Report

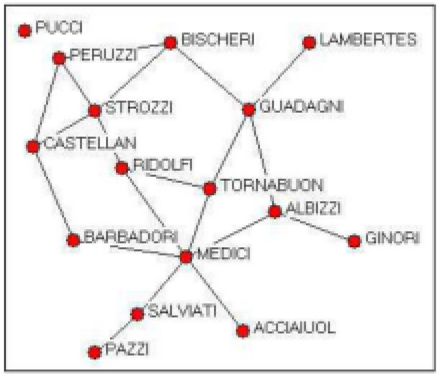
# Graph And AI CA-1

Sunil Judhistira Gauda

Student Id: 10595858

## Excercise 1

- Use Padgett's data on Florentine Families Marriage alliances
- Make Nodes Representing each Family
- Edge Denoting Connection in Marriage
- Load it in Neo4J Knowledge Graph
- Calculate Degree Centrality; Closeness Centrality; Betweenness Centrality; Eigenvector Centrality; PageRank



### Creating Nodes for Family

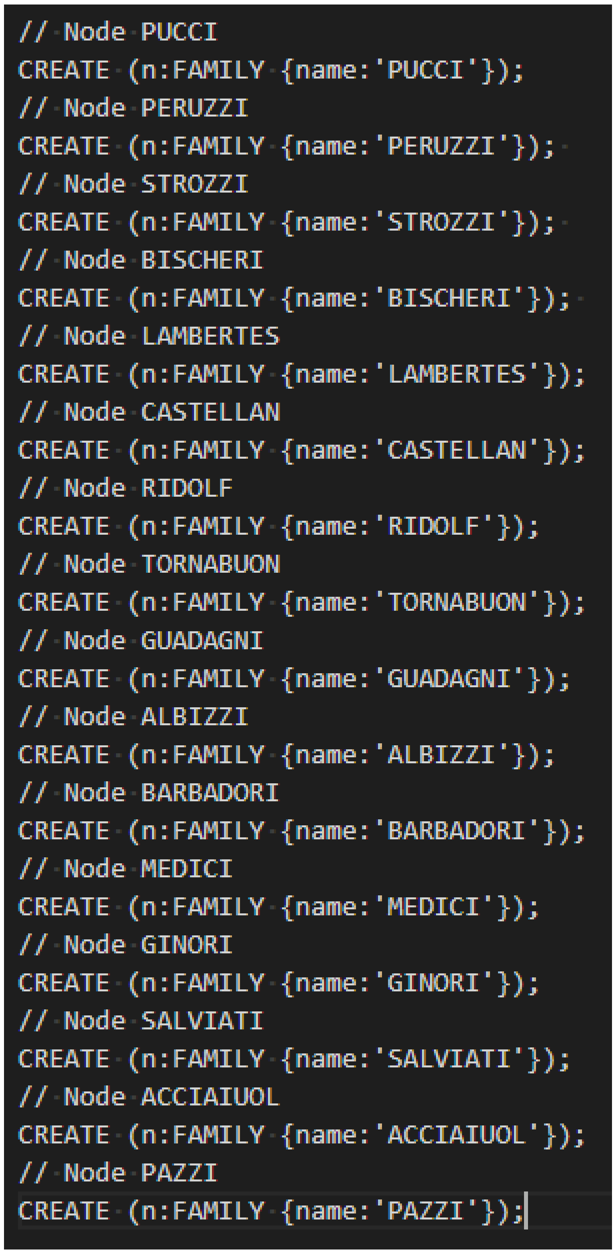
Each Node Represents a Florentine  Family, we create the node for each family one by one using the CREATE query of neo4j.

Pseudo code

// Creating the Family Nodes, the nodes will represent the family names, upon which relationships are built.

**// Node PUCCI**

**CREATE (n:<NodeName> {name:'<NodeAttribute - eg.PUCCI >'});**



```
// Node PUCCI
CREATE (n:FAMILY {name:'PUCCI'});
// Node PERUZZI
CREATE (n:FAMILY {name:'PERUZZI'});
// Node STROZZI
CREATE (n:FAMILY {name:'STROZZI'});
// Node BISCHERI
CREATE (n:FAMILY {name:'BISCHERI'});
// Node LAMBERTES
CREATE (n:FAMILY {name:'LAMBERTES'});
// Node CASTELLAN
CREATE (n:FAMILY {name:'CASTELLAN'});
// Node RIDOLF
CREATE (n:FAMILY {name:'RIDOLF'});
// Node TORNABUON
CREATE (n:FAMILY {name:'TORNABUON'});
// Node GUADAGNI
CREATE (n:FAMILY {name:'GUADAGNI'});
// Node ALBIZZI
CREATE (n:FAMILY {name:'ALBIZZI'});
// Node BARBADORI
CREATE (n:FAMILY {name:'BARBADORI'});
// Node MEDICI
CREATE (n:FAMILY {name:'MEDICI'});
// Node GINORI
CREATE (n:FAMILY {name:'GINORI'});
// Node SALVIATI
CREATE (n:FAMILY {name:'SALVIATI'});
// Node ACCIAIUOL
CREATE (n:FAMILY {name:'ACCIAIUOL'});
// Node PAZZI
CREATE (n:FAMILY {name:'PAZZI'});
```

Fig 1 :- Nodes Created in Neo4J

## Creating Relationship between families

After our Nodes are created we have a starting point of creating relations, each node represents a Family so the relations they are connected to and we can use to represent is **`MARRIAGE`.**

Pseudo code

```
// MEDICI - SALVIATI

MATCH (n1:FAMILY), (n2:FAMILY)

WHERE n1.name = 'MEDICI' AND n2.name = 'SALVIATI'

CREATE (n1)-[mrg:MARRAGE]->(n2);
```

On the above code **CREATE (n1)-[mrg:MARRAGE]->(n2)** is where we are creating the relationship, **n1** and **n2** are variables used to create this relationship which is declared **MATCH (n1:FAMILY), (n2:FAMILY)** here.

To Create the Reverse Relationship

```
// SALVIATI - MEDICI

MATCH (n1:FAMILY), (n2:FAMILY)

WHERE n1.name = 'MEDICI' AND n2.name = 'SALVIATI'

CREATE (n2)-[mrg:MARRAGE]->(n1);
```

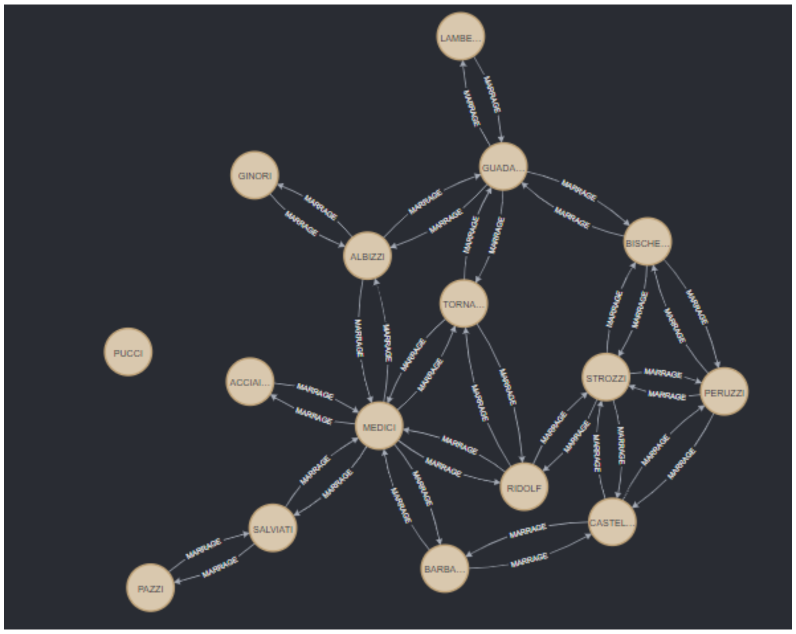


Fig 2 : - Graph Visualised after Nodes added and Relationships Added.

## Centrality Calculations

Calculations for Degree Centrality; Closeness Centrality; Betweenness Centrality; Eigenvector Centrality; PageRank.

### Degree Centrality

It represents the number of edges the node has.

Pseudo Code

```
CALL gds.degree.write('MEDICIGraph', { writeProperty: 'degree' })

YIELD centralityDistribution, nodePropertiesWritten

RETURN centralityDistribution.min AS minimumScore, centralityDistribution.mean AS meanScore, nodePropertiesWritten;
```

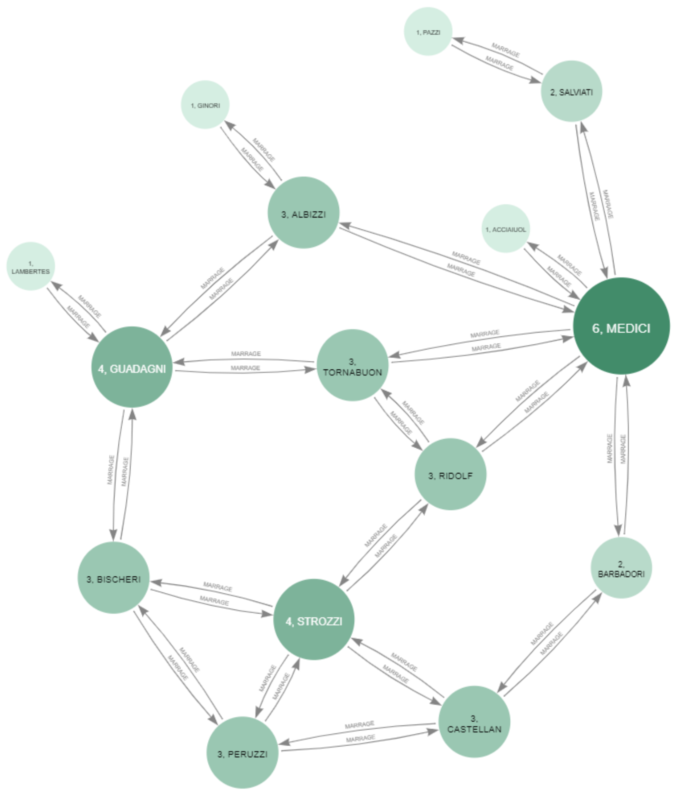


Fig 3 : - Degree Centrality Visual Representation by Size and Color - Using Neo4j Bloom

## Closeness Centrality

Closeness Centrality gives a numerical output representing how close is a particular node from other nodes.

Pseudo Code

```
//Using GDS to calculate closeness

CALL gds.alpha.closeness.write({

  nodeProjection: 'FAMILY',

  relationshipProjection: 'MARRAGE',

  writeProperty: 'centrality'

}) YIELD nodes, writeProperty;
```

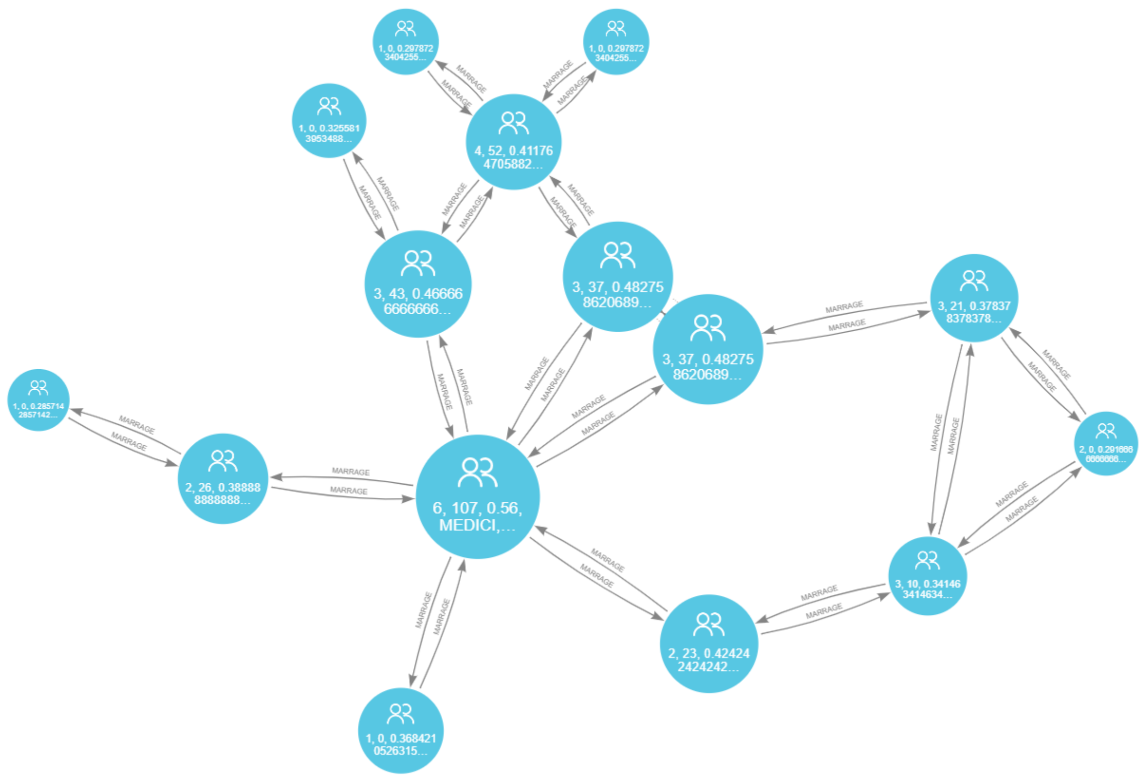


Fig 4 : - Closeness Centrality Visual Representation by Size and Color - Using Neo4j Bloom

## Betweenness Centrality

Betweenness Represents the shortest path in numerical terms

Pseudo Code

```
CALL gds.betweenness.write('MEDICIGraph', { writeProperty: 'betweenness' })

YIELD centralityDistribution, nodePropertiesWritten

RETURN centralityDistribution.min AS minimumScore, centralityDistribution.mean AS meanScore,
nodePropertiesWritten;
```

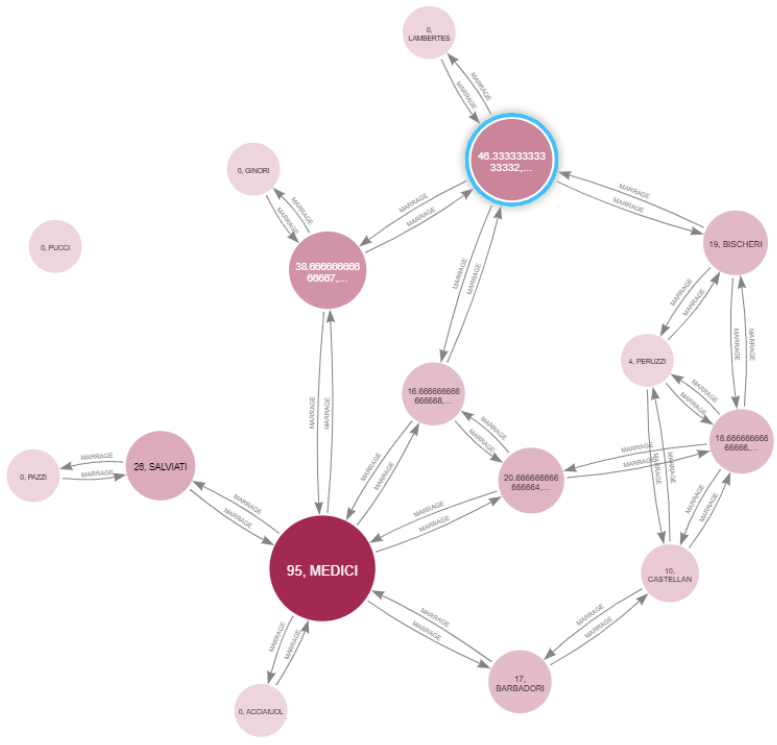


Fig 5 : - Betweenness Centrality Visual Representation by Size and Color - Using Neo4j Bloom

## Eigenvector Centrality

Eigenvector Centrality represents Influence of the node in numerical terms

Pseudo Code

```
CALL gds.eigenvector.write('MEDICIGraph', {

  maxIterations: 20,

  writeProperty: 'Eigcentrality'

})

YIELD nodePropertiesWritten, ranIterations;
```

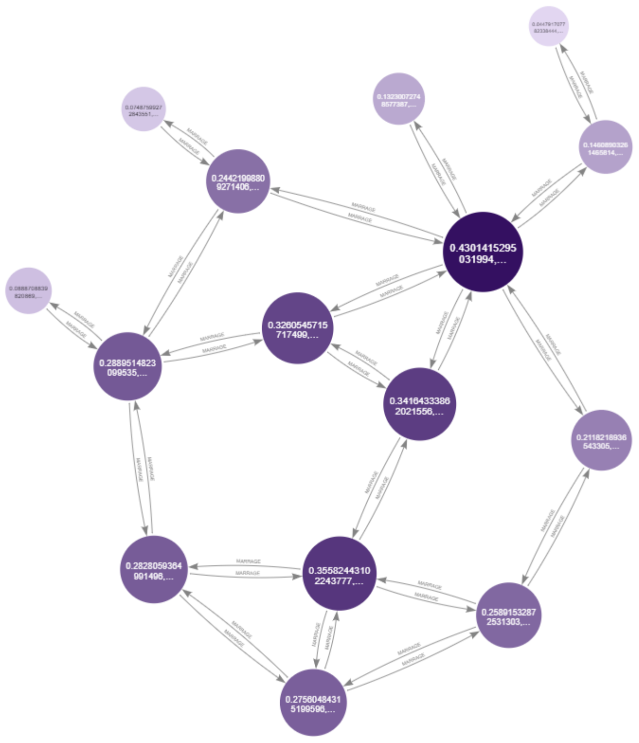


Fig 5 : - Eigenvector Centrality Visual Representation by Size and Color - Using Neo4j Bloom

## PageRank

Pagerank sets Quality of node based on importance of the node

Pseudo Code

```
CALL gds.pageRank.write('MEDICIGraph', {
  maxIterations: 20,
  dampingFactor: 0.85,
  writeProperty: 'pagerank'
})
YIELD nodePropertiesWritten, ranIterations;
```

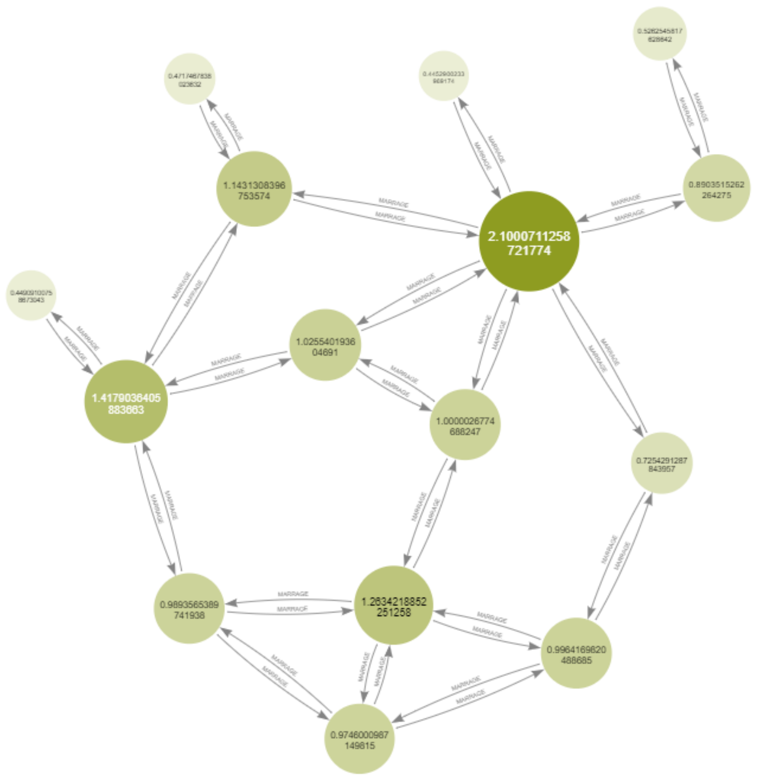


Fig 6 : -Pagerank Visual Representation by Size and Color - Using Neo4j Bloom

## Final Calculations

| ID | Name | Labels | Closeness | Eigenvector | Degree | Betweenness | Pagerank |
|---|---|---|---|---|---|---|---|
| 1 | PERUZZI | FAMILY | 0.368421052631579 | 0.275604843151996 | 3 | 4 | 0.974600098714982 |
| 2 | STROZZI | FAMILY | 0.4375 | 0.355824431022438 | 4 | 18.6666666666667 | 1.26342188522513 |
| 3 | BISCHERI | FAMILY | 0.4 | 0.28280593649915 | 3 | 19 | 0.989356538974194 |
| 4 | LAMBERTES | FAMILY | 0.325581395348837 | 0.0888708839820869 | 1 | 0 | 0.44909100758673 |
| 5 | CASTELLAN | FAMILY | 0.388888888888889 | 0.258915328725313 | 3 | 10 | 0.996416982048869 |
| 6 | RIDOLF | FAMILY | 0.5 | 0.341643338620216 | 3 | 20.6666666666667 | 1.00000267746882 |
| 7 | TORNABUON | FAMILY | 0.482758620689655 | 0.32605457157175 | 3 | 16.6666666666667 | 1.02554019360469 |
| 8 | GUADAGNI | FAMILY | 0.466666666666667 | 0.288951482309953 | 4 | 46.3333333333333 | 1.41790364058837 |
| 9 | ALBIZZI | FAMILY | 0.482758620689655 | 0.244219988092714 | 3 | 38.6666666666667 | 1.14313083967536 |
| 10 | BARBADORI | FAMILY | 0.4375 | 0.21182189365433 | 2 | 17 | 0.725429128784396 |
| 11 | MEDICI | FAMILY | 0.56 | 0.430141529503199 | 6 | 95 | 2.10007112587218 |
| 12 | GINORI | FAMILY | 0.333333333333333 | 0.0748759927284355 | 1 | 0 | 0.471746783802363 |
| 13 | SALVIATI | FAMILY | 0.388888888888889 | 0.146089032614658 | 2 | 26 | 0.890351526226428 |
| 14 | ACCIAIUOL | FAMILY | 0.368421052631579 | 0.132300727485774 | 1 | 0 | 0.445290023396917 |
| 15 | PAZZI | FAMILY | 0.285714285714286 | 0.0447917077823384 | 1 | 0 | 0.526254581762864 |

# Excercise 3

## Personal Findings on Applying CRISP-DM Methodology using Neo4J

### Business Understanding

Movies comprise of many people, which includes directors, writers, producers actors, and many more, but when choosing a movie recommendation its extremely important to provide information about the movies the person wants to see, as it will reduce the time taken to search such movies by the customer and it will increase the business value of the platform that provides the service. Using IMDBs Top 250 Movies from the link below,

https://www.kaggle.com/datasets/jillanisofttech/imdb-top-250-eng-movies-dataset

We have visualized and modeled data for using algorithms like `adamicAdar` which helps us to find similar movies to recommend to customers.

### Data Understanding

A Movie is directed in a Particular Year, in a particular Genre with specific people involved, hence we can better understand it by representing it in the form of nodes.

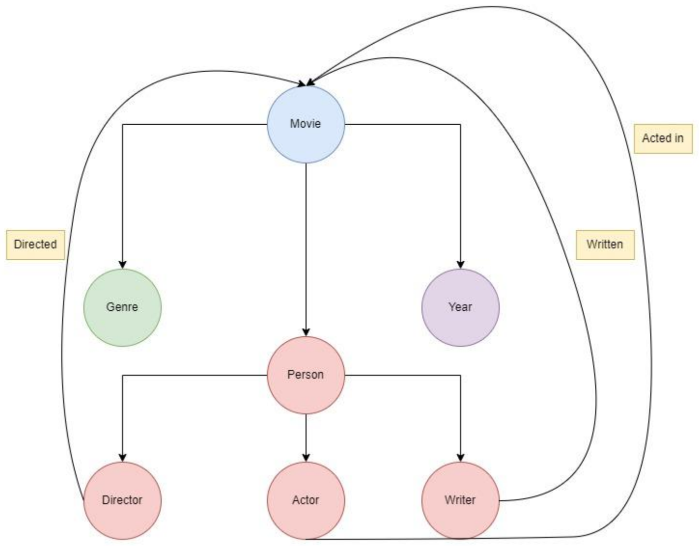


Fig 7: Relationships Defined as Nodes

As we can see above the entire movie process is related to each other by the people working on it and the genre and the year it was released on. The important aspect is the end goal, which is the rating of the movie that is to be determined on varying degrees of the likelihood of liking a movie, based on genre, person, and year, it might change from person to person.

## Data Preparation

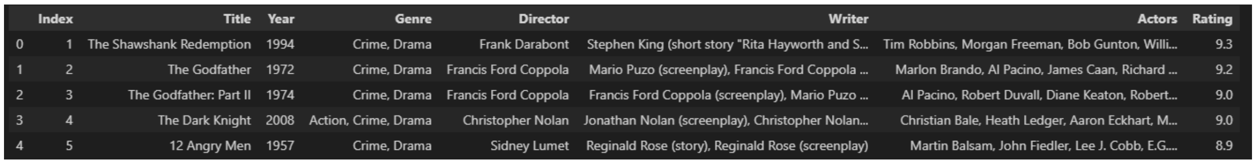
### Data Cleaning

The data we have comprises 38 Columns, we selected the following columns due to their use in our project

- **Index**: To provide us with a starting point while we write the graph to neo4 js as nodes and relationships.
- **Title**: Movies are identified by titles, and that makes it one of the most important columns in our data set, as all the relationships will branch out of it.
- **Year**: Year at which the movie was released would create a link to the type of movies released depending on the community of the people present by that time.
- **Genre**: The column defines the theme of the movie, this will also determine what kind of similar movies the person would like.
- **Director**: People have affiliation with the movie directors as they also tend to choose the movies also directed by the same director which have a similar theme, like: “Christopher Nolan”  Movie or “Martin Scorsese”  Movie.
- **Writers**: The Entire story of the movie is written by the writers of the movie and that also determines how well the movie will do. Usually, Novel adaptations have a greater chance of success because of the depth of the story.
- **Actors**: The talent in the movies determines the screenplay quality and more or less the success of that screenplay and popularity of the talent itself to gain a pace of success, a movie with great acting and a mediocre story could also do ok in the industry.
- **Rating**: Rating reflects the end product of the movie, that is customer satisfaction, which is represented in the data on a scale of 1 - 10

All the columns above was important for the project, the rest was not necessary for our specific use-case, hence it was dropped.

The End Result looks something like this.

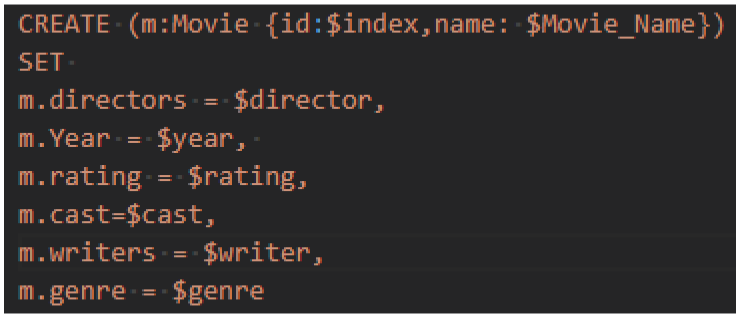


| | Index | Title | Year | Genre | Director | Writer | Actors | Rating |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | The Shawshank Redemption | 1994 | Crime, Drama | Frank Darabont | Stephen King (short story "Rita Hayworth and S... | Tim Robbins, Morgan Freeman, Bob Gunton, Willi... | 9.3 |
| 1 | 2 | The Godfather | 1972 | Crime, Drama | Francis Ford Coppola | Mario Puzo (screenplay), Francis Ford Coppola ... | Marlon Brando, Al Pacino, James Caan, Richard ... | 9.2 |
| 2 | 3 | The Godfather: Part II | 1974 | Crime, Drama | Francis Ford Coppola | Francis Ford Coppola (screenplay), Mario Puzo ... | Al Pacino, Robert Duvall, Diane Keaton, Robert... | 9.0 |
| 3 | 4 | The Dark Knight | 2008 | Action, Crime, Drama | Christopher Nolan | Jonathan Nolan (screenplay), Christopher Nolan... | Christian Bale, Heath Ledger, Aaron Eckhart, M... | 9.0 |
| 4 | 5 | 12 Angry Men | 1957 | Crime, Drama | Sidney Lumet | Reginald Rose (story), Reginald Rose (screenplay) | Martin Balsam, John Fiedler, Lee J. Cobb, E.G.... | 8.9 |

Fig 8 : - Data Set Viewing using Pandas Dataframe.

After the data is usable we need to load it as nodes in the Ne04j Graph Database.

Create Nodes



```
CREATE (m:Movie {id:$index,name: $Movie_Name})
SET
m.directors = $director,
m.Year = $year,
m.rating = $rating,
m.cast=$cast,
m.writers = $writer,
m.genre = $genre
```

Fig 9 : - Query String to create nodes

Using Py2Neo we can use the query to write to the graph database.

Creating Relations

Pseudo Code

```
//Match Query
MATCH (m:Movie)
//Where Clause
WHERE m.cast IS NOT NULL
WITH m
UNWIND split(m.cast, ',') AS actor
MERGE (p:Person {name: trim(actor)})
MERGE (p)-[r:ACTED_IN]->(m);
```

Following Relations have been Created for Our use

1. Genre
2. CREATED_ON
3. WORK_WITH
4. ACTED_IN
5. DIRECTED
6. NEXT

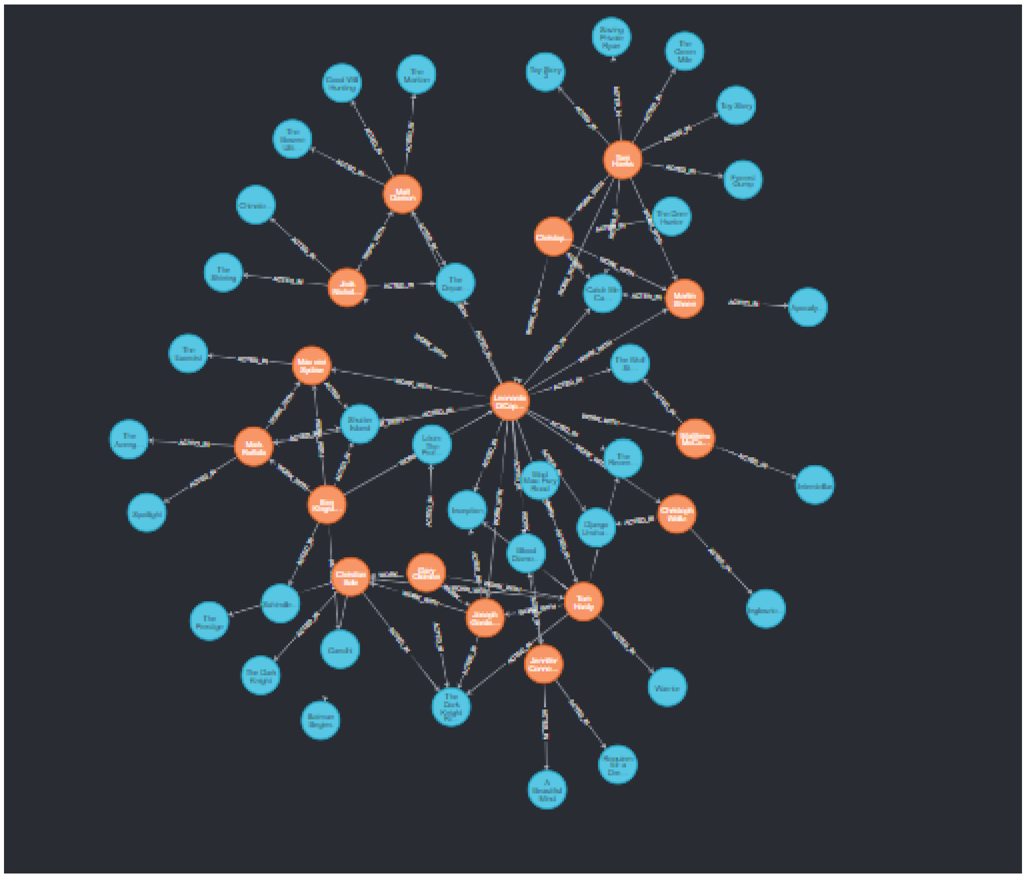


Fig 10: - Node Preview with relations

## Modeling

**Using Graph data Science Link Prediction Library Adamic Adar**

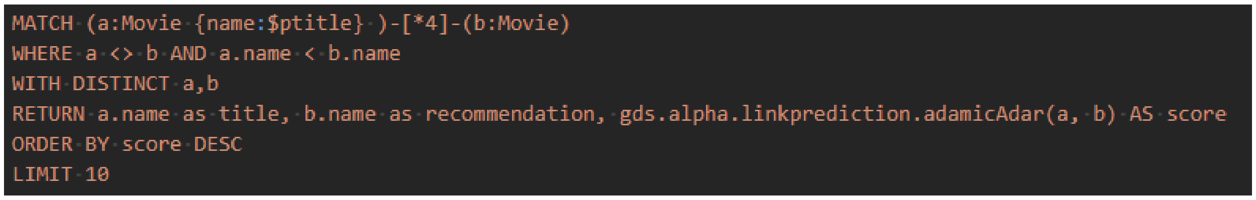
The Algorithm **Adamic Adar** uses graph links we created above and determines the most similar nodes to it, provided we give a starting point, in this case, it will be our **Movie Name** in data we can leverage the prebuilt models in neo4j to find patterns in the data.

$$A(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{\log |N(u)|}$$

Fig 10: - Adamic Adar Equation

**Adamic Adar** takes the **inverse of the logarithmic value of degree centrality**, we run the above formula using the Graph Data Science library of Neo4j, it calculates recommendation on Node Strength and provides us the well-connected nodes with a score called **Adamic Adar Index**, The index shows what is the strength of the nodes relations and provides us with a numerical value for each node. It was generally developed for social network analysis.

Below is the implementation of Adamic Adar using Neo4j.



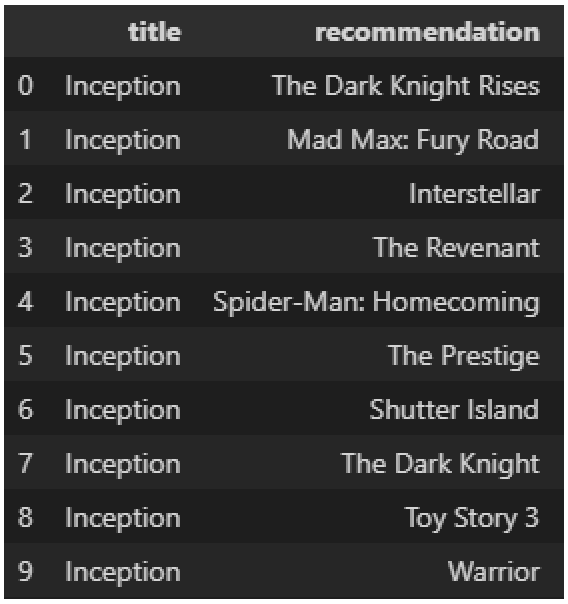
```
MATCH (a:Movie {name:$ptitle} )-[*4]-(b:Movie)
WHERE a <> b AND a.name < b.name
WITH DISTINCT a,b
RETURN a.name as title, b.name as recommendation, gds.alpha.linkprediction.adamicAdar(a, b) AS score
ORDER BY score DESC
LIMIT 10
```

Fig 11: - Adamic Adar Neo4j Query

Using the graph object from Neo4J we can query the data as below

graph.run(request_link_prediction_movie,ptitle="Inception").to_data_frame()

We receive the output as a dictionary which contains the Movie Names that are similar to the ones we provided



|   | title | recommendation |
|---|---|---|
| 0 | Inception | The Dark Knight Rises |
| 1 | Inception | Mad Max: Fury Road |
| 2 | Inception | Interstellar |
| 3 | Inception | The Revenant |
| 4 | Inception | Spider-Man: Homecoming |
| 5 | Inception | The Prestige |
| 6 | Inception | Shutter Island |
| 7 | Inception | The Dark Knight |
| 8 | Inception | Toy Story 3 |
| 9 | Inception | Warrior |

Fig 12: - Recommendations eg

## Evaluation

The Evaluation of Adamic Adar can be determined by the score provided by Adamic Adar, the higher the value of score the closer the nodes are to the recommendation, which can be seen below.

| Title | Recommendation | Score |
|---|---|---|
| Inception | The Dark Knight Rises | 1.65071083440955 |
|  | Mad Max: Fury Road | 1.17882767875969 |
|  | Interstellar | 1.06981187145939 |
|  | The Revenant | 0.882428994290986 |
|  | Spider-Man: Homecoming | 0.832851422498494 |
|  | The Prestige | 0.824565809649557 |
|  | Shutter Island | 0.805105018589713 |
|  | The Dark Knight | 0.790836235778608 |

|  |  |  |
|---|---|---|
|  | Toy Story 3 | 0.759144404179581 |

### Deployment

We can create a Py2Neo based Web API to get the recommender system of the Neo4j Working, Neo4j itself could be deployed as a cluster server which does the AI Part by itself when provided a pipeline for basic data management and machine learning.

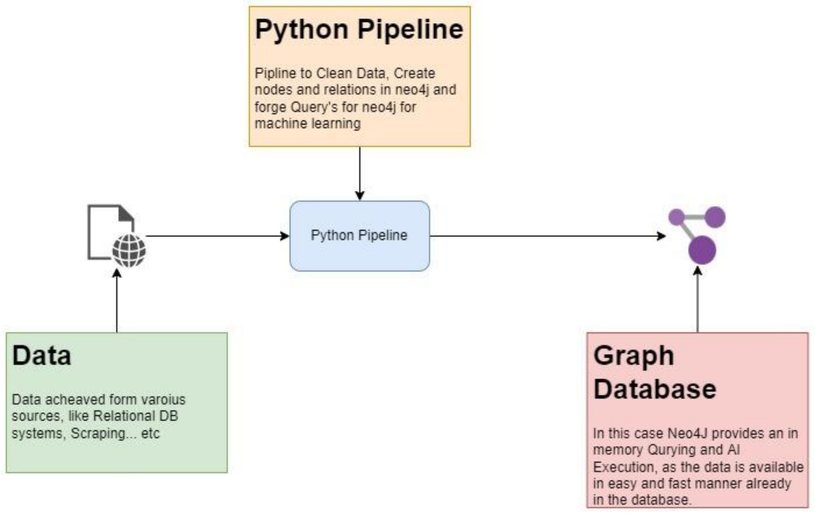

Python Pipeline
Pipline to Clean Data, Create nodes and relations in neo4j and forge Query's for neo4j for machine learning
Python Pipeline
Data
Data acheaved form varoius sources, like Relational DB systems, Scraping... etc
Graph Database
In this case Neo4J provides an in memory Qurying and AI Execution, as the data is available in easy and fast manner already in the database.

Fig 13: - Graph DB Pipeline

## Note

Please find the cypher codes for Excercise 1 and Jupyter Notebook for Excercise 3 in the Zip File

## References

- Neo4j Graph Data Platform. (n.d.). *Adamic Adar - Neo4j Graph Data Science*. [online] Available at: https://neo4j.com/docs/graph-data-science/current/alpha-algorithms/adamic-adar/ [Accessed 21 Apr. 2022].
- Neo4j Graph Database Platform. (n.d.). *The Neo4j Graph Data Science Library Manual v1.8 - Neo4j Graph Data Science*. [online] Available at: https://neo4j.com/docs/graph-data-science/current/.

- Wikipedia Contributors (2022). *Adamic–Adar index*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Adamic%E2%80%93Adar_index [Accessed 21 Apr. 2022].