Big Data Modeling and Management Assigment - Homework 1

🍻 The Beer project 🍻

As it was shown in classes, graph databases are a natural way of navegating distinct types of data. For this first project we will be taking a graph database to analyse beer and breweries!

For reference the dataset used for this project has been extracted from kaggle, released by Evan Hallmark. Even though the author does not present metada on the origin of the data it is probably a collection of open data from places like beeradvocate.

Problem description

Explore the database via python neo4j connector and/or the graphical tool in the NEO4J webpage. Answer the questions. Submit the results by following the instructions

Questions

1. How many different countries exist in the database?
2. Most reviews:
   A. Which `Beer` has the most reviews?
   B. Which `Brewery` has the most reviews for its beers? [Hint: 5-node path]
   C. Which `Country` has the most reviews for its beers? [Hint: 5-node path]
3. Find the user/users that have the most shared reviews (reviews of the same beers) with the user CTJman?
4. Which Portuguese brewery has the most beers?
5. From those beers (the ones returned from the previous question), which has the most reviews?
6. On average how many different beer styles does each brewery produce?
7. Which brewery produces the strongest beers according to ABV? [Hint: database has NaN values]
8. If I typically enjoy a beer due to its aroma and appearance, which beer style should I try? (Justify your answer well!) [Hint: database has NaN values]
9. Using Graph Algorithms answer **one** of the following questions: [Hint: make sure to clear the graph before using it again]
   A. Which two countries are most similiar when it comes to their **top 10** most produced Beer styles?
   B. Which beer is the most influential when considering beers are conected by users who review them? [Please use limit of 1000 on beer-review-user path]]
   C. Users are connected together by their reviews to beers, taking into consideration the "overall" score they review as a weight, how many communities are formed from these relationships? How many users has the biggest community? [Please use limit of 1000 on beer-review-user path]]
10. Using Graph Algorithms answer **one** of the following questions:
    A. Which beer has the most similar reviews as the beer `Super Bock Stout` ? [Hint:inspect two subsets: with and without the beer in question]
    B. Which user is the most influential when it comes to reviews made?

11. If you had to pick 3 beers to recommend using only this database, which would you pick and why? (Justify your answer well!) [Hint: database has NaN values]

Questions 8 to 10 are somewhat open, which means we'll also be evaluating the reasoning behind your answer. So there aren't necessarily bad results there are only wrong criteria, explanations or execution.

Groups

Groups should have 4 to 5 people. You should register your group on **moodle**.

Submission

The code used to produce the results and respective explations should be uploaded to moodle. They should have a clear reference to the group, either on the file name or on the document itself. Preferably one Jupyter notebook per group.

Delivery date: Until the **midnight of March 13**

Evaluation

This will be 20% of the final grade.
Each solution will be evaluated on 2 components: correctness of results and simplicity of the solution.
All code will go through plagiarism automated checks. Groups with the same code will undergo investigation.

Loading the Database

Be sure that you **don't have** the neo4j docker container from the classes running (you can Stop it in the desktop app or with the command " `docker stop Neo4JLab` ")

The default container does not have any data whatsoever, we will have to load a database into our docker image:

- Download and unzip the `Neo4JHWData` file provided in Moodle.
- Copy the path of the `Neo4JHWData` folder of the unziped file, e.g. `C:/Users/nunoa/Desktop/Aulas/Big Data Management and Modelling/2023/HW1/Neo4JHWData/data` .
- Download and unzip the `Neo4JPlugins` file provided in Moodle.
- Copy the path of the `Neo4JPlugins` folder of the unziped file, e.g. `C:/Users/nunoa/Desktop/Aulas/Big Data Management and Modelling/2023/Neo4Jplugins` .
- Change the code bellow accordingly. As you might have noticed, you do not have a user called `nunoa` , please use the appropriate path that you got from the previous step. Be sure that you have a neo4j docker container running: \

```
docker run --name Neo4JHW -p 7474:7474 -p 7687:7687 -d -v "C:\Users\ABeatriz\Desktop\Mestrado\1ºANO DS-BA\2SEMESTRE\BDMM\HW1\Neo4JPlugins":/plugins -v "C:\Users\ABeatriz\Desktop\Mestrado\1ºANO DS-BA\2SEMESTRE\BDMM\HW1\Neo4JHWData\data":/data --env NEO4J_AUTH=neo4j/test --env NEO4J_dbms_connector_https_advertised__address="localhost:7473" --env
```

```
NEO4J_dbms_connector_http_advertised__address="localhost:7474" --env
NEO4J_dbms_connector_bolt_advertised__address="localhost:7687" --env
NEO4J_dbms_security_procedures_unrestricted=gds.* --env
NEO4J_dbms_security_procedures_allowlist=gds.* neo4j:4.4.5
```

- Since Neo4j is trying to recognize a new database folder, this might take a bit (let's say 3 minutes), so don't worry.

In [1]:
```
from neo4j import GraphDatabase
from pprint import pprint
```

In [2]:
```
NEO4J_URI="neo4j://localhost:7687"
NEO4J_USERNAME="neo4j"
NEO4J_PASSWORD="test"
```

In [3]:
```
driver = GraphDatabase.driver(NEO4J_URI, auth=(NEO4J_USERNAME, NEO4J_PASSWORD), )
```

All The Functions you'll need to run queries in Neo4J

In [4]:
```
def execute_read(driver, query):
    with driver.session(database="neo4j") as session:
        result = session.execute_read(lambda tx, query: list(tx.run(query)), query)
    return result
```

Understanding the Database

In [5]:
```
query = """
        call db.labels();
    """

result = execute_read(driver, query)

pprint(result)
```

```
[<Record label='COUNTRIES'>,
 <Record label='CITIES'>,
 <Record label='BREWERIES'>,
 <Record label='BEERS'>,
 <Record label='REVIEWS'>,
 <Record label='STYLE'>,
 <Record label='USER'>]
```

In [6]:
```
query = """
        CALL db.relationshipTypes();
    """

result = execute_read(driver, query)

pprint(result)
```

```
[<Record relationshipType='REVIEWED'>,
 <Record relationshipType='BREWED'>,
 <Record relationshipType='IN'>,
 <Record relationshipType='HAS_STYLE'>,
 <Record relationshipType='POSTED'>]
```

Submission

GROUP NUMBER:

1

GROUP MEMBERS:

STUDENT NUMBER	STUDENT NAME
Ana Beatriz Esteves	20191209
Ana Gonçalinho	20191210
Wai Kong Ng	20221384
Marta Antunes	20221094
Johnas Chami	20220723

1. How many different countries exist in the database?

In [7]:
```
countries = """
        MATCH (c:COUNTRIES)
        RETURN COUNT(c) AS NumCountries
"""

result = execute_read(driver, countries)

pprint(result)
```

```
[<Record NumCountries=200>]
```

2. Most reviews:

A) Which `Beer` has the most reviews?

In [8]:
```
mostReviewedBeer = """
        MATCH (b:BEERS)-[:REVIEWED]->(r:REVIEWS)
        RETURN b.name AS MostReviewedBeer, COUNT(r) AS NumReviews
        ORDER BY NumReviews DESC
        LIMIT 1

"""
result = execute_read(driver, mostReviewedBeer)

pprint(result)
```

```
[<Record MostReviewedBeer='IPA' NumReviews=31387>]
```

B) Which `Brewery` has the most reviews for its beers?

In [9]:
```
mostReviewedBrewery_forBeers = """
        MATCH (bw:BREWERIES)-[:BREWED]->(:BEERS)-[:REVIEWED]->(r:REVIEWS)
        RETURN bw.name AS MostReviewedBrewery_forBeers, COUNT(r) AS NumReviews
        ORDER BY NumReviews DESC
        LIMIT 1

"""
result = execute_read(driver, mostReviewedBrewery_forBeers)

pprint(result)
```

```
[<Record MostReviewedBrewery_forBeers='Sierra Nevada Brewing Co.' NumReviews=175161>]
```

C) Which `Country` has the most reviews for its beers?

In [10]:
```
mostReviewedCountry_forBeers = """
    MATCH (c:COUNTRIES)<-[:IN]-(:CITIES)<-[:IN]-(:BREWERIES)-[:BREWED]->(:BEERS)-[:REVIE
    RETURN c.name AS CountryName, COUNT(r) AS ReviewCount
    ORDER BY ReviewCount DESC
    LIMIT 1
"""
result = execute_read(driver, mostReviewedCountry_forBeers)

pprint(result)
```

```
[<Record CountryName='US' ReviewCount=7675804>]
```

3. Find the user/users that have the most shared reviews (reviews of the same beers) with the user CTJman?

In [11]:
```
sharedReviews = """
        MATCH (u1:USER {name: 'CTJman'})<-[:POSTED]-(r1:REVIEWS)<-[:REVIEWED]-(b:BEERS)-
        WHERE NOT u2.name = 'CTJman'
        WITH u2, COUNT(DISTINCT b) AS NumReviews
        RETURN u2.name AS UsersName, NumReviews AS SharedReviews
        ORDER BY NumReviews DESC
        LIMIT 10

"""

result = execute_read(driver, sharedReviews)

pprint(result)
```

```
[<Record UsersName='acurtis' SharedReviews=1428>,
 <Record UsersName='Texasfan549' SharedReviews=1257>,
 <Record UsersName='kjkinsey' SharedReviews=1205>,
 <Record UsersName='oline73' SharedReviews=1191>,
 <Record UsersName='chippo33' SharedReviews=1161>,
 <Record UsersName='mendvicdog' SharedReviews=1156>,
 <Record UsersName='spycow' SharedReviews=1142>,
 <Record UsersName='djrn2' SharedReviews=1122>,
 <Record UsersName='duceswild' SharedReviews=1081>,
 <Record UsersName='SocalKicks' SharedReviews=1077>]
```

4. Which Portuguese brewery has the most beers?

In [12]:
```
portugueseBreweryMostBeers = """
        MATCH (bw:BREWERIES)-[:IN]->(:CITIES)-[:IN]->(c:COUNTRIES)
        WHERE c.name = 'PT'
        MATCH (bw)-[:BREWED]->(b:BEERS)
        WITH bw, COUNT(b) AS BeerCount
        ORDER BY BeerCount DESC
        LIMIT 1
        RETURN bw.name AS BreweryName, BeerCount

"""
result = execute_read(driver, portugueseBreweryMostBeers)

pprint(result)
```

```
[<Record BreweryName='Dois Corvos Cervejeira' BeerCount=40>]
```

5. From those beers (the ones produced in the brewery from the previous question), which has the most reviews?

In [13]:
```
mostReviewedBeerPT = """
```

```
        MATCH (bw:BREWERIES)-[:BREWED]->(b:BEERS)-[:REVIEWED]->(r:REVIEWS)
        WHERE bw.name = 'Dois Corvos Cervejeira'
        RETURN b.name AS MostReviewedBeer, COUNT(r) AS NumReviews
        ORDER BY NumReviews DESC
        LIMIT 1
"""
result = execute_read(driver, mostReviewedBeerPT)

pprint(result)
```

```
[<Record MostReviewedBeer='Finisterra' NumReviews=10>]
```

6. On average how many different beer styles does each brewery produce?

In [14]:
```
avgBeerStylesBrewery = """
    MATCH (bw:BREWERIES)-[:BREWED]->(:BEERS)-[:HAS_STYLE]->(s:STYLE)
    WITH bw, COUNT(DISTINCT s) AS NumStyles
    RETURN ROUND(AVG(NumStyles),2) AS AvgBeerStylesBrewery
    """

result = execute_read(driver, avgBeerStylesBrewery)

pprint(result)
```

```
[<Record AvgBeerStylesBrewery=10.6>]
```

7. Which brewery produces the strongest beers according to ABV?

In [15]:
```
highestABV_BreweryBeers = """
    MATCH (bw:BREWERIES)-[:BREWED]->(b:BEERS)
    WHERE NOT b.abv = 'None' AND NOT b.abv = '' AND toFloat(b.abv) <> 100.0
    WITH bw.name AS Brewery, b.name AS Beer, toFloat(b.abv) AS ABV
    RETURN Brewery, Beer, MAX(ABV) AS highestABV
    ORDER BY highestABV DESC
    LIMIT 5
"""

result = execute_read(driver, highestABV_BreweryBeers)

pprint(result)
```

```
[<Record Brewery='Morgan Territory Brewing' Beer='Dark Reckoning' highestABV=80.0>,
 <Record Brewery='Brewmeister' Beer='Snake Venom' highestABV=67.5>,
 <Record Brewery='Redline Brewhouse' Beer='series 3' highestABV=66.0>,
 <Record Brewery='Brewmeister' Beer='Armageddon' highestABV=65.0>,
 <Record Brewery="Brouwerij 't Koelschip" Beer='Start The Future' highestABV=60.0>]
```

8. If I typically enjoy a beer due to its aroma and appearance, which beer style should I try?

In [16]:
```
BeerAppreciationScore= """
    MATCH (s:STYLE)<-[:HAS_STYLE]-(b:BEERS)-[:REVIEWED]->(r:REVIEWS)
    WHERE NOT r.smell = 'nan' AND NOT r.look = 'nan'
    WITH s.name AS BeerStyle, AVG(toFloat(r.smell)) AS BeerSmell, AVG(toFloat(r.look)) A
    RETURN BeerStyle, ROUND((BeerSmell + BeerAppearance),2) AS BeerAppreciationScore, Nu
    ORDER BY BeerAppreciationScore DESC
    LIMIT 5
    """

result = execute_read(driver, BeerAppreciationScore)

pprint(result)

# Since the person enjoys a beer depending on its aroma and appearance and understood it
```

```
# beer charateriscticts, the more the person will enjoy the drink. We started by calculat
# drink, because the same drink can have a lot of reviews and in those reviews different
# appearance. After computing the average score for every drink we calculated the sum of
# average score of appearance in order to identify the drinks with higher score in these
# Finally we just had to order by that score in descending order to obtain the highest s
# and appearance.
```

```
[<Record BeerStyle='New England IPA' BeerAppreciationScore=8.8 NumReviews=110696>,
 <Record BeerStyle='American Imperial Stout' BeerAppreciationScore=8.55 NumReviews=35219
3>,
 <Record BeerStyle='Belgian Gueuze' BeerAppreciationScore=8.41 NumReviews=20237>,
 <Record BeerStyle='American Imperial Porter' BeerAppreciationScore=8.39 NumReviews=3064
3>,
 <Record BeerStyle='Russian Imperial Stout' BeerAppreciationScore=8.38 NumReviews=147535
>]
```

9. Using Graph Algorithms answer **one** of the following questions:

1. Which two countries are most similiar when it comes to their **top 10** most produced Beer styles?

In [17]:
```
# Step 0 - Clear graph

try:
    query = """
            CALL gds.graph.drop('similar_countries') YIELD graphName;
        """

    result = execute_read(driver, query)

    pprint(result)
except Exception as e:
    pprint(e)
```

```
[<Record graphName='similar_countries'>]
```

In [18]:
```
# Step 1 - Create an appropriate graph

try:
    query = """
        CALL gds.graph.project.cypher(
            'similar_countries',
            "MATCH (n) where head(Labels(n))='COUNTRIES' or head(Labels(n))='STYLE' RETU
            "MATCH (c:COUNTRIES)<-[:IN]-(:CITIES)<-[:IN]-(:BREWERIES)-[:BREWED]->(b:BEER
                WITH c, COUNT(s.name) AS freq
                ORDER BY freq DESC
                LIMIT 10
                MATCH (c2:COUNTRIES)<-[:IN]-(:CITIES)<-[:IN]-(:BREWERIES)-[:BREWED]->(b:
                WITH collect(distinct id(c)) as old, c2, s2, COUNT(s2.name) AS freq2
                ORDER BY freq2 DESC
                WHERE any(x IN old WHERE x = id(c2))
                return id(c2) AS source, id(s2) AS target
                LIMIT 1000"
            )
        """

    result = execute_read(driver, query)

    pprint(result)
except Exception as e:
    pprint(e)
```

```
[<Record nodeQuery="MATCH (n) where head(Labels(n))='COUNTRIES' or head(Labels(n))='STYL
E' RETURN id(n) AS id" relationshipQuery='MATCH (c:COUNTRIES)<-[:IN]-(:CITIES)<-[:IN]-(:
BREWERIES)-[:BREWED]->(b:BEERS)-[:HAS_STYLE]->(s:STYLE)\n                WITH c, COUNT
```

```
(s.name) AS freq\n                ORDER BY freq DESC\n                LIMIT 10\n
        MATCH (c2:COUNTRIES)<-[:IN]-(:CITIES)<-[:IN]-(:BREWERIES)-[:BREWED]->(b:BEERS)-
[:HAS_STYLE]->(s2:STYLE)\n                WITH collect(distinct id(c)) as old, c2, s2, C
OUNT(s2.name) AS freq2\n                ORDER BY freq2 DESC\n                WHERE any(x
IN old WHERE x = id(c2))\n                return id(c2) AS source, id(s2) AS target\n
            LIMIT 1000' graphName='similar_countries' nodeCount=313 relationshipCount=1
000 projectMillis=57964>]
```

In [19]:
```
# Step 2 - Run the algorithm

try:
    query = """
        CALL gds.nodeSimilarity.stream('similar_countries')
            YIELD node1, node2, similarity
            WITH gds.util.asNode(node1).name AS Country1, gds.util.asNode(node2).name AS
            RETURN Country1, Country2, similarity
            ORDER BY similarity DESCENDING
            LIMIT 1
        """

    result = execute_read(driver, query)

    pprint(result)
except Exception as e:
    pprint(e)
```

```
[<Record Country1='US' Country2='CA' similarity=0.990909090909091>]
```

10. Using Graph Algorithms answer **one** of the following questions:

2. Which user is the most influential when it comes to reviews made?

In [20]:
```
# Step 0 - Clear graph, graph names need to be unique

try:
    query = """
            CALL gds.graph.drop('influential_user') YIELD graphName;
        """

    result = execute_read(driver, query)

    pprint(result)
except Exception as e:
    pprint(e)
```

```
[<Record graphName='influential_user'>]
```

In [21]:
```
# Step 1 - Create an appropriate graph

try:
    query = """
        CALL gds.graph.project.cypher(
            'influential_user',
            'MATCH (n:USER) RETURN id(n) AS id',
            'MATCH (u1:USER)<-[:POSTED]-(:REVIEWS)<-[:REVIEWED]-(b:BEERS)-[:REVIEWED]->(
             RETURN id(u1) AS source, id(u2) AS target
             LIMIT 1000'
            )
        """

    result = execute_read(driver, query)

    pprint(result)
except Exception as e:
    pprint(e)
```

```
[<Record nodeQuery='MATCH (n:USER) RETURN id(n) AS id' relationshipQuery='MATCH (u1:USE
R)<-[:POSTED]-(:REVIEWS)<-[:REVIEWED]-(b:BEERS)-[:REVIEWED]->(:REVIEWS)-[:POSTED]->(u2:U
SER)\n             RETURN id(u1) AS source, id(u2) AS target\n             LIMIT 1000' g
raphName='influential_user' nodeCount=164935 relationshipCount=1000 projectMillis=483>]
```

In [22]:
```
# Step 2 - Run the algorithm

try:
    query = """
        CALL gds.pageRank.stream('influential_user')
            YIELD nodeId, score
            RETURN gds.util.asNode(nodeId).name AS Username, ROUND(score,2) AS UserScore
            ORDER BY UserScore DESC
            LIMIT 1
        """

    result = execute_read(driver, query)

    pprint(result)
except Exception as e:
    pprint(e)
```

```
[<Record Username='bluejacket74' UserScore=120.51>]
```

11. If you had to pick 3 beers to recommend using only this database, which would you pick and why?

In [23]:
```
# Excellent Portuguese Beers
# Beers with a rating of four or higher are considered "excellent" here.
# We will infer that among those great Portuguese beers, those with a bigger number of r
# will be more consistently excellent for various tasters.
# The concept behind this is that although Portugal is primarily recognized for its wine
# we wish to locate and recommend excellent beers from Portugal.

excellentPortuguese_Beers = """
     MATCH (c:COUNTRIES)<-[]-(:CITIES)<-[]-(:BREWERIES)-[]->(b:BEERS)-[:REVIEWED]->(r:RE
     WHERE r.score <> 'nan' AND c.name ='PT'
         WITH b.name AS BeerName, ROUND(AVG(toFloat(r.score)),1) AS AvgScore, COUNT(r) A
     WHERE AvgScore > 4
     RETURN BeerName, AvgScore, NumReviews
     ORDER BY NumReviews DESC
     LIMIT 3
"""
result = execute_read(driver, excellentPortuguese_Beers)

pprint(result)
```

```
[<Record BeerName='Voragem' AvgScore=4.1 NumReviews=7>,
 <Record BeerName='Passarola IPA' AvgScore=4.1 NumReviews=7>,
 <Record BeerName='Mean Sardine / De Molen Ginja Ninja' AvgScore=4.2 NumReviews=4>]
```

In [24]:
```
# Popular Portuguse Beers
# We searched for the greatest beers in Portugal, but we found that there weren't enough
# Hence, we can also suggest the most often consumed beers in Portugal, regardless of qu
# only so that he or she can sample the common beer that the majority of Portuguese peop
# (again, the reviews are only a proxy, as we believe this review website is not used by
# considering the number of reviews for Super Bock, arguably the most famous beer in Por

popularPortuguese_Beers = """
     MATCH (c:COUNTRIES)<-[:IN]-(:CITIES)<-[:IN]-(:BREWERIES)-[:BREWED]->(b:BEERS)-[:REV
     WHERE r.score <> 'nan' AND c.name ='PT'
     RETURN b.name AS BeerName, ROUND(AVG(toFloat(r.score)),1) AS AvgScore, COUNT(r) AS
     ORDER BY NumReviews DESC
     LIMIT 3
"""
```

```
result = execute_read(driver, popularPortuguese_Beers)

pprint(result)
```

```
[<Record BeerName='Super Bock' AvgScore=2.8 NumReviews=391>,
 <Record BeerName='Sagres Cerveja' AvgScore=2.8 NumReviews=279>,
 <Record BeerName='Super Bock Stout' AvgScore=3.1 NumReviews=82>]
```

In [25]:

```
#The Great Unknowns
# Using the quantity of reviews as a proxy, we wish to try to identify those beers that
# to the database's users yet have a high score.
# Subjectively, the range of 200 to 500 reviews was chosen while taking into account the
# of well-liked beers.
# 2) Less than 500 reviews would suggest not many individuals have tried the beer as man
# if not tens of thousands, of evaluations, however we believe the average score from 20

greatUnknown_Beers = """
     MATCH (b:BEERS)-[:REVIEWED]->(r:REVIEWS)
     WHERE r.overall <> 'nan'
     WITH b.name AS BeerName, ROUND(AVG(toFloat(r.score)),2) AS AvgScore, COUNT(r) AS Nu
     WHERE AvgScore > 4 AND NumReviews >= 200 AND NumReviews <= 500
     RETURN BeerName, AvgScore, NumReviews
     ORDER BY AvgScore DESC
     LIMIT 3
"""
result = execute_read(driver, greatUnknown_Beers)

pprint(result)
```

```
[<Record BeerName='Kentucky Brunch Brand Stout' AvgScore=4.84 NumReviews=434>,
 <Record BeerName='Drie Fonteinen Zenne Y Frontera' AvgScore=4.74 NumReviews=250>,
 <Record BeerName='King JJJuliusss' AvgScore=4.73 NumReviews=403>]
```