Proyecto de Fin de Curso: To Be or Not To Be (A Graph)

Bioing. Juan Manuel Perez *Departamento del Agua*
Regional Litoral Norte, Universidad de la República
Salto, Uruguay
juamanuelperez1986@gmail.com

Resumen

En este proyecto se diseñó una base de datos de grafo para representar la obra completa de Shakespeare que se encontraba alojada en un archivo tipo csv. Se implementó en Neo4j, se probaron consultas básicas, se usó la biblioteca Graph Data Science Library (GDS) de Neo para análisisde centralidad y rango y se usó la biblioteca nltk de Procesamiento de Lenguaje Natural (NLP) en Python para, usando procesamiento de lenguaje natural, obtener otro tipo de información del grafo. No se pudo usar la biblioteca APOC por dificultades con los pluggins.

I. Introducción a la plantilla

En este proyecto se busca modelar y armar una base de datos de grafos usando Neo4j que contenga todas las obras de Shakespeare. Una vez hecho esto, se propone utilizar la librería Graph Data Science (GDC) para aplicar distintos algoritmos para analizar el grafo, como centralidad y detección de comunidades. También se plantea la posibilidad de utilizar las herramientas para el procesamiento y análisis del lenguaje natural (NLP) para determinar sentimientos predominantes o temas comunes en los diálogos.

Se espera lograr una estructura de grafos que permita efectuar eficazmente consultas del tipo:

- Visualizar las relaciones entre actos, escenas y obra, según obra y para un acto en particular.
- ¿Qué personajes hablan en un acto específico de una obra?
- ¿En qué cantidad de líneas habla un personaje en un acto en una obra?
- ¿Visualizar cómo se relacionan personajes en un acto de una obra?
- ¿Listar parejas de personajes que se relacionan en una misma escena?
- Verificar cuáles son los nodos de mayor rango o importancia en el marco del grafo que armemos, tratar de determinar cuáles son los nodos más influyentes en nuestro grafo.
- Verificar cuáles son los nodos de mayor rango o importancia para una obra en particular.
- Verificar cuáles son los nodos centrales.
- Verificar cuáles son los nodos centrales de una obra en particular.
- Probar técnicas de análisis de sentimiento usando NLP.

Los resultados obtenidos han sido satisfactorios, se han podido responder las consultas planteadas eficazmente. Queda pendiente mayor exploración de uso de NLP para análisis de sentimiento.

En este documento se encontrarán con un apartado en el que se describe un trabajo de características similares a este que motivó el desarrollo del presente proyecto, el desarrollo central que describe el proceso de diseño, de implementación y las dificultades presentadas, las pruebas hechas y sus resultados y, finalmente las conclusiones y propuestas a futuro.

II. Trabajos relacionados

Se tomó como referencia y guía inicial para el desarrollo del presente trabajo, el trabajo “Network Analysis of Shakespeare's plays (Bruggen Blog, 2021)”. En este trabajo se propone una estructura de base de datos de grafo para la Obra de Shakespeare y se hacen consultas de prueba para analizar brevemente la base de datos y para explicar el uso de herramientas de Neo4j y de algunas de sus bibliotecas.

III. Diseño e implementación de la Base de datos

Para la estructuración de la base de datos se siguieron los siguientes pasos:

- Se creó un nodo Line por cada línea del csv que se usó como fuente de datos. Para cada uno de estos nodos se crearon atributos, uno por cada columna del csv.
- A partir del atributo Player de cada nodo Line, se crearon los nodos Player (personajes) y una relación entre Player y Line. Esta relación permite asociar cada personaje a una línea.
- Se separó al atributo ActSceneLine en atributos independientes, Act, Scene, Line.
- Se crean nodos Scene, únicos para cada obra y acto se crea una relación que nodos línea es parte de una escena.
- Se crean nodos acto. Cada acto es único para cada obra. Se crea una relación que indica que una escena pertenece a un acto.
- Entonces nos quedan líneas vinculadas a escenas (son parte de), escenas vinculadas a actos (son parte de) y actos vinculados a obras.
- Se crean relaciones que permiten vincular líneas que indican inicio de una escena, con la propia escena, y líneas que indican fin de una escena, con la misma escena. La líneas entre la de inicio y fin de una escena, son parte de la escena, por lo que se eliminan las relaciones part_of que vinculan líneas con escenas, solo quedan las end_with y start_with.
- Se crean relaciones entre nodos Player y nodos Play.
- Se crean relaciones de tres niveles entre parece de personajes que forman parte de la misma obra, acto y escena.

Los comandos utilizados para hacer los pasos indicados son:

```
load csv with headers from "Schakespeare_data.csv" as line
create (l:Line)
 set l.Dataline = toInteger(line.Dataline)
 set l.Play = line.Play
 set l.PlayerLinenumber = toInteger(line.PlayerLinenumber)
 set l.ActSceneLine = line.ActSceneLine
 set l.Player = line.Player
 set l.PlayerLine = line.PlayerLine;

create index on :Play(name);
create index on :Player(name);
create index on :Scene(name);
create index on :Act(name);
create index on :Line(PlayerLine);
create index on :Line(Dataline);
create index on :Line(Play);
create index on :Line(Act);
create index on :Line(Scene);

match (l:Line)
where l.Player is not null
with l
merge (pl:Player {name: l.Player})
create (pl)-[:ARTICULATES]->(l);

match (l:Line)
where l.ActSceneLine is not null
with l, split(l.ActSceneLine,".") as Array
set l.Act = Array[0]
set l.Scene = Array[1]
set l.Line = Array[2];
```

```
match (l:Line)
where l.ActSceneLine is not null
merge (sc:Scene {name: l.Play+" - Act "+l.Act+" - Scene "+l.Scene})
create (l)-[:PART_OF]->(sc);

match (l:Line)-->(sc:Scene)
where l.ActSceneLine is not null
merge (a:Act {name: l.Play+" - Act "+l.Act})
erge (sc)-[:PART_OF]->(a);

match (l:Line)-->(sc:Scene)-->(a:Act)
where l.ActSceneLine is not null
merge (p:Play {name: l.Play})
merge (a)-[:PART_OF]->(p);

match (l:Line)
where l.ActSceneLine is null
merge (p:Play {name: l.Play})
merge (l)-[:PART_OF]->(p);

match (l1:Line), (l2:Line)
where id(l1)>id(l2)
and l1.Play = l2.Play
and l1.Dataline = l2.Dataline + 1
create (l2)-[:FOLLOWED_BY]->(l1);

match (l:Line)-->(s:Scene)
with s, min(l.Dataline) as startline
match (l:Line)
where l.Dataline = startline
create (s)-[:STARTS_WITH]->(l);

match (l:Line)-->(s:Scene)
with s, max(l.Dataline) as endline
match (l:Line)
where l.Dataline = endline
create (s)<-[:ENDS]-(l);

match (l:Line)-[pao:PART_OF]-(sc:Scene)
delete pao;

match (pl:Player)-->(l:Line)
with pl, l
match (p:Play {name:l.Play})
merge (pl)-[pi:PLAYS_IN]->(p)
   on create set pi.nroflines=1
   on match set pi.nroflines= pi.nroflines+1;

match (p:Play)
call {
   with p
   match (pl:Player)-[pi:PLAYS_IN]->(p)
   return p.name as Play, pl.name as Players, pi.nroflines as NrOfLines
   order by NrOfLines desc
   limit 3 }
return Play, collect(Players) as TopPlayers, collect(NrOfLines) as TopPlayersLines
order by Play;
```

```
match (pl1:Player)-->(p:Play)<--(pl2:Player)
where id(pl1)>id(pl2)
merge (pl1)-[r:RELATED_TO]->(pl2)
set r.level=1;

match (pl:Player)-->(l:Line)
with pl, l
match (a:Act {name:l.Play+" - Act "+l.Act})
merge (pl)-[pi:PLAYS_IN]->(a)
    on create set pi.nroflines=1
    on match set pi.nroflines= pi.nroflines+1;

match (pl1:Player)-->(a:Act)<--(pl2:Player)
where id(pl1)>id(pl2)
merge (pl1)-[r:RELATED_TO]->(pl2)
set r.level=2;

match (pl:Player)-->(l:Line)
with pl, l
match (s:Scene {name:l.Play+" - Act "+l.Act+" - Scene "+l.Scene})
merge (pl)-[pi:PLAYS_IN]->(s)
    on create set pi.nroflines=1
    on match set pi.nroflines= pi.nroflines+1;

match (pl1:Player)-->(s:Scene)<--(pl2:Player)
where id(pl1)>id(pl2)
merge (pl1)-[r:RELATED_TO]->(pl2)
set r.level=3;
```

III. Experimentación

En este apartado se presentan las consultas que se usaron para verificar el alcance de los objetivos planteados como motivación para la implementación de la BD de Grafo en Neo4j.

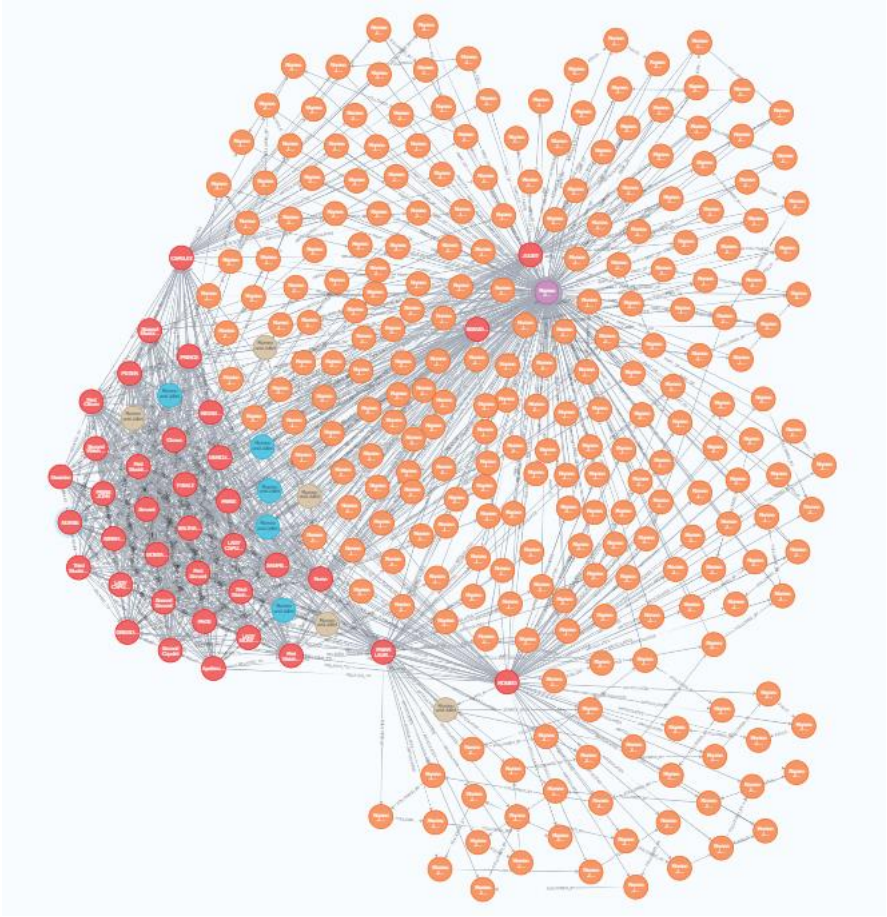
Se especifican los comandos utilizados y los resultados obtenidos par cada objetivo planteado.

- Visualizar las relaciones entre actos, escenas y obra, según obra y para un acto en particular.

Script de visualización:

```
match entirescene = (p:Play)--(a:Act)--(s:Scene)-[:STARTS_WITH]->(firstline:Line)-[:FOLLOWED_BY*]-
(lastline:Line)-[:ENDS]-(s)
where p.name contains "Romeo"
with entirescene, nodes(entirescene) as nodes
limit 1
unwind nodes as node
match (node)-[r]-(conn)
return entirescene, node, r, con
```

Resultado:



Lo que hace este comando es:
- match entirescene = (p:Play)--(a:Act)--(s:Scene)-[:STARTS_WITH]->(firstline:Line)-[:FOLLOWED_BY*]-(lastline:Line)-[:ENDS]-(s): Este match está buscando una escena completa de una obra teatral, empezando por el nodo Play, pasando por un nodo Act, un nodo Scene, un nodo Line al que se apunta con la relación STARTS_WITH desde la escena, una serie de nodos Line conectados por relaciones FOLLOWED_BY y terminando en un nodo Line que apunta a la Scene con la relación ENDS. Este patrón completo se guarda en la variable entirescene.

- where p.name contains "Romeo": Este where limita la búsqueda a escenas que pertenecen a una Play cuyo nombre contiene "Romeo".
- with entirescene, nodes(entirescene) as nodes: Este with pasa las variables entirescene y una colección de todos los nodos en entirescene (que se guarda en la variable nodes) a la siguiente parte de la consulta.
- limit 1: Esta línea limita los resultados a una única escena (la primera que se encuentra que cumple con los criterios anteriores).
- unwind nodes as node: El unwind agarra el conjunto de nodos y lo desarma en una secuencia de filas, cada una con un único nodo de la colección. Esto permite que la consulta procese cada nodo individualmente en los siguientes pasos.
- match (node)-[r]-(conn): Este match busca todas las relaciones r que conectan cada nodo node con otro nodo conn.
- return entirescene, node, r, conn: devuelve la escena completa, así como cada nodo individual, la relación que conecta ese nodo con otro, y el nodo al que está conectado.

- ¿Qué personajes hablan en un acto específico de una obra?

Script para consultar que personajes hablan en un acto específico de una obra:
(Ejemplo para el primer acto de Hamlet)

```
MATCH (p:Player)-[:PLAYS_IN]->(a:Act {name: "Hamlet - Act 1"})
RETURN DISTINCT p.name
```

Resultado:
p.name
"MARCELLUS"
"QUEEN GERTRUDE"
"BERNARDO"
"KING CLAUDIUS"
"All"
"LAERTES"
"LORD POLONIUS"
"HAMLET"
"VOLTIMAND"
"FRANCISCO"
"OPHELIA"
"HORATIO"
"Ghost"

- ¿En qué cantidad de líneas habla un personaje en un acto en una obra?

Script para consultar cantidad de participaciones en un acto en una obra para cada personaje:
(Otra vez lo hice para el primer acto de Hamlet)

```
MATCH (p:Player)-[pi:PLAYS_IN]->(a:Act {name: "Hamlet - Act 1"})
RETURN p.name, pi.nroflines
```

Resultado:

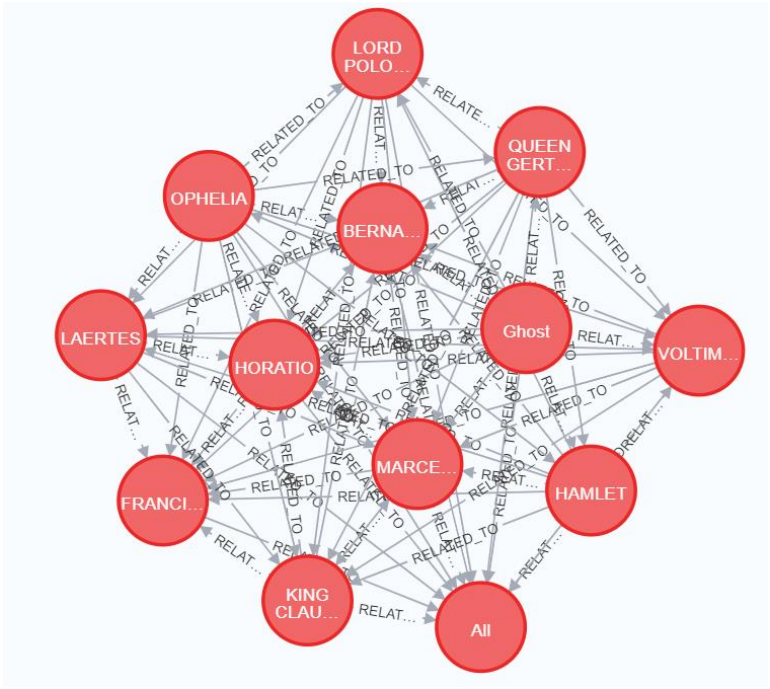
p.name	pi.nroflines
"MARCELLUS"	62
"QUEEN GERTRUDE"	10
"BERNARDO"	38
"KING CLAUDIUS"	93
"All"	1
"LAERTES"	59
"LORD POLONIUS"	72
"HAMLET"	268
"VOLTIMAND"	1
"FRANCISCO"	10
"OPHELIA"	20
"HORATIO"	190
"Ghost"	89

- ¿Visualizar cómo se relacionan personajes en un acto de una obra?

Script para visualizar como los personajes se relacionan dentro de un acto de una obra (sigo con acto primero de hamlet):

```
MATCH (p1:Player)-[r:RELATED_TO]->(p2:Player)
WHERE exists((p1)-[:PLAYS_IN]->(:Act {name: "Hamlet - Act 1"}))
AND exists((p2)-[:PLAYS_IN]->(:Act {name: "Hamlet - Act 1"}))
RETURN p1, r, p2
```

Resultado:



- Listar parejas de personajes que se relacionan en una misma escena

Script para listar parejas de personajes que se relacionan en una misma escena:

```
MATCH (p1:Player)-[r:RELATED_TO {level: 3}]->(p2:Player)
RETURN p1.name, p2.name
```

Resultado (solo los primeros de la lista):

p1.name	p2.name
"WESTMORELAND"	"KING HENRY IV"
"FALSTAFF"	"KING HENRY IV"
"FALSTAFF"	"WESTMORELAND"
"PRINCE HENRY"	"KING HENRY IV"
"PRINCE HENRY"	"FALSTAFF"
"PRINCE HENRY"	"WESTMORELAND"
"POINS"	"PRINCE HENRY"
"POINS"	"FALSTAFF"
"EARL OF WORCESTER"	"KING HENRY IV"
"EARL OF WORCESTER"	"FALSTAFF"
"EARL OF WORCESTER"	"PRINCE HENRY"

Script para listar parejas de personajes que se relacionan dentro de una misma escena en una obra específica:

```
MATCH (p1:Player)-[r:RELATED_TO {level: 3}]->(p2:Player),
(s:Scene {name: 'Hamlet - Act 1 - Scene 1'})
WHERE (p1)-[:PLAYS_IN]->(s) AND (p2)-[:PLAYS_IN]->(s)
RETURN p1.name, p2.name
```

Resultado:

p1.name	p2.name
"FRANCISCO"	"BERNARDO"
"HORATIO"	"FRANCISCO"
"HORATIO"	"BERNARDO"
"MARCELLUS"	"BERNARDO"
"MARCELLUS"	"FRANCISCO"
"MARCELLUS"	"HORATIO"

- Verificar cuáles son los nodos de mayor rango o importancia en el marco del grafo que armemos, tratar de determinar cuáles son los nodos más influyentes en nuestro grafo.

Script para calcular, ordenar y listar rango/importancia de nodo:

```
CALL gds.pageRank.stream('myGraph')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC
LIMIT 10
```

Resultado:

Name	Score
"Messenger"	10.329905080812791
"Servant"	9.377599395763047
"All"	7.307480589164442
"Captain"	4.681323753239484
"ALL"	4.60382031253037
"First Servant"	4.251533646571126
"Lord"	4.195495903171481
"First Lord"	4.12880891410159
"First Gentleman"	3.867978466131641
"Second Gentleman"	3.867978466131641

- Verificar cuáles son los nodos centrales.

Script para calcular, ordenar y mostrar centralidad de nodos:

```
CALL gds.betweenness.stream('myGraph')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC
LIMIT 10
```

Resultado:

name	Score
"Messenger"	85388.67896209672
"Servant"	73132.15412361886
"All"	33416.71087265586
"ALL"	19683.77261175528
"First Lord"	15463.45602091321
"Captain"	12781.375741267195
"Lord"	9398.127422885495
"First Servant"	8543.645361073652
"First Citizen"	8431.386372753856
"Clown"	7888.520169831046

- Verificar cuáles son los nodos de mayor rango o importancia para una obra en particular.

Script para encontrar los 5 nodos más importantes o de mayor rango en una obra en particular:

```
MATCH (p:Play {name: 'Romeo and Juliet'})<--(pl:Player)
RETURN pl.name AS name, pl.pagerank AS pagerank
ORDER BY pagerank DESC
LIMIT 5
```

Resultado:

Name	pagerank
"Servant"	9.377599395763047
"First Servant"	4.251533646571126
"First Citizen"	3.82111944605361
"Second Servant"	3.699919664526915
"PETER"	2.4345392017383243

- Verificar cuáles son los nodos centrales de una obra en particular.

Script para encontrar los 5 nodos centrales en una obra en particular:

```
MATCH (p:Play {name: 'Romeo and Juliet'})<--(pl:Player)
RETURN pl.name AS name, pl.betweenness AS betweenness
ORDER BY betweenness DESC
LIMIT 5
```

Resultado:

Name	Betweenness
"Servant"	73132.15412361886
"First Servant"	8543.645361073652
"First Citizen"	8431.386372753856
"Second Servant"	6270.148613277098
"PETER"	2543.96872398185

Para visualizar mejor esto, me conecto desde Python y grafico los 3 scores más altos de PageRank y los 3 más altos de Betweenness para cada obra.
Script:

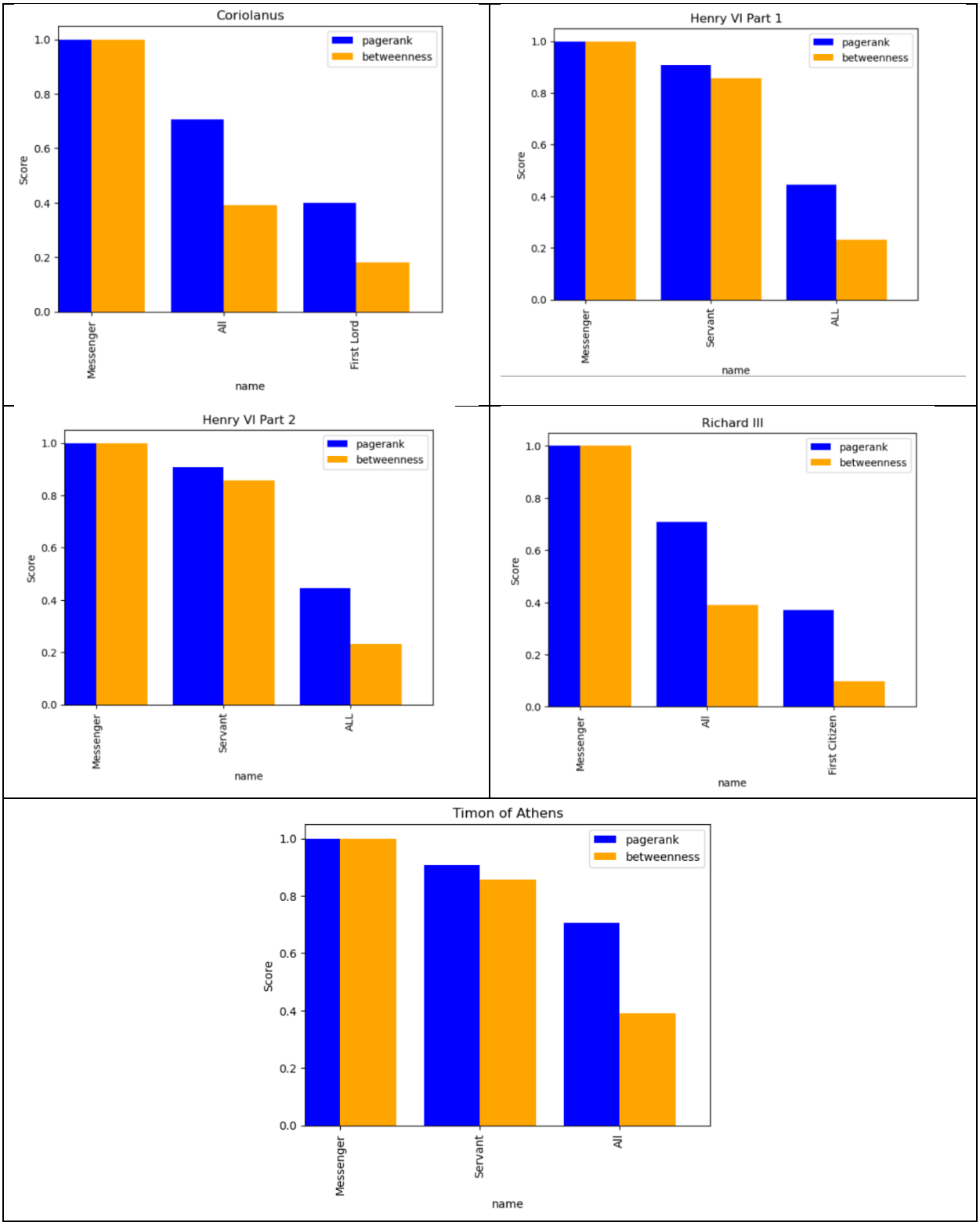
```
from py2neo import Graph
import pandas as pd

# Me conecto a la base de datos
graph = Graph("bolt://localhost:7687", auth=("neo4j", "Laacade230186$23"))
# Hago las consultas
pagerank_query = """
MATCH (p:Play)<--(pl:Player)
RETURN p.name AS play, pl.name AS name, pl.pagerank AS pagerank
ORDER BY pagerank DESC
"""

betweenness_query = """
MATCH (p:Play)<--(pl:Player)
RETURN p.name AS play, pl.name AS name, pl.betweenness AS betweenness
ORDER BY betweenness DESC
"""

pagerank_df = graph.run(pagerank_query).to_data_frame()
betweenness_df = graph.run(betweenness_query).to_data_frame()

# Group by play and get top 3 nodes
top3_pagerank = pagerank_df.groupby('play').head(3)
top3_betweenness = betweenness_df.groupby('play').head(3)
```

Creo una variable que normaliza y luego combina PageRank y Betweenness para seleccionar las 5 obras que tienen mayor score combinado y grafico para éstas, PageRank y Betweenness normalizados:

```
import matplotlib.pyplot as plt

# Junto el PageR y el BTW df
df = pd.merge(pagerank_df, betweenness_df, on=['play', 'name'])

# Normalizo PageR y BTW
df['pagerank'] = df['pagerank'] / df['pagerank'].max()
df['betweenness'] = df['betweenness'] / df['betweenness'].max()
# Calculo score combinado
df['combined_score'] = df['pagerank'] + df['betweenness']
# Agarro las 5 obras con score más alto
top5_plays = df.groupby('play')['combined_score'].sum().nlargest(5).index
# Filtro df para incluir las 5 obras con mayor score
df = df[df['play'].isin(top5_plays)]

# Ordeno por obra y agarro los 3 nodos más altos
top3 = df.groupby('play').apply(lambda x: x.nlargest(3, 'combined_score')).reset_index(drop=True)

# Grafico PageR y BTW para cada obra
for play, data in top3.groupby('play'):
    fig, ax = plt.subplots()
    data.plot(x='name', y='pagerank', kind='bar', ax=ax, color='blue', width=0.4, position=1)
    data.plot(x='name', y='betweenness', kind='bar', ax=ax, color='orange', width=0.4, position=0)
    plt.title(play)
    plt.ylabel('Score')
```

```
plt.show()
```

Resultados:



- Probar técnicas de análisis de sentimiento usando NLP.

Para las pruebas de NLP se usó la biblioteca NLTK, se probó con una frase en Inglés y se probó procesar líneas de la base de datos.

Script:

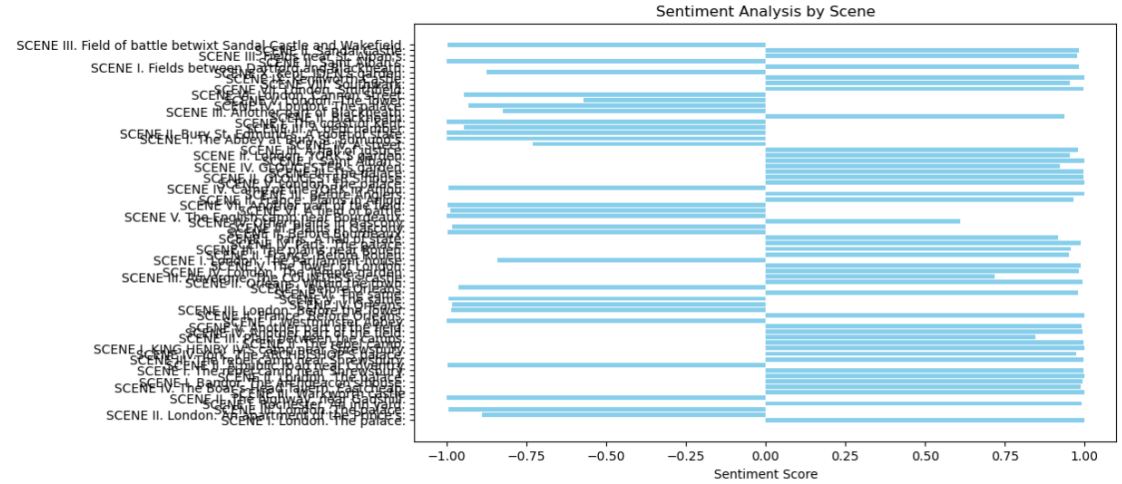
```
import neo4j
import nltk

from neo4j import GraphDatabase

uri = "neo4j://localhost:7687"
username = "neo4j"
password = "Laacade230186$23"

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


def run_query(query):
    with driver.session() as session:
        result = session.run(query)
        return [record for record in result]

query = "MATCH (n) RETURN n LIMIT 5"
result = run_query(query)
for record in result:
    print(record)

#prueba de biblioteca

from nltk.sentiment import SentimentIntensityAnalyzer

# Descargo el analizador de sentimientos VADER
nltk.download('vader_lexicon')

# Ejemplo de uso:
sia = SentimentIntensityAnalyzer()
text = "this is a good sentence." # en español no anda, al menos a mí me devolvió cualquier cosa.
print(sia.polarity_scores(text))
```

script para graficar sentimientos en las obras:

```
#función para obtener líneas de la base de datos
def get_lines(skip, limit):
    query = f"MATCH (n:Line) RETURN n.PlayerLine SKIP {skip} LIMIT {limit}"
    return run_query(query)

#función para analizar un conjunto de líneas y guardar los resultados
def analyze_sentiment(lines):
    for line in lines:
        text = line["n.PlayerLine"]
        sentiment = sia.polarity_scores(text)
        print(f"Text: {text}, Sentiment: {sentiment}")
```

```
import matplotlib.pyplot as plt
from collections import defaultdict

def get_text_by_scene(lines):
    scenes = defaultdict(list)
    current_scene = ""

    for line in lines:
        text = line["n.PlayerLine"]

        if text.startswith("SCENE") or text.startswith("ACT"):
            current_scene = text
        else:
            scenes[current_scene].append(text)

    return scenes

def get_sentiment(text):
    sia = SentimentIntensityAnalyzer()
    sentiment = sia.polarity_scores(text)
    return sentiment['compound']

def analyze_sentiment_by_scene(lines):
    scenes = get_text_by_scene(lines)
    sentiments = {}

    for scene, text_list in scenes.items():
        text = " ".join(text_list)
        sentiment = get_sentiment(text)
        sentiments[scene] = sentiment

    return sentiments

def plot_sentiments(sentiments):
    scenes = list(sentiments.keys())
    scores = list(sentiments.values())

    plt.figure(figsize=(10, 6))
    plt.barh(scenes, scores, color='skyblue')
    plt.xlabel('Sentiment Score')
    plt.title('Sentiment Analysis by Scene')
    plt.show()

lines = get_lines(0, 10000)
sentiments = analyze_sentiment_by_scene(lines)
plot_sentiments(sentiments)
```

Resultado:



El resultado no es muy útil. Debería elegir otra forma de graficar.

IV-E. Conclusiones y trabajo futuro

El proyecto implicó un gran desafío para mí como estudiante. Durante el proceso logré interactuar eficazmente con la base de datos en Neo4j y desde python utilizando bibliotecas de NLP. El diseñó final permitió hacer las consultas que se habían planteado. Se priorizó, en líneas generales, el diseño propuesto que motivó el presente trabajo ya que en los intentos por avanzar en otras direcciones, surgieron dificultades que no se pudieron salvar. No se pudo hacer un aprovechamiento claro de los algoritmos de NLP pero queda evidenciado que mejorando algunas consultas y con post-procesamiento, más allá de los hechos en esta instancia, se pueden mejorar los resultados.

REFERENCIAS

"Network Analysis of Shakespeare's plays" Bruggen Blog: Rambling thoughts about life in and around the data industry. Junio 10, 2021. https://blog.bruggen.com/2021/06/network-analysis-of-shakespeares-plays.html.