ReportGraph And AI

short line

Sunil Judhistira Gauda

Student Id: 10595858

# Table of Contents

[Table of Contents](#_w8qd2498gdx3)

[Excercise 1](#_4lqp25cx7kth)

[Creating Nodes for Family](#_tjfqewctqwaf)

[Pseudo code](#_5v5c8bq307gq)

[Creating Relationship between families](#_q4e8ac2d02ki)

[Pseudo code](#_rg0yddcibvqo)

[Centrality Calculations](#_7l0uo0dxobt7)

[Degree Centrality](#_zdijn9cqdrp4)

[Pseudo Code](#_5zi7nrcb3m2y)

[Closeness Centrality](#_gb6139b3yya4)

[Psudo Code](#_t2g5irwp69yd)

[Betweenness Centrality](#_8rx1jtnzyie)

[Psudo Code](#_wdqhgjk39dfh)

[Eigenvector Centrality](#_ayxrw9j36wsq)

[Psudo Code](#_bbtniqjn2pt6)

[PageRank](#_87ivvw38wcq)

[Psudo Code](#_7kbyb3et9es1)

[Final Calculations](#_4lyeey1y7ddz)

[Excercise 3](#_6upsc3wiea00)

[Personal Findings on Applying CRISP-DM Methodology using Neo4J](#_bn0ze21opxb8)

[Business Understanding](#_momq0uhdyq3)

[Data Understanding](#_coa9nwme1w09)

[Data Preparation](#_8eg4dhyufj13)

[Data Cleaning](#_on6udm9u3z7g)

[Create Nodes](#_x5ccn8ll25vb)

[Creating Relations](#_x1n79aa044gx)

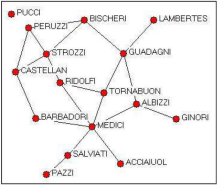
[Modelling](#_aurb4n8110u3)

[Evaluation](#_wori63hbqb1)

[Deployment](#_4pjicp3rfklc)

# 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 >'});**

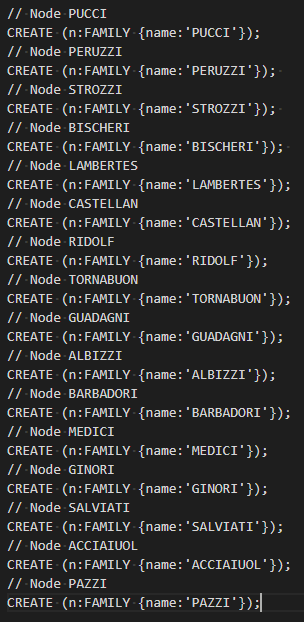


Fig 1 :- Nodes Created in Neo4J

## 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);

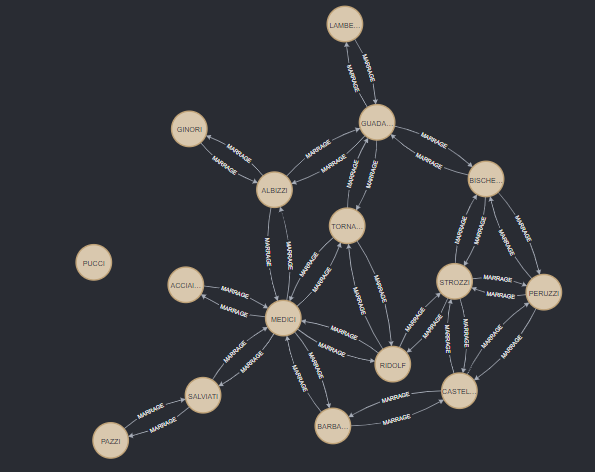


Fig 2 : - Graph Visualised after Nodes added and Relationships Added.

## 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;

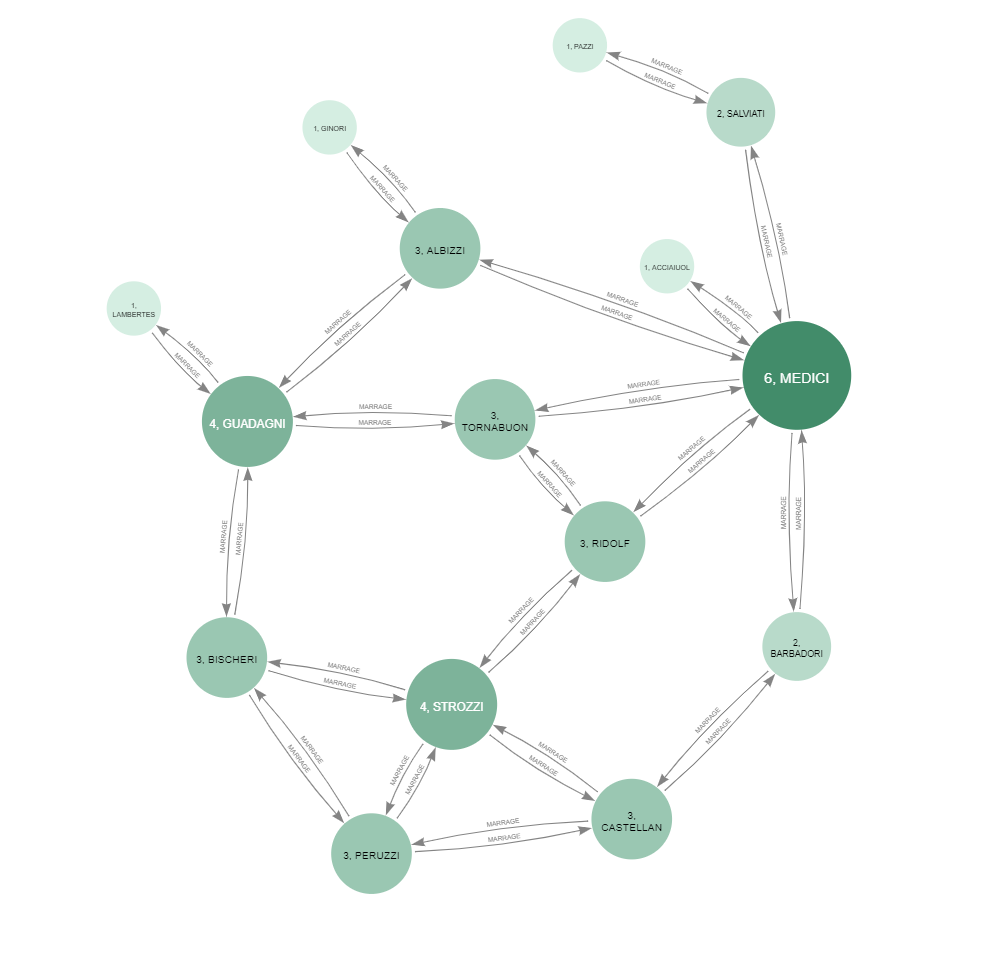


Fig 3 : - Degree Centrality Visual Representation by Size and Color - Using Neo4j Bloom

### Closeness Centrality

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

#### Psudo Code

//Using GDS to calculate closeness

CALL gds.alpha.closeness.write({

nodeProjection: 'FAMILY',

relationshipProjection: 'MARRAGE',

writeProperty: 'centrality'

}) YIELD nodes, writeProperty;

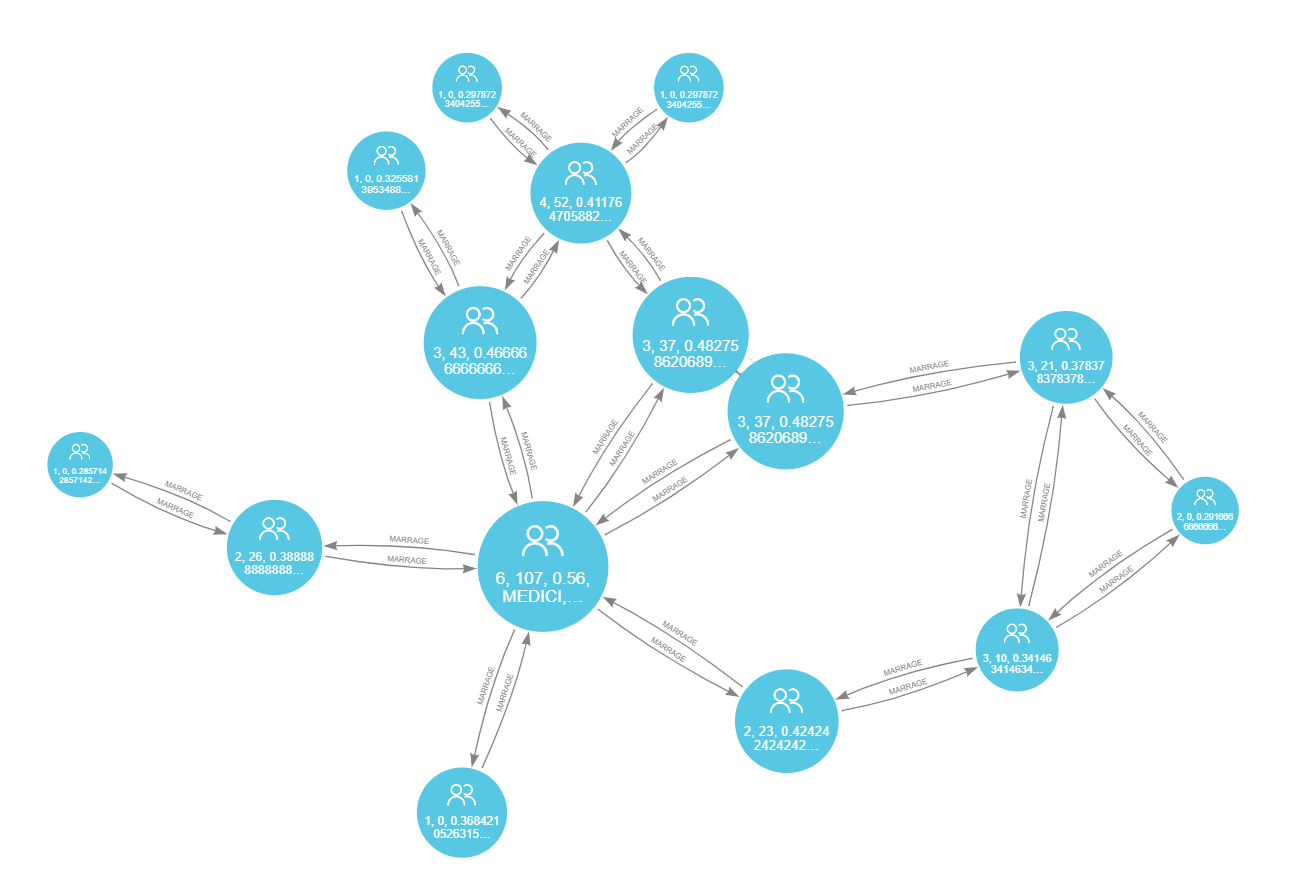


Fig 4 : - Closeness Centrality Visual Representation by Size and Color - Using Neo4j Bloom

### Betweenness Centrality

Betweenness Represents the shortest path in numerical terms

#### Psudo Code

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

YIELD centralityDistribution, nodePropertiesWritten

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

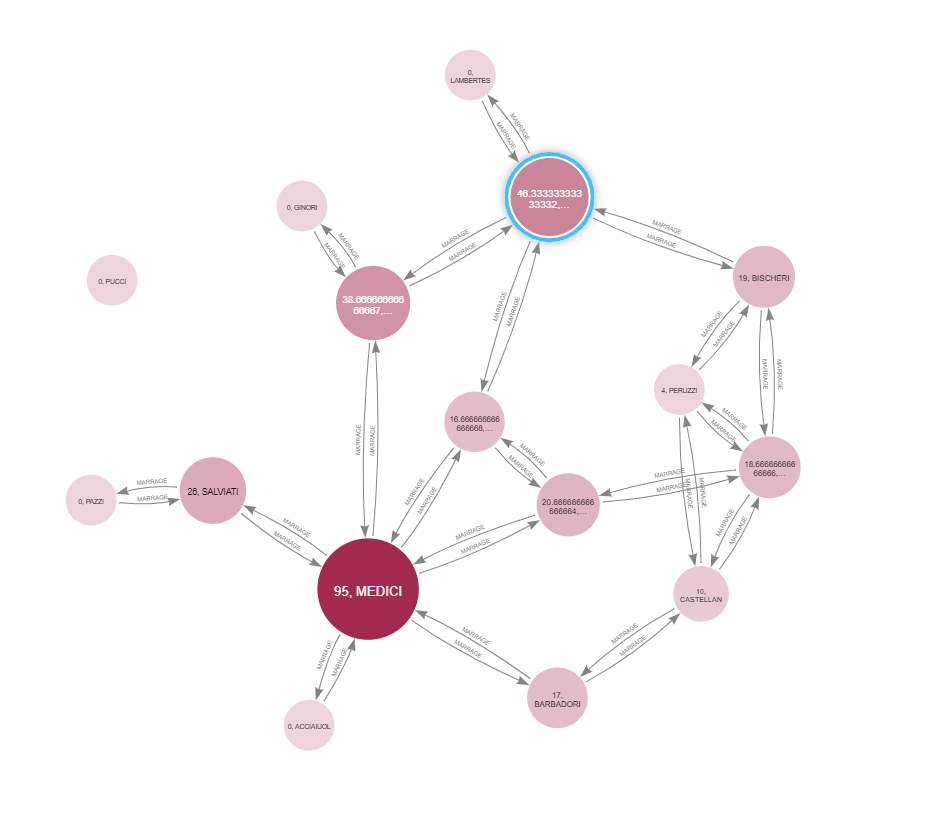


Fig 5 : - Betweenness Centrality Visual Representation by Size and Color - Using Neo4j Bloom

### Eigenvector Centrality

Eigenvector Centrality represents Influence of the node in numerical terms

#### Psudo Code

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

maxIterations: 20,

writeProperty: 'Eigcentrality'

})

YIELD nodePropertiesWritten, ranIterations;

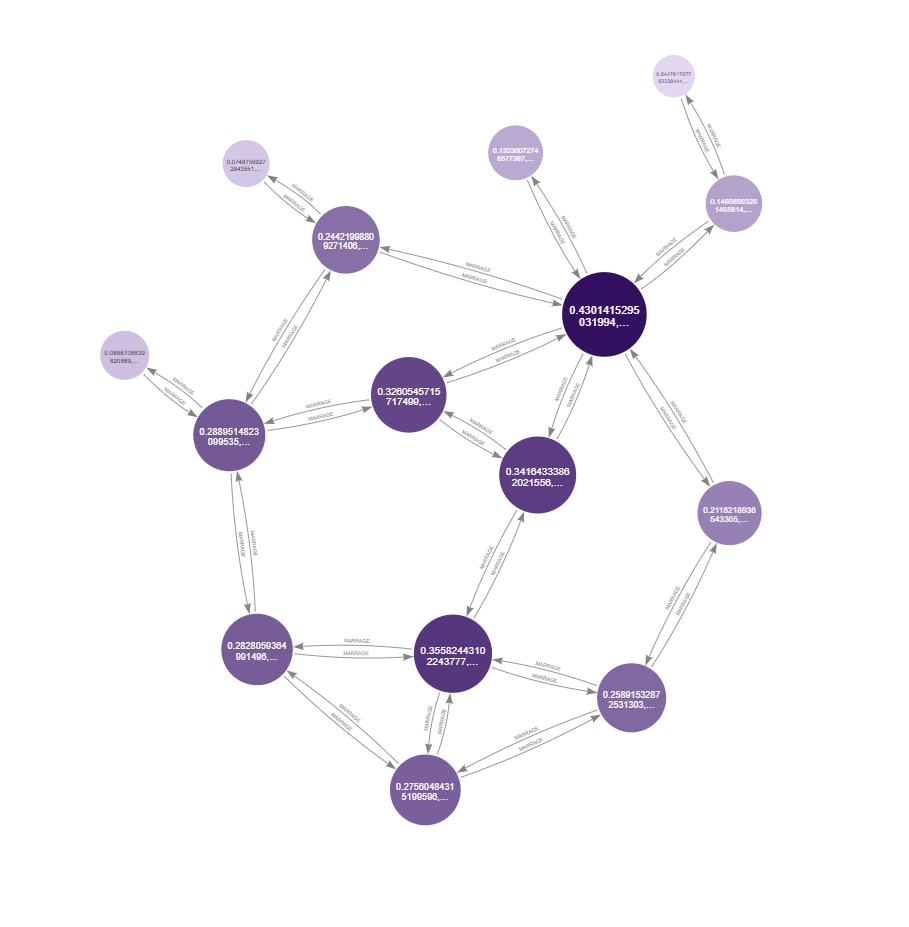


Fig 5 : - Eigenvector Centrality Visual Representation by Size and Color - Using Neo4j Bloom

### PageRank

Pagerank sets Quality of node based on importance of the node

#### Psudo Code

CALL gds.pageRank.write('MEDICIGraph', {

maxIterations: 20,

dampingFactor: 0.85,

writeProperty: 'pagerank'

})

YIELD nodePropertiesWritten, ranIterations;

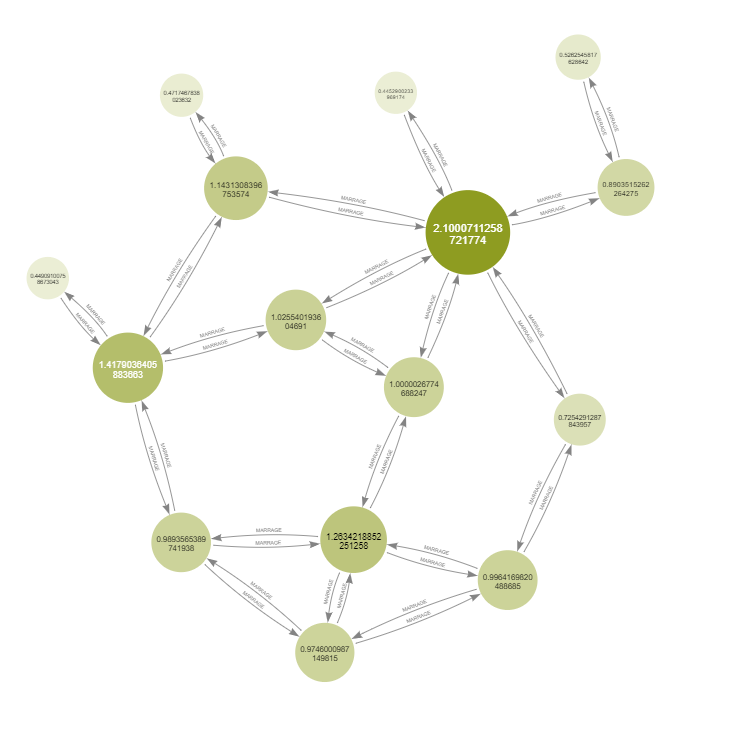


Fig 6 : -Pagerank Visual Representation by Size and Color - Using Neo4j Bloom

## 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 comprises of many people, which includes directors, writers, producers actors and many more, but when choosing a movie recommendation its extremely important to provide information of 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 direacrted 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 a nodes.

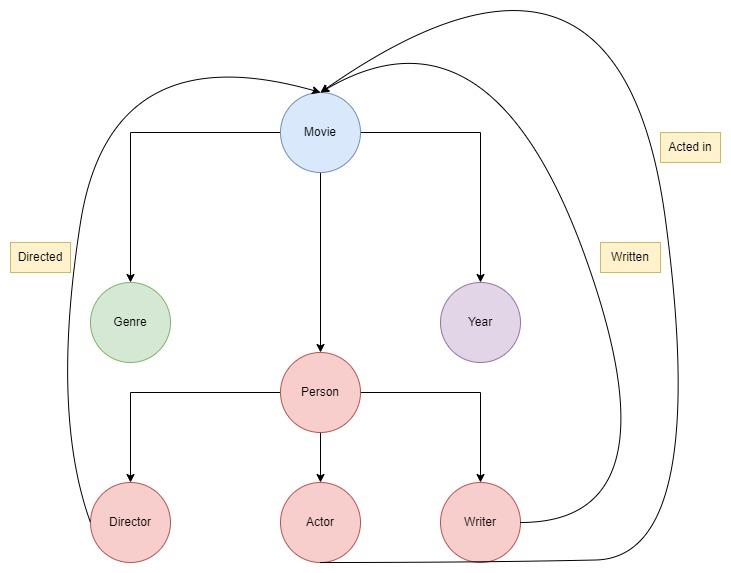


Fig 7 : Relationships Defined as Nodes

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

### Data Preparation

#### Data Cleaning

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

* **Index** : To provide us with 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 column 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 type of moves released depending on 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 tends to choose the movies also directed by the same director which has a similar theme, like : “Christopher Nolan” Movie or “Martin Scorsese” Movie.
* **Writers** : The Entire story of the movie is written by writers of the movie and that also determines how well the movie will do, Usually Novel adaptations have greater chance of success because of the depth in the storys.
* **Actors** : The talent in the movies determine 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 mediocre story could also do ok in industry.
* **Rating** : Rating reflects to the end product of the movie, that is customer satisfaction, which is represented in the data from scale 1 - 10

All the columns above were important for the project, rest was not necessary for our specific usecase, hence it was dropped.

The End Result looks something like this.



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

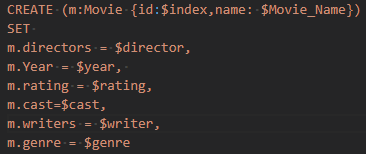


Fig 9 : - Query String to create nodes

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

#### Creating Relations

Psudo 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. Genere
2. CREATED\_ON
3. WORK\_WITH
4. ACTED\_IN
5. DIRECTED
6. NEXT

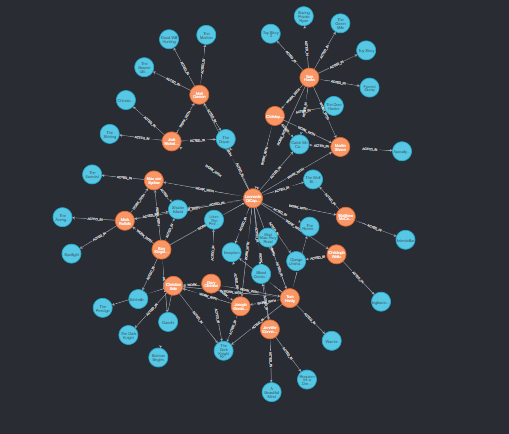


Fig 10 : - Node Preview with relations

### Modelling

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 pre-built models in neo4j to find patterns in the data.

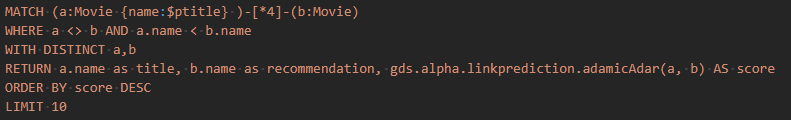


Fig 10 : - Adamic Adar Neo4j Query

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



Fig 11 : - 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.

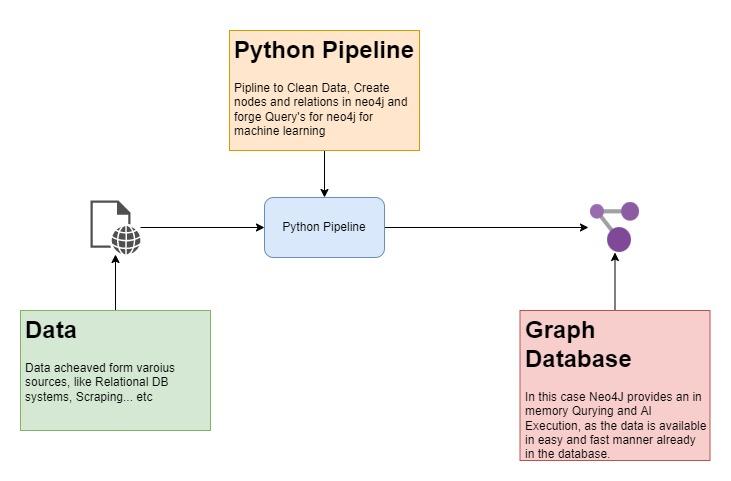


Fig 12 : - Graph DB Pipeline