latin (Cuban, Dominican, Puerto Rican, South & Central American),
Ratings: 3830
7. Cuisine: Mexican, Ratings: 2973
8. Cuisine: Japanese, Ratings: 2908
9. Cuisine: Bakery, Ratings: 2873
```

```
10. Cuisine: Caribbean, Ratings: 2613
11. Cuisine: Spanish, Ratings: 2092
12. Cuisine: Pizza/Italian, Ratings: 1875
13. Cuisine: Donuts, Ratings: 1855
14. Cuisine: Hamburgers, Ratings: 1760
15. Cuisine: Sandwiches, Ratings: 1718

---------------------------------
```

6.2.3 Top 15 cuisine categories with highest ratings and at least (1718) ratings / for entire dataset

[12]:
```python
# Establish connection to MongoDB server and select "Scenario3" database
with client:
    db = client.Scenario3

    # Minimum number of ratings required for a cuisine category to be considered
    min_ratings = 5

    # Aggregation pipeline query to calculate the rating for each cuisine␣
 ↪category
    pipeline = [
        {"$unwind": "$grades"},
        {"$group": {"_id": "$cuisine", "rating": {"$avg": "$grades.score"},␣
 ↪"count": {"$sum": 1}}},
        {"$match": {"count": {"$gte": min_ratings}}},
        {"$sort": {"rating": -1}},
        {"$project": {"_id": 0, "cuisine": "$_id", "rating": {"$round":␣
 ↪["$rating", 2]}}}
    ]

    cuisine_ratings = list(db.restaurants.aggregate(pipeline))

    print("Top 15 cuisine categories with highest ratings and at least 1718␣
 ↪ratings:")
    for i, cuisine in enumerate(cuisine_ratings[:100], start=1):
        cuisine_name = cuisine["cuisine"]
        rating = cuisine["rating"]
        print(f"{i}. Cuisine: {cuisine_name}, Rating: {rating}")

    print("\n---------------------------------\n")
```

```
Top 15 cuisine categories with highest ratings and at least 1718 ratings:
1. Cuisine: Iranian, Rating: 17.12
2. Cuisine: Hawaiian, Rating: 15.55
3. Cuisine: Chinese/Japanese, Rating: 14.87
```

4. Cuisine: Polynesian, Rating: 14.6
5. Cuisine: Creole, Rating: 14.34
6. Cuisine: Chinese/Cuban, Rating: 13.83
7. Cuisine: Bangladeshi, Rating: 13.8
8. Cuisine: African, Rating: 13.76
9. Cuisine: Pakistani, Rating: 13.7
10. Cuisine: Korean, Rating: 13.52
11. Cuisine: Peruvian, Rating: 13.15
12. Cuisine: Vietnamese/Cambodian/Malaysia, Rating: 13.11
13. Cuisine: German, Rating: 13.06
14. Cuisine: Latin (Cuban, Dominican, Puerto Rican, South & Central American), Rating: 13.04
15. Cuisine: Asian, Rating: 13.01
16. Cuisine: Russian, Rating: 12.97
17. Cuisine: Filipino, Rating: 12.92
18. Cuisine: Delicatessen, Rating: 12.9
19. Cuisine: Thai, Rating: 12.9
20. Cuisine: Indian, Rating: 12.9
21. Cuisine: Japanese, Rating: 12.87
22. Cuisine: Chinese, Rating: 12.79
23. Cuisine: Spanish, Rating: 12.55
24. Cuisine: Mexican, Rating: 12.55
25. Cuisine: Brazilian, Rating: 12.26
26. Cuisine: Caribbean, Rating: 12.17
27. Cuisine: Soul Food, Rating: 12.12
28. Cuisine: Polish, Rating: 12.1
29. Cuisine: Jewish/Kosher, Rating: 12.07
30. Cuisine: Pancakes/Waffles, Rating: 12.07
31. Cuisine: Eastern European, Rating: 12.05
32. Cuisine: Southwestern, Rating: 11.97
33. Cuisine: Pizza/Italian, Rating: 11.95
34. Cuisine: Italian, Rating: 11.91
35. Cuisine: Continental, Rating: 11.87
36. Cuisine: Moroccan, Rating: 11.86
37. Cuisine: Portuguese, Rating: 11.83
38. Cuisine: Turkish, Rating: 11.82
39. Cuisine: English, Rating: 11.65
40. Cuisine: Middle Eastern, Rating: 11.61
41. Cuisine: Tapas, Rating: 11.6
42. Cuisine: French, Rating: 11.44
43. Cuisine: Scandinavian, Rating: 11.42
44. Cuisine: Bakery, Rating: 11.39
45. Cuisine: Greek, Rating: 11.35
46. Cuisine: Seafood, Rating: 11.32
47. Cuisine: Tex-Mex, Rating: 11.3
48. Cuisine: Afghan, Rating: 11.26
49. Cuisine: Pizza, Rating: 11.26
50. Cuisine: Vegetarian, Rating: 11.1

```
51. Cuisine: American, Rating: 11.05
52. Cuisine: Cajun, Rating: 11.0
53. Cuisine: Irish, Rating: 10.98
54. Cuisine: Barbecue, Rating: 10.98
55. Cuisine: Mediterranean, Rating: 10.96
56. Cuisine: Indonesian, Rating: 10.86
57. Cuisine: Chicken, Rating: 10.57
58. Cuisine: Bagels/Pretzels, Rating: 10.56
59. Cuisine: Steak, Rating: 10.53
60. Cuisine: Bottled beverages, including water, sodas, juices, etc., Rating:
10.34
61. Cuisine: CafÃ©/Coffee/Tea, Rating: 10.14
62. Cuisine: Czech, Rating: 10.12
63. Cuisine: Ethiopian, Rating: 10.11
64. Cuisine: Egyptian, Rating: 10.06
65. Cuisine: Armenian, Rating: 10.04
66. Cuisine: Australian, Rating: 9.65
67. Cuisine: Hamburgers, Rating: 9.54
68. Cuisine: Other, Rating: 9.22
69. Cuisine: Fruits/Vegetables, Rating: 9.05
70. Cuisine: Sandwiches/Salads/Mixed Buffet, Rating: 9.0
71. Cuisine: Soups & Sandwiches, Rating: 8.97
72. Cuisine: Salads, Rating: 8.97
73. Cuisine: Soups, Rating: 8.92
74. Cuisine: Juice, Smoothies, Fruit Salads, Rating: 8.75
75. Cuisine: Sandwiches, Rating: 8.67
76. Cuisine: Café/Coffee/Tea, Rating: 8.67
77. Cuisine: Not Listed/Not Applicable, Rating: 8.41
78. Cuisine: Donuts, Rating: 8.31
79. Cuisine: Ice Cream, Gelato, Yogurt, Ices, Rating: 8.31
80. Cuisine: Hotdogs, Rating: 7.46
81. Cuisine: Nuts/Confectionary, Rating: 7.22
82. Cuisine: Hotdogs/Pretzels, Rating: 5.59

----------------------------------
```

6.2.4 Top 5 cuisines in each borough based on ratings

[114]:
```
pipeline = [
    {"$match": {"borough": {"$nin": ["", "Missing"]}}},
    {"$unwind": "$grades"},
    {"$group": {"_id": {"borough": "$borough", "cuisine": "$cuisine"},␣
 ↪"total_ratings": {"$sum": 1}}},
    {"$sort": {"total_ratings": -1}},
    {"$group": {"_id": "$_id.borough", "cuisine_ratings": {"$push": {"cuisine":␣
 ↪"$_id.cuisine", "total_ratings": "$total_ratings"}}}}
```

```
]


borough_cuisine_ratings = list(db.restaurants.aggregate(pipeline))

for borough in borough_cuisine_ratings:
    borough_name = borough["_id"]
    cuisine_ratings = borough["cuisine_ratings"]
    print(f"Borough: {borough_name}")
    top_cuisines = sorted(cuisine_ratings, key=lambda x: x["total_ratings"],␣
 ↪reverse=True)[:5]
    for cuisine in top_cuisines:
        cuisine_name = cuisine["cuisine"]
        total_ratings = cuisine["total_ratings"]
        print(f"Cuisine: {cuisine_name}, Total Ratings: {total_ratings}")
    print("----------------------------------")
```

```
Borough: Brooklyn
Cuisine: American, Total Ratings: 4695
Cuisine: Chinese, Total Ratings: 2981
Cuisine: Caribbean, Total Ratings: 1283
Cuisine: Pizza, Total Ratings: 1196
Cuisine: Café/Coffee/Tea, Total Ratings: 929
----------------------------------
Borough: Bronx
Cuisine: American, Total Ratings: 1561
Cuisine: Chinese, Total Ratings: 1296
Cuisine: Latin (Cuban, Dominican, Puerto Rican, South & Central American), Total
Ratings: 824
Cuisine: Pizza, Total Ratings: 788
Cuisine: Caribbean, Total Ratings: 440
----------------------------------
Borough: Staten Island
Cuisine: American, Total Ratings: 850
Cuisine: Chinese, Total Ratings: 287
Cuisine: Italian, Total Ratings: 273
Cuisine: Pizza/Italian, Total Ratings: 214
Cuisine: Pizza, Total Ratings: 185
----------------------------------
Borough: Manhattan
Cuisine: American, Total Ratings: 12452
Cuisine: Italian, Total Ratings: 2622
Cuisine: Café/Coffee/Tea, Total Ratings: 2320
Cuisine: Chinese, Total Ratings: 2058
Cuisine: Japanese, Total Ratings: 1705
----------------------------------
Borough: Queens
Cuisine: American, Total Ratings: 3916
```

```
Cuisine: Chinese, Total Ratings: 2719
Cuisine: Latin (Cuban, Dominican, Puerto Rican, South & Central American), Total
Ratings: 1354
Cuisine: Pizza, Total Ratings: 1097
Cuisine: Bakery, Total Ratings: 895
----------------------------------
```

6.3 Top 5 cuisines in each borough/ based on average rating per rating count (Query 4)

[90]:
```
### Query 4

# Establish connection to MongoDB server and select "Scenario3" database
with client:
    db = client.Scenario3

    # Minimum number of ratings required for a cuisine category to be considered
    min_ratings = {"Staten Island": 250, "Bronx": 500, "default": 1000}

    # List of boroughs
    boroughs = ["Brooklyn", "Queens", "Manhattan", "Staten Island", "Bronx"]

    for borough in boroughs:
        print(f"\n------ {borough} ------\n")

        # Get the minimum rating count based on borough
        min_rating_count = min_ratings.get(borough, min_ratings["default"])

        # Aggregation pipeline query to calculate the average rating for each␣
↪cuisine category in the current borough
        pipeline = [
            {"$match": {"borough": borough}},
            {"$unwind": "$grades"},
            {"$group": {"_id": "$cuisine", "rating": {"$avg": "$grades.score"},␣
↪"count": {"$sum": 1}}},
            {"$match": {"count": {"$gte": min_rating_count}}},
            {"$sort": {"rating": -1}},
            {"$project": {"_id": 0, "cuisine": "$_id", "rating": {"$round":␣
↪["$rating", 2]}}}
        ]

        cuisine_ratings = list(db.restaurants.aggregate(pipeline))

        print("Top 5 cuisine categories with highest ratings:")
        for i, cuisine in enumerate(cuisine_ratings[:5], start=1):
            cuisine_name = cuisine["cuisine"]
            rating = cuisine["rating"]
```

```
            print(f"{i}. Cuisine: {cuisine_name}, Rating: {rating}")

        print("\n")
```

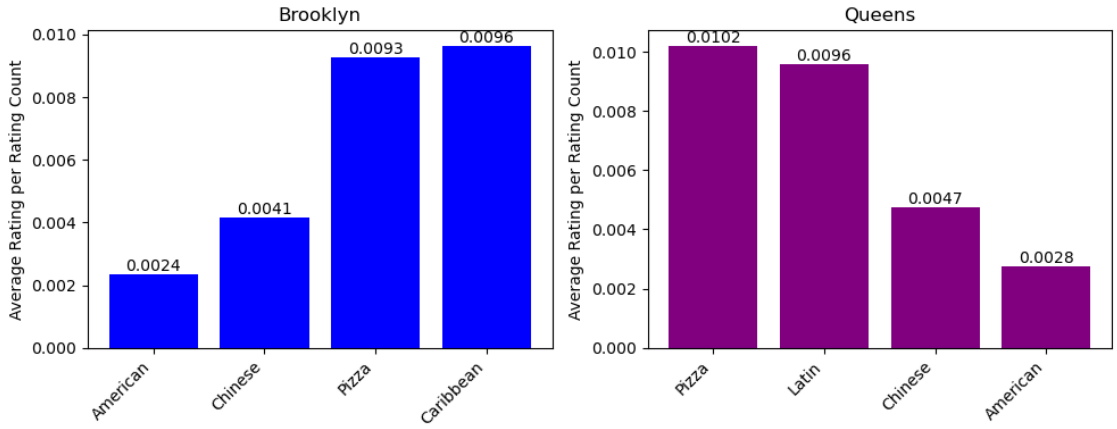
```
------ Brooklyn ------

Top 5 cuisine categories with highest ratings:
1. Cuisine: Caribbean, Rating: 12.38
2. Cuisine: Chinese, Rating: 12.36
3. Cuisine: Pizza, Rating: 11.08
4. Cuisine: American, Rating: 11.04



------ Queens ------

Top 5 cuisine categories with highest ratings:
1. Cuisine: Latin (Cuban, Dominican, Puerto Rican, South & Central American),
Rating: 12.99
2. Cuisine: Chinese, Rating: 12.89
3. Cuisine: Pizza, Rating: 11.2
4. Cuisine: American, Rating: 10.82



------ Manhattan ------

Top 5 cuisine categories with highest ratings:
1. Cuisine: Chinese, Rating: 14.44
2. Cuisine: Japanese, Rating: 13.08
3. Cuisine: Mexican, Rating: 12.15
4. Cuisine: Italian, Rating: 12.11
5. Cuisine: Pizza, Rating: 11.66



------ Staten Island ------

Top 5 cuisine categories with highest ratings:
1. Cuisine: Chinese, Rating: 12.45
2. Cuisine: Italian, Rating: 11.86
3. Cuisine: American, Rating: 11.44



------ Bronx ------
```

```
Top 5 cuisine categories with highest ratings:
1. Cuisine: Latin (Cuban, Dominican, Puerto Rican, South & Central American),
Rating: 13.36
2. Cuisine: Chinese, Rating: 11.04
3. Cuisine: Pizza, Rating: 10.85
4. Cuisine: American, Rating: 10.47
```

6.4 Plotting average rating per rating count for Brooklyn & Queens

[215]:
```python
import matplotlib.pyplot as plt
import numpy as np

# Define the cuisine categories and their respective average ratings and rating␣
 ↪counts for Brooklyn
cuisine_categories_brooklyn = ["Caribbean", "Chinese", "American", "Pizza"]
average_ratings_brooklyn = [12.38, 12.36, 11.04, 11.08]
rating_counts_brooklyn = [1283, 2981, 4695, 1196]

# Calculate the average rating per rating count for each cuisine category in␣
 ↪Brooklyn
average_rating_per_count_brooklyn = [rating / count for rating, count in␣
 ↪zip(average_ratings_brooklyn, rating_counts_brooklyn)]

# Sort the cuisine categories and their average ratings in ascending order
sorted_indices_brooklyn = np.argsort(average_rating_per_count_brooklyn)
cuisine_categories_brooklyn = [cuisine_categories_brooklyn[i] for i in␣
 ↪sorted_indices_brooklyn]
average_rating_per_count_brooklyn = [average_rating_per_count_brooklyn[i] for i␣
 ↪in sorted_indices_brooklyn]

# Define the cuisine categories and their respective average ratings and rating␣
 ↪counts for Queens
cuisine_categories_queens = ["Latin", "Chinese", "American", "Pizza"]
average_ratings_queens = [12.99, 12.89, 10.82, 11.2]
rating_counts_queens = [1354, 2719, 3916, 1097]

# Calculate the average rating per rating count for each cuisine category in␣
 ↪Queens
average_rating_per_count_queens = [rating / count for rating, count in␣
 ↪zip(average_ratings_queens, rating_counts_queens)]

# Sort the cuisine categories and their average ratings in descending order
sorted_indices_queens = np.argsort(average_rating_per_count_queens)[::-1]  #␣
 ↪Reverse the order
```

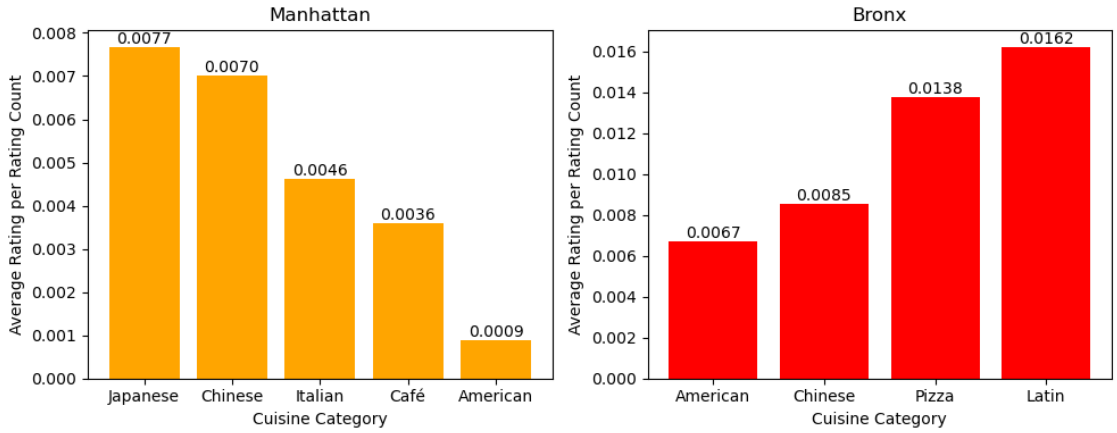
```
cuisine_categories_queens = [cuisine_categories_queens[i] for i in␣
 ↪sorted_indices_queens]
average_rating_per_count_queens = [average_rating_per_count_queens[i] for i in␣
 ↪sorted_indices_queens]

# Set up the subplot with 1 row and 2 columns
fig, axs = plt.subplots(1, 2, figsize=(10, 4))

# Plot the data for Brooklyn with green bars
axs[0].bar(np.arange(len(cuisine_categories_brooklyn)),␣
 ↪average_rating_per_count_brooklyn, color='blue')
axs[0].set_xticks(np.arange(len(cuisine_categories_brooklyn)))
axs[0].set_xticklabels(cuisine_categories_brooklyn, rotation=45, ha="right")
axs[0].set_ylabel("Average Rating per Rating Count")
axs[0].set_title("Brooklyn")

# Add labels for each data point in Brooklyn
for i in range(len(cuisine_categories_brooklyn)):
    axs[0].text(i, average_rating_per_count_brooklyn[i] + 0.000001,␣
 ↪f"{average_rating_per_count_brooklyn[i]:.4f}", ha="center", va="bottom")

# Plot the data for Queens with purple bars
axs[1].bar(np.arange(len(cuisine_categories_queens)),␣
 ↪average_rating_per_count_queens, color='purple')
axs[1].set_xticks(np.arange(len(cuisine_categories_queens)))
axs[1].set_xticklabels(cuisine_categories_queens, rotation=45, ha="right")
axs[1].set_ylabel("Average Rating per Rating Count")
axs[1].set_title("Queens")

# Add labels for each data point in Queens
for i in range(len(cuisine_categories_queens)):
    axs[1].text(i, average_rating_per_count_queens[i] + 0.0000001,␣
 ↪f"{average_rating_per_count_queens[i]:.4f}", ha="center", va="bottom")

# Adjust the layout to prevent labels from overlapping
plt.tight_layout()

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



6.5 Plotting average rating per rating count for Manhattan & The Bronx

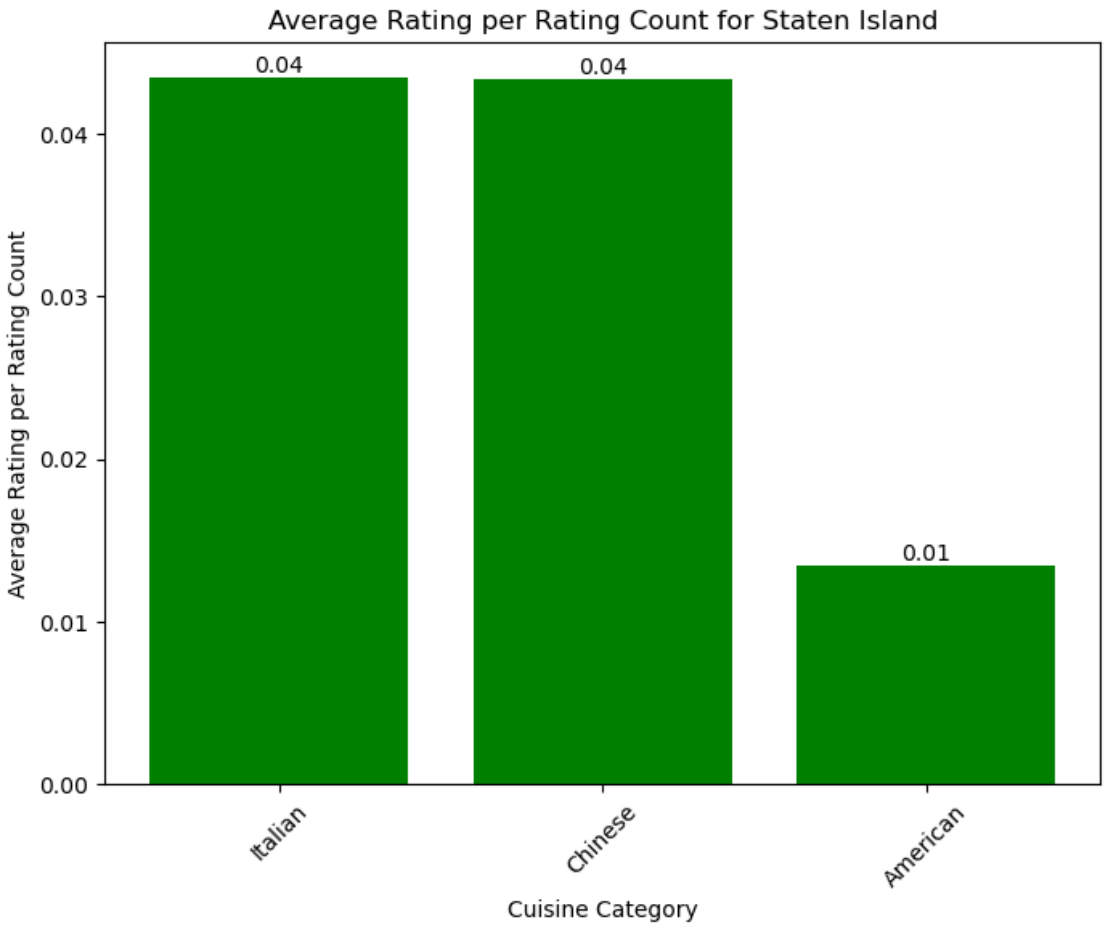
[9]:
```
import matplotlib.pyplot as plt

# Define the cuisine categories and their respective average ratings and rating␣
 ↪counts for Manhattan
cuisine_categories_manhattan = ["Japanese", "Chinese", "Italian", "Pizza",␣
 ↪"Café", "American", "French"]
average_ratings_manhattan = [13.08, 14.44, 12.11, 11.66, 8.35, 11.18, 11.39]
rating_counts_manhattan = [1705, 2058, 2622, None, 2320, 12452, None]

# Calculate the average rating per rating count for each cuisine category in␣
 ↪Manhattan
average_rating_per_count_manhattan = []
for rating, count in zip(average_ratings_manhattan, rating_counts_manhattan):
    if count is not None:
        average_rating_per_count_manhattan.append(rating / count)
    else:
        average_rating_per_count_manhattan.append(None)

# Remove cuisine categories with missing rating counts
cuisine_categories_manhattan = [category for category, count in␣
 ↪zip(cuisine_categories_manhattan, rating_counts_manhattan) if count is not␣
 ↪None]
average_rating_per_count_manhattan = [score for score in␣
 ↪average_rating_per_count_manhattan if score is not None]

# Define the cuisine categories and their respective average ratings and rating␣
 ↪counts for the Bronx
cuisine_categories_bronx = ["American", "Chinese", "Pizza", "Latin"]
average_ratings_bronx = [10.47, 11.04, 10.85, 13.36]
```

```
rating_counts_bronx = [1561, 1296, 788, 824]

# Calculate the average rating per rating count for each cuisine category in␣
 ↪the Bronx
average_rating_per_count_bronx = [rating / count for rating, count in␣
 ↪zip(average_ratings_bronx, rating_counts_bronx)]

# Set up the subplot with 1 row and 2 columns
fig, axs = plt.subplots(1, 2, figsize=(10, 4))

# Plot the data for Manhattan with orange bars
axs[0].bar(cuisine_categories_manhattan, average_rating_per_count_manhattan,␣
 ↪color='orange')
axs[0].set_xlabel("Cuisine Category")
axs[0].set_ylabel("Average Rating per Rating Count")
axs[0].set_title("Manhattan")

# Add labels for each data point in Manhattan
for i in range(len(cuisine_categories_manhattan)):
    axs[0].text(cuisine_categories_manhattan[i],␣
 ↪average_rating_per_count_manhattan[i] + 0.000001,␣
 ↪f"{average_rating_per_count_manhattan[i]:.4f}", ha="center", va="bottom")

# Plot the data for the Bronx with red bars
axs[1].bar(cuisine_categories_bronx, average_rating_per_count_bronx,␣
 ↪color='red')
axs[1].set_xlabel("Cuisine Category")
axs[1].set_ylabel("Average Rating per Rating Count")
axs[1].set_title("Bronx")

# Add labels for each data point in the Bronx
for i in range(len(cuisine_categories_bronx)):
    axs[1].text(cuisine_categories_bronx[i], average_rating_per_count_bronx[i]␣
 ↪+ 0.0000001, f"{average_rating_per_count_bronx[i]:.4f}", ha="center",␣
 ↪va="bottom")

# Adjust the layout to prevent labels from overlapping
plt.tight_layout()

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



[219]:
```
import matplotlib.pyplot as plt

# Define the cuisine categories and their respective average ratings and rating␣
 ↪counts for Staten Island
cuisine_categories_staten_island = ["American", "Chinese", "Italian", "Pizza/
 ↪Italian", "Pizza"]
average_ratings_staten_island = [11.44, 12.45, 11.86, None, None]  # Fill in␣
 ↪the average ratings for each cuisine category
rating_counts_staten_island = [850, 287, 273, 214, 185]  # Fill in the rating␣
 ↪counts for each cuisine category

# Calculate the average rating per rating count for each cuisine category in␣
 ↪Staten Island
average_rating_per_count_staten_island = []
for rating, count in zip(average_ratings_staten_island,␣
 ↪rating_counts_staten_island):
    if rating is not None and count is not None:
        average_rating_per_count_staten_island.append(rating / count)
    else:
        average_rating_per_count_staten_island.append(None)

# Remove cuisine categories with missing average ratings or rating counts
cuisine_categories_staten_island = [category for category, rating, count in␣
 ↪zip(cuisine_categories_staten_island, average_ratings_staten_island,␣
 ↪rating_counts_staten_island) if rating is not None and count is not None]
average_rating_per_count_staten_island = [score for score in␣
 ↪average_rating_per_count_staten_island if score is not None]

# Sort the cuisine categories and average ratings in descending order
sorted_data = sorted(zip(cuisine_categories_staten_island,␣
 ↪average_rating_per_count_staten_island), key=lambda x: x[1], reverse=True)
```

```
cuisine_categories_staten_island, average_rating_per_count_staten_island =␣
 ↪zip(*sorted_data)

# Set up the bar chart
plt.figure(figsize=(8, 6))
plt.bar(range(len(cuisine_categories_staten_island)),␣
 ↪average_rating_per_count_staten_island, color='green')

# Add labels for each data point
for i in range(len(cuisine_categories_staten_island)):
    plt.text(i, average_rating_per_count_staten_island[i],␣
 ↪f"{average_rating_per_count_staten_island[i]:.2f}", ha="center", va="bottom")

# Customize the chart appearance
plt.xticks(range(len(cuisine_categories_staten_island)),␣
 ↪cuisine_categories_staten_island, rotation=45)
plt.ylabel("Average Rating per Rating Count")
plt.xlabel("Cuisine Category")
plt.title("Average Rating per Rating Count for Staten Island")

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



6.6 Correlattion of average rating per rating count vs total cuisine restaurant saturation in each borough

[223]:
```
# Establish connection to MongoDB server and select "Scenario3" database
with client:
    db = client.Scenario3

    # Aggregation pipeline query to calculate the total number of restaurants␣
 ↪in each borough
    pipeline = [
        {"$group": {"_id": "$borough", "count": {"$sum": 1}}},
        {"$project": {"_id": 0, "borough": "$_id", "count": 1}}
    ]

    borough_restaurant_counts = db.restaurants.aggregate(pipeline)

    print("Total number of restaurants in each borough:")
```

```
    for borough in borough_restaurant_counts:
        borough_name = borough["borough"]
        restaurant_count = borough["count"]
        print(f"Borough: {borough_name}, Restaurant Count: {restaurant_count}")
```

```
Total number of restaurants in each borough:
Borough: Queens, Restaurant Count: 5656
Borough: Bronx, Restaurant Count: 2338
Borough: Manhattan, Restaurant Count: 10259
Borough: Brooklyn, Restaurant Count: 6086
Borough: Staten Island, Restaurant Count: 969
Borough: Missing, Restaurant Count: 51
```

6.7 Restaurant saturation per cuisine in each borough

6.7.1 (Query 5)

[226]:
```
##### Query 5

# Establish connection to MongoDB server and select "Scenario3" database
with client:
    db = client.Scenario3

    boroughs = ["Brooklyn", "Bronx", "Manhattan", "Queens", "Staten Island"]

    for borough in boroughs:
        # Aggregation pipeline query to count the number of restaurants for␣
 ↪each cuisine category in the borough
        pipeline = [
            {"$match": {"borough": borough}},
            {"$group": {"_id": "$cuisine", "count": {"$sum": 1}}},
            {"$project": {"_id": 0, "cuisine": "$_id", "count": 1}},
            {"$sort": {"count": -1}},
            {"$limit": 5}
        ]

        cuisine_counts = db.restaurants.aggregate(pipeline)

        print(f"Top 5 cuisine categories with the highest restaurant counts in␣
 ↪{borough}:")
        for cuisine in cuisine_counts:
            cuisine_name = cuisine["cuisine"]
            cuisine_count = cuisine["count"]
            print(f"Cuisine: {cuisine_name}, Count: {cuisine_count}")
        print("\n---------------------------------\n")
```

```
Top 5 cuisine categories with the highest restaurant counts in Brooklyn:
Cuisine: American, Count: 1273
```

```
Cuisine: Chinese, Count: 763
Cuisine: Caribbean, Count: 314
Cuisine: Pizza, Count: 296
Cuisine: Café/Coffee/Tea, Count: 289

---------------------------------

Top 5 cuisine categories with the highest restaurant counts in Bronx:
Cuisine: American, Count: 411
Cuisine: Chinese, Count: 323
Cuisine: Pizza, Count: 197
Cuisine: Latin (Cuban, Dominican, Puerto Rican, South & Central American),
Count: 187
Cuisine: Spanish, Count: 127

---------------------------------

Top 5 cuisine categories with the highest restaurant counts in Manhattan:
Cuisine: American, Count: 3205
Cuisine: Café/Coffee/Tea, Count: 680
Cuisine: Italian, Count: 621
Cuisine: Chinese, Count: 510
Cuisine: Japanese, Count: 438

---------------------------------

Top 5 cuisine categories with the highest restaurant counts in Queens:
Cuisine: American, Count: 1040
Cuisine: Chinese, Count: 728
Cuisine: Latin (Cuban, Dominican, Puerto Rican, South & Central American),
Count: 300
Cuisine: Pizza, Count: 277
Cuisine: Other, Count: 236

---------------------------------

Top 5 cuisine categories with the highest restaurant counts in Staten Island:
Cuisine: American, Count: 244
Cuisine: Chinese, Count: 88
Cuisine: Italian, Count: 73
Cuisine: Pizza/Italian, Count: 58
Cuisine: Pizza, Count: 53

---------------------------------
```

6.8 Scoring each cuisine based on saturation weight & average rating per rating count

6.8.1 Brooklyn

[246]:
```
### Saturation weight is the amount of restaurants per cuisine type/ amount of␣
 ↪restaurants total in the specific borough

### The higher the saturation weight the less market saturation is present.

###Ex: American saturation weight: 1273/6086 = 0.2093

##(this was the highest saturated in this example) giving it the lowest␣
 ↪saturation weight.


# Define the saturation weights and average rating weights for each cuisine
saturation_weights = {
    "American": 1,
    "Chinese": 2,
    "Caribbean": 5,
    "Pizza": 5
}

average_rating_weights = {
    "American": 0.0024,
    "Chinese": 0.0041,
    "Caribbean": 0.096,
    "Pizza": 0.0093
}

# Calculate the weighted scores for each cuisine
weighted_scores = {}

for cuisine in ["American", "Chinese", "Caribbean", "Pizza"]:
    weighted_score = saturation_weights[cuisine] *␣
 ↪average_rating_weights[cuisine]
    weighted_scores[cuisine] = weighted_score

# Find the cuisine with the highest weighted score
best_option = max(weighted_scores, key=weighted_scores.get)

# Output the results
for cuisine in weighted_scores:
    print(f"Cuisine: {cuisine}")
    print(f"Weighted Score: {weighted_scores[cuisine]}")
    print()
```

```
print(f"The best option for opening a restaurant in Brooklyn is: {best_option}")
```

```
Cuisine: American
Weighted Score: 0.0024

Cuisine: Chinese
Weighted Score: 0.0082

Cuisine: Caribbean
Weighted Score: 0.48

Cuisine: Pizza
Weighted Score: 0.0465

The best option for opening a restaurant in Brooklyn is: Caribbean
```

6.8.2 Queens

[243]:
```
# Define the saturation weights and average rating weights for each cuisine
#Queens
saturation_weights = {
    "Latin": 4,
    "Chinese": 2,
    "American": 1,
    "Pizza": 5
}

average_rating_weights = {
    "Latin": 0.0096,
    "Chinese": 0.0047,
    "American": 0.0028,
    "Pizza": 0.0102
}

# Calculate the weighted scores for each cuisine
weighted_scores = {}

for cuisine in ["Latin", "Chinese", "American", "Pizza"]:
    weighted_score = saturation_weights[cuisine] *␣
 ↪average_rating_weights[cuisine]
    weighted_scores[cuisine] = weighted_score

# Find the cuisine with the highest weighted score
best_option = max(weighted_scores, key=weighted_scores.get)

# Output the results
for cuisine in weighted_scores:
```

```
    print(f"Cuisine: {cuisine}")
    print(f"Weighted Score: {weighted_scores[cuisine]}")
    print()

print(f"The best option for opening a restaurant in Queens is: {best_option}")
```

```
Cuisine: Latin
Weighted Score: 0.0384

Cuisine: Chinese
Weighted Score: 0.0094

Cuisine: American
Weighted Score: 0.0028

Cuisine: Pizza
Weighted Score: 0.051000000000000004

The best option for opening a restaurant in Queens is: Pizza
```

6.8.3 Manhattan

[245]:
```
# Define the saturation weights and average rating weights for each cuisine
saturation_weights = {
    "Japanese": 5,
    "Chinese": 4,
    "Italian": 2,
    "Café": 3,
    "American": 1
}

average_rating_weights = {
    "Japanese": 0.0077,
    "Chinese": 0.0070,
    "Italian": 0.0046,
    "Café": 0.0036,
    "American": 0.0009
}

# Calculate the weighted scores for each cuisine
weighted_scores = {}

for cuisine in ["Japanese", "Chinese", "Italian", "Café", "American"]:
    weighted_score = saturation_weights[cuisine] *␣
 ↪average_rating_weights[cuisine]
    weighted_scores[cuisine] = weighted_score

# Find the cuisine with the highest weighted score
```

```
best_option = max(weighted_scores, key=weighted_scores.get)

# Output the results
for cuisine in weighted_scores:
    print(f"Cuisine: {cuisine}")
    print(f"Weighted Score: {weighted_scores[cuisine]}")
    print()

print(f"The best option for opening a restaurant in Manhattan is:␣
 ↪{best_option}")
```

```
Cuisine: Japanese
Weighted Score: 0.0385

Cuisine: Chinese
Weighted Score: 0.028

Cuisine: Italian
Weighted Score: 0.0092

Cuisine: Café
Weighted Score: 0.0108

Cuisine: American
Weighted Score: 0.0009

The best option for opening a restaurant in Manhattan is: Japanese
```

6.8.4 The Bronx

[251]:
```
# Define the saturation weights and average rating weights for each cuisine
saturation_weights = {
    "American": 2.5,
    "Chinese": 3,
    "Pizza": 5,
    "Latin": 4.8
}

average_rating_weights = {
    "American": 0.0067,
    "Chinese": 0.0085,
    "Pizza": 0.0138,
    "Latin": 0.0162
}

# Calculate the weighted scores for each cuisine
weighted_scores = {}
```

```
for cuisine in ["American", "Chinese", "Pizza", "Latin"]:
    weighted_score = saturation_weights[cuisine] *␣
 ↪average_rating_weights[cuisine]
    weighted_scores[cuisine] = weighted_score

# Find the cuisine with the highest weighted score
best_option = max(weighted_scores, key=weighted_scores.get)

# Output the results
for cuisine in weighted_scores:
    print(f"Cuisine: {cuisine}")
    print(f"Weighted Score: {weighted_scores[cuisine]}")
    print()

print(f"The best option for opening a restaurant in the Bronx is:␣
 ↪{best_option}")
```

```
Cuisine: American
Weighted Score: 0.01675

Cuisine: Chinese
Weighted Score: 0.025500000000000002

Cuisine: Pizza
Weighted Score: 0.069

Cuisine: Latin
Weighted Score: 0.07776

The best option for opening a restaurant in the Bronx is: Latin
```

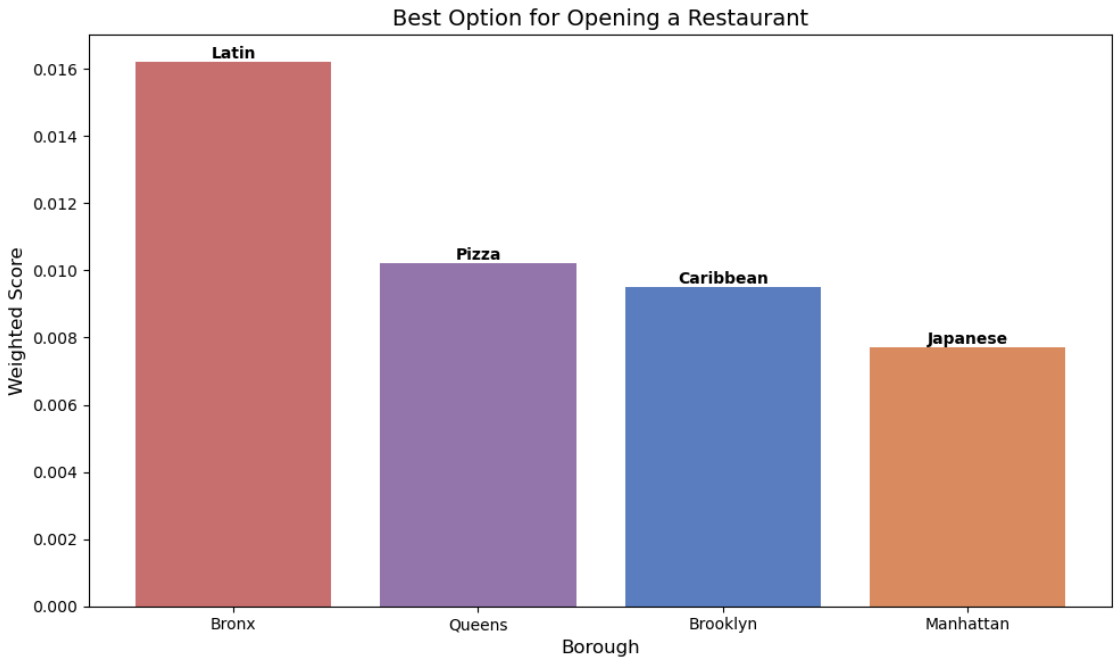
Based on correlating average rating per rating count and the saturation the best option correlates with the highest weighted score. Which means opening a latin restaurant in the bronx is the most optimal location to open up a new restaraurant.

Which is why we have left out staten island as the weighted rating scores were too low compared to the other cuisines in other boroughs.

6.9 Plotting the best option for new restaurant location

[15]:
```
import matplotlib.pyplot as plt

boroughs = ['Bronx', 'Queens', 'Brooklyn', 'Manhattan']
best_options = ['Latin', 'Pizza', 'Caribbean', 'Japanese']
scores = [0.0162, 0.0102, 0.0095, 0.0077]

plt.figure(figsize=(10, 6))
plt.bar(boroughs, scores, color=['#C76E6E', '#9475AB', '#597DBF', '#D98B5F'])
```

```
plt.xlabel('Borough', fontsize=12)
plt.ylabel('Weighted Score', fontsize=12)
plt.title('Best Option for Opening a Restaurant', fontsize=14)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)


    # Add labels to each borough's best cuisine option
plt.text(0, scores[0], "Latin", ha='center', va='bottom', fontsize=10,␣
 ↪weight='bold', color='black', zorder=10)
plt.text(1, scores[1], "Pizza", ha='center', va='bottom', fontsize=10,␣
 ↪weight='bold', color='black', zorder=10)
plt.text(2, scores[2], "Caribbean", ha='center', va='bottom', fontsize=10,␣
 ↪weight='bold', color='black', zorder=10)
plt.text(3, scores[3], "Japanese", ha='center', va='bottom', fontsize=10,␣
 ↪weight='bold', color='black', zorder=10)

# Adjust the layout to prevent labels from overlapping
plt.tight_layout()

# Add the best option labels
for i, option in enumerate(best_options):
    plt.text(i, scores[i], option, ha='center', va='bottom', fontsize=10,␣
 ↪weight='bold', color='white')

plt.show()
```



7 6. Conclusion:

Based on the findings due to our analysis of the new york restaurant dataset, valuable insights were established.This information provided further understanding of the restaurant market, further validating optimization opportunities for current locations, and expansion.

The following are reccomendations to target the aspects of each business objective:

1. Tailor current restaurant location elements, based on specific cuisine market trends and saturation.

- The 5 most prevelant cuisine categories were:American, Chinese, Cafe's, Pizza, and Italian.
- The 5 least preveleant cuisines categories were: Californian,Polynesian, and Chilean.
- Highly prevalent cuisine categories indicate higher competition levels.
- Less prevalent cuisines offer opportunities to fill market gaps with lower competition.
- The prevelance and saturation levels identified potential niche opportunities by analyzing prevalence of cuisine categories.

This information along with the rest of the ratings of cuisine categories grants insight into what the market trends and saturation are.

2. Implement target marketing, to capture new customers based on revealed current market success. The research unveiled that the top 3 rated restaurants in each borough are as follows:

- Brooklyn: D & Y Restaurant, Anella, Cheikh Futiyu.
- Manhattan: Murals on 54/Randolphs, Bella Napoli, Baluchis indian food.
- Staten Island: Oaxaca Deli, Liugis Dolceria, Greenleafs Grille.
- Bronx: la Potencia, El Molino, RojoZymi Bar.
- Queens:Spicy Shallot, Gal Bi Ma Eul, Takeshushi.
- This information displays what cuisine and restaurant is currently succesful in each of the various boroughs.
- This information allows us to optimize current restaurant locations based on the cuisine types and restaurants that are already successful in those areas.
- By customizing marketing and promotional efforts more towards the elements of these restaurants, existing success can be capitilized on

3. Expand restaurant location strategically based on high customer demand/ low market saturation opportunities. Based on the analysis, each borough varies differently in specific cuisine types that are highly demanded, and also in low competition.

Average rating per rating count was calculated to identify the most optimal cuisine style for expansion in each borough.

The findings are as follows: - The Bronx:0.0162 latin - Queens:0.0102 pizza - Brooklyn:0.0096 Caribbean - Manhattan:0.0077 japanese - Staten island:0.4 italian

Overall, these are the top cuisine types with the highest rating and lowest demand for each of the 5 various boroughs.

To maximize expansion success, a latin cuisine style restaurant in the Bronx is the most valid choice.

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]:
```
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]:
```python
#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-h10Ftb7xBYgmZ1o"

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()
```

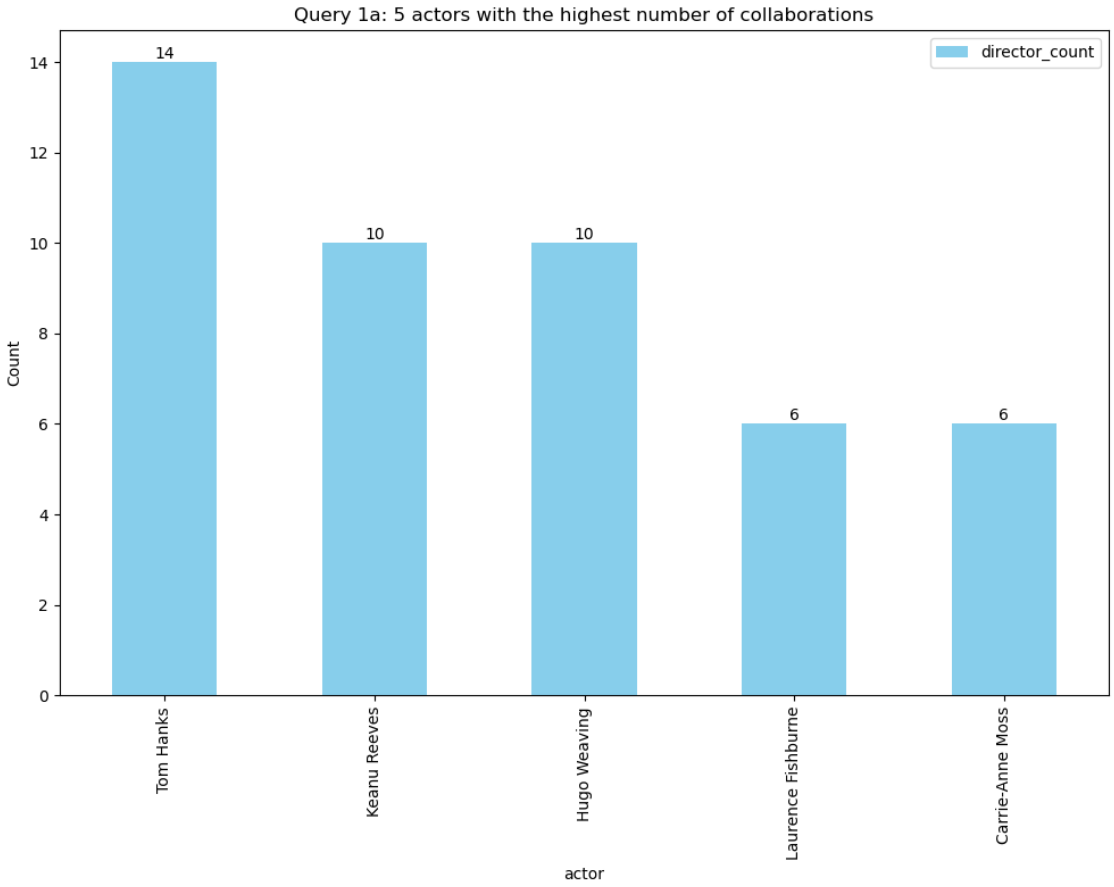
```
Query 1a: 5 actors with the highest number of collaborations
<Record actor='Tom Hanks' director_count=14>
<Record actor='Keanu Reeves' director_count=10>
<Record actor='Hugo Weaving' director_count=10>
<Record actor='Laurence Fishburne' director_count=6>
<Record actor='Carrie-Anne Moss' director_count=6>

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"
    )
]
```

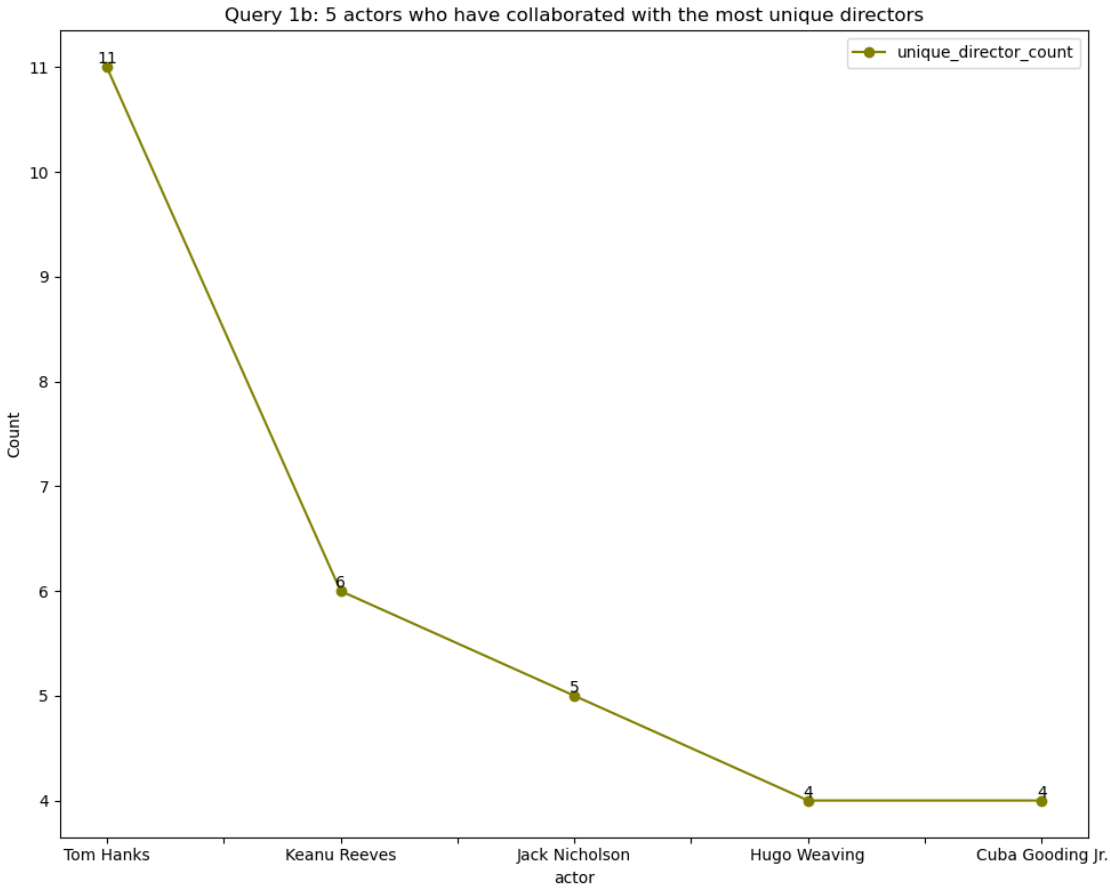
```
with driver.session() as session:
    for index, (query, comment) in enumerate(queries):
        result = session.run(query)
        print(comment)

        # 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()
```

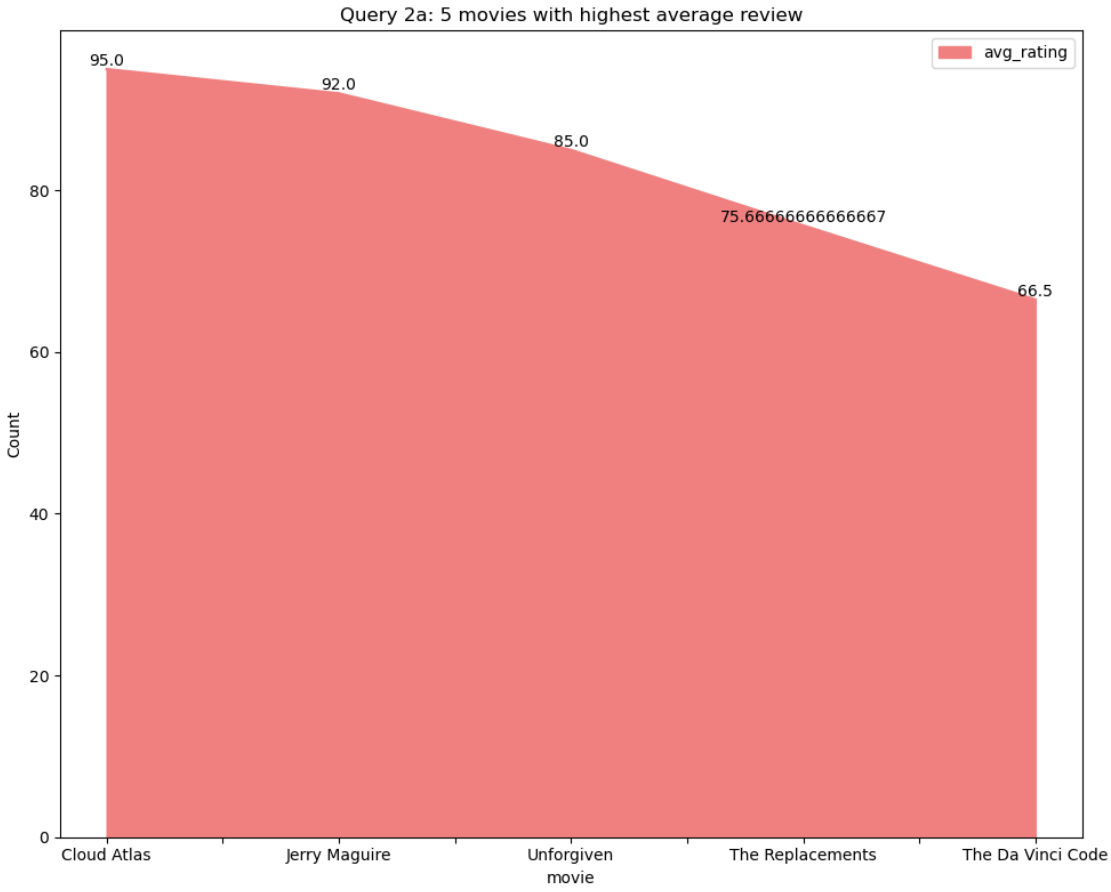
Query 1a: 5 actors with the highest number of collaborations



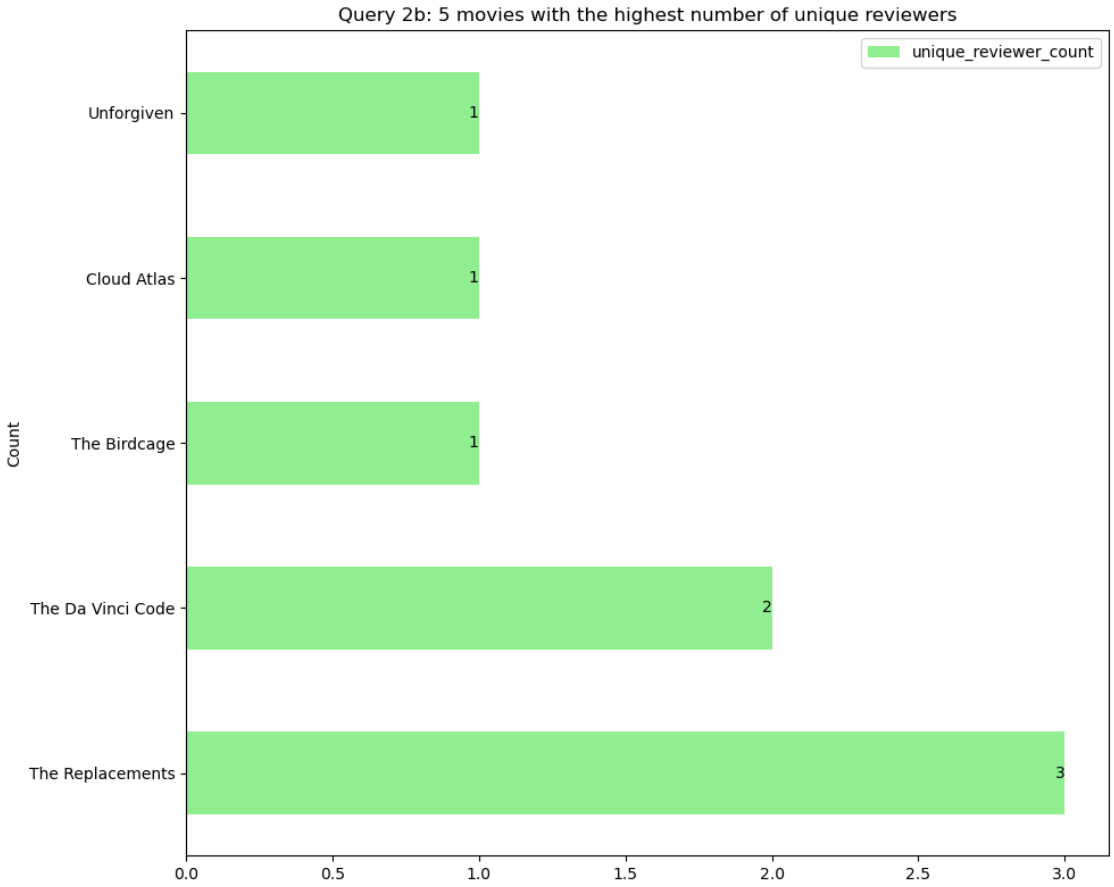
Query 1b: 5 actors who have collaborated with the most unique directors



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.