Information Systems and Applied Computer Sciences
Otto-Friedrich-University Bamberg

Graph Database Representation and Visualization

Bachelor Thesis

Author:	Reem Eslam Mohamed Mekky Khalil
Supervisors:	Prof. Dr. Daniela Nicklas
Submission Date:	31 July, 2019

This is to certify that:

(i) the thesis comprises only my original work toward the Bachelor Degree

(ii) due acknowledgement has been made in the text to all other material used

Reem Eslam Mohamed Mekky Khalil
31 July, 2019

Acknowledgments

I would like to express my sincere gratitude to my Professor, Prof. Dr. Daniela Nicklas for giving me this unique opportunity by accepting me to take part in her chair.

I want to also Thank Nasr Kasrin, for his continuous support, patience, help, and guidance throughout this bachelor project.

I would like to thank all of my doctors, especially Prof. Dr. Slim Abdennadher, Assoc. Prof. Haythem Ismail, Dr. Amr ElMougy, Dr. Mohamed mahdy as they where the reason behind my love to my major Computer science and engineering. I would like also to thank my teacher, friend and sister Eng. Nourhan Ehab for her continuous support, patience, help, and guidance throughout my bachelor degree life stage.

I would like to thank my mother for raising me to be the person i am today and for her contentious support throughout every stage in my life. My dad, my role model who always supports me to be open to the world, depend on myself and receive good education. My Sisters, Rana and Raghda who always supports me in the downs before the ups.

At the end I would like to thank all of my friends for being always by my side despite anything.

Abstract

The world is built like blocks of data that are stacked on top of each other, where the presence of each block affects the sequence and content of the other blocks. In the past, people used to collect information and tried to connect them with each other to understand more about the world. Gathering and combining information is the basis of different fields including History, Physics, Biology, Chemistry, Geology and so on. Data makes us see overall different. Moreover, data gives us the ability to understand more about ourselves and our environment which leads to the enhancement and causes us to upgrade how our brain functions while also aiding us in taking information-based life decisions.

In this thesis,We will present a new technology for data management called **Graph Database**. It is a powerful technology that will revolutionize different fields. The problem with Graph database is the same as every data management program, which is that handling big data in a way where the user can easily analyze, understand and retrieve data in a relatively acceptable duration. Visualizing graphs make us see the full image and discover new patterns. However, limitations of humans perception make human unable to understand or analyze bigger graph. We will talk about two different ways for solving this problem. First is representing data on simple charts so that the user can analyze it easily. Second is adding extra features to the graph in order to make it easier to understand.

Contents

Chapter 1

Introduction

Machine learning, which is one of nowadays hot topics, is based on gathering big data that cover all the situations that the machine may be exposed to and tying them together in order to extract useful information. However, handling big data is one of the biggest challenges in recent days. The contentious growth in technologies, platforms, and users is causing a dramatic increase in the amount of data and its evolution making data more interlinked. It is also hard to guarantee the truthfulness of the data as the size increases. Big data makes us always discover new information but it is not easy to retrieve hidden data without managing it. Relational databases are symmetric. They also require defining everything in the query. Additionally, running complex queries is time consuming as we need to check all the rows in the tables included in the query. Graph databases will be the solution for this problem. We will use nodes and relationships instead of storing data in tables of rows and columns. Consequently, querying the database will be graph tracing. Therefore, any complex query will be directly executed. The only problem with graph databases is the limitations of humans perception make human unable to understand and analyze bigger graphs. Graph database with big data will result in a complex graph that will be difficult for users to understand. For this reason we will use libraries and frameworks for plotting charts to analyze data. We also utilize graph visualization tools that use methods such as clustering and edge thickness to enable users to easily understand the retrieved data and interact with it.
This thesis is organized as follows:

- **Chapter 2 Background**, discusses big data, relational databases, graph databases , relational databases vs graph databases and Visualizing big graphs problem
- **Chapter 3 Charts**, we will talk about charts types, Matplotlib, Seaborn, Bokeh and Plotly.py
- **Chapter 4, Neovis.js**, we will talk about Vis.js and Neovis.
- **Chapter 5 D3.js**, we will talk about D3.js, Force directed graphs, Collapsible Force Layout, Cluster Dendrogram and Tidy Tree.

- **Chapter 6 Recommendation**, brief and personal opinion in case you do not have time to read all of the thesis.
- **Chapter 7 Conclusion**

Chapter 2

Background

In this chapter we will talk about Big data, Relational database, Graph database, compare between Relational database and Graph database, and Visualizing Large.

2.1 Big Data

Big Data is a field which treats ways to analyze, systematically extract information from, or otherwise deals with, data sets[1] that are too large or complex to be dealt with by traditional data-processing application software[2] [34]. Simply, Big data is a term that describe a large volume of structured, semi-structured or unstructured data, making it tedious for commonly used software tools in regards to managing, controlling or processing within a relatively acceptable interval.

In 2016, a more precise definition of big data was released. It states that: "The Information asset characterized by such a High **Volume, Velocity and Variety** to require specific Technology and Analytical Methods for its transformation into Value" [2]. This basically means, there are three main characteristics to describe big data:

- Volume is the quantity of generated and stored data. Volume of data increases exponentially without any bound makes it very hard to manage [26] [27].
- Velocity is the speed of growth and speed of transferring data. Velocity is defined with the volume of data which is growing exponentially making it contributes bigger database [26][27].
- Variety is the type and nature of data that includes structured, semi-structured and unstructured data but unstructured data has more variety of data. The massive amount of varieties of data cause a big issue, either with low volume or high volume[26][27].

[1]**Data set** is a collection of data

[2]**Data-processing** is the collection and manipulation of data items to produce meaningful information

Nowadays, there is more Vs and one C added to the big data which are

- **Veracity**: it is accuracy, truthfulness, and meaningfulness of data. Big Data with huge volume becomes problematic especially when we want to perform some operations on these data, the ability of determining if the operation is done successfully or not will be more difficult. Any data is meaningless if it is not accurate which makes veracity one of big data problems[27].

- **Value**: Big Data = Data + Value [1] as big data is concerned mainly on extracting value from enormous stored data. Extracting values from data reveals hidden truth, uncover useful message and creates value of data while data in itself has no value. Simply, big data is a platform to provide worthiness of unworthy data[27].

- **Validity**: even if we are sure of the accuracy of performed operation on the data, data may not be valid [1]. For example, some data in e-commerce becomes obsolete and it can be truncated on the other hand also some data never obsoletes. Validity of data may differ from time to time so the correct data may not be valid for certain processing[27]. Validity of huge volume of data is difficult to be guaranteed; however, it is a crucial factor because Volume − Validity = Worthlessness [1].

- **Variability and Volatility**: they are conflicting terminologies but both are on the same boat. Volatility is the nature of changing abruptly, sudden change, instability, and changed by unintentionally or anonymously. For example, when a video spreads so fast on social media and the number of views increases exponentially [27]. Variability is the nature of changing, shifting, mutation, and modication with good intention. For example, Katy Perry is currently having the highest number of followers on twitter but after some time data may modify and the second highest person which is Barack Obama become the first. This is the nature of Variability or Volatility.

- **Visualization**: it is the process to show the hidden data of Big Data and it is commonly used in data analysis. Hence, it is the key process to enhance the performance of the data and business processes/decisions. Visualization of Big Data is a very complex process that requires a lot of effort [27].

- **Complexity**: it is the pure form of computational term [27]. Handling huge volume of data always associates with high complexity.

"More data does not only let us see more, it allows us to see better, see new and see different" Kenneth Cukie. It helps us to limit our choices when we want to watch a movie, travel to other country, choose programming languages , apply for a job, a school or a university. Controlling or managing big data is challenging task because data is in contentious increase but on the other hand, big data is so important by which society can make use of it to develop. In the past when we used to have small data we were always wondering what does it mean or how we will use it to understand more about the world.

Now we are having extremely big amount of data that we can not even afford but also we discovered that as data increases we fundamentally can do things that we could not do when we have smaller amounts of data. Big data is so important as it can make us know more about the past or even predict the future.

Big data is usually used for data analysis by users, programmers or certain other advanced data analytic methods that are used to retrieve values from data. The quantity of data now is dramatically increasing and becoming more interlinked with more different kind of relations. Analyzing relations between data can find new correlations to improve technologies, discover new treatments for diseases, business innovation and so on.

2.2 Relational Database

Relational database is a digital database based on the relational model of data proposed by E.F.Codd in 1970 [37]. He introduced the term in his research paper "A Relational Model of Data for Large Shared Data Banks" [9]. The reason was to help users in storing their activities at terminals and most application programs without being affected when the internal representation of data is changed, even when some aspects of the external representation are changed and without having to know how the data is organized. Changes in data representation will often be needed as a result of changes in query, update, report traffic and natural growth in the types of stored information [9]. He made thirteen rules to define what is required from a database management system in order for it to be considered relational [35].

2.2.1 Relational Data Model

Relational Data Model is the primary data model, which is used widely around the world for data storage and processing [28]. The idea of relational data model is simple and it has all the properties and capabilities required to process data with storage efficiency [28]. In relational data model, the database is represented and organized as a collections of relations which means tables. Relations consists of rows and columns, rows represents records/tuples[3] and columns represent attributes. Each table generally represents one Entity[4]. Rows represent instances of that type of entity. Columns represent values/attributed assigned to that instance. Each row in a table has its own unique key called Primary key. Rows in a table can be linked to rows in other tables by adding a column for the unique key of the linked row and it will not be called primary key anymore it will be called Foreign key because there must be only one primary key in each row. Processing data involves consistently being able to select or modify one and only one row so most of the implementations have a unique primary key to each row in a table so when

[3]**Tuple** is one record/row in a relation

[4]**Entity** is anything we can store, about person, concept, physical object or event

a new row is added to the table, a new unique primary key is generated so the system can use this key later to access this row. Primary keys are also used in defining relationships among the tables by migrating to the other table. This is used to represent one to one relationship or one to many relationship. Most relational database designs resolve many to many relationships by creating an additional table that contains the Primary keys from both of the other entity tables. The table is then named and the two foreign keys are combined to form a primary key [37].
In tables 2.1 and 2.2, you can see the pros and cons of Relational database.

2.3 Graph Database

Graph is defined by Bender and Williamson in 2010 as a pair of finite number of vertices and each two vertices are connected by an edge **G=(V,E)** [16]. In 1735, Leonhard Euler represented a solution to a problem known as **Seven Bridges of Knigsberg**[5].The problem stats that there were four land masses separated by water and seven bridges are connecting this four land masses, the question was how to pass these seven bridges without using the same bridge twice. Euler looked to the problem with another respective, he presented each land as a vertex and the bridges as edges connecting between them, he reduced the problem to a graph which helped him to notice and prove that there is no solution for this problem because in order to traverse every edge exactly once, each vertex must have even degree -number of edges connected to vertex- and in this problem all the vertices have odd number of edges.

After that, Euler has published the first paper about **Graph Theory** and since then a new branch was added to mathematics called Topology[6]. Now Graph theory is widely used in more than one field like Path Optimization, Logistics, Social Network, Network analysis, Linguistics, Physics, Chemistry, Biology and Database.

There is wide usage of graphs in different fields, as they help us see the full image of any problem. We live in a connected world where most of the information are related to each others with different kinds of relationships. Accordingly we can map this concept using graphs, people will be presented as nodes and relationships between people will be presented by the edges connecting these nodes which makes data visualization better, because it will show relations that we might not have noticed before. Graph database is more unique than other kind of databases as it can map the human brain more realistically, process the world around it, prioritize the representation and maintain data relationships [31]. Graph database is the future because it helps in discovering new relations in business, technologies and many other fields.

[5]**Knigsberg** is the name of the historic German city that is now Kaliningrad in Russia.
[6]**Topology** is concerned with the properties of a geometric object that are preserved under continuous deformations, such as stretching, twisting, crumpling and bending, but not tearing or gluing.[38]

Graph database consists of two elements nodes and edges. Nodes represent people or entities and edges represent relationships between nodes [36]. For example, Twitter connects 330 million monthly active users as shown in figure 2.1. Each node represents a single user and edges between them represents the follow relationship.

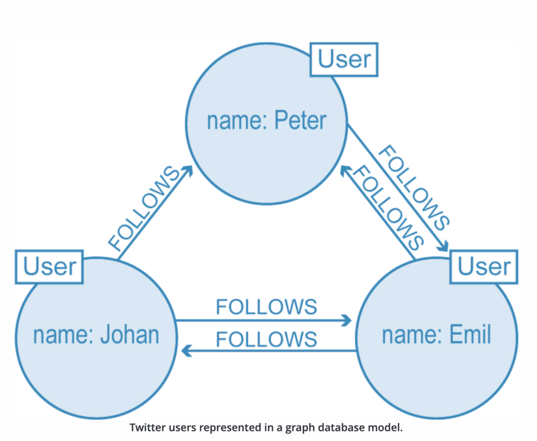


Figure 2.1: Example of Twitter user in graph database [31].

Neo4j is a graph database management system. It has the same structure of graph database, nodes and relationships connecting them. Each Node or relationship can have any number of attributes. Nodes and relationships can be labelled, that will help in narrowing searches. Querying language of Neo4j is Cypher. Cypher is also heavily based on nodes and relationships and is designed to recognize various versions of these patterns in data, making it a simple and logical language for users to learn [11]. Cypher is designed to be human readable, it is based on English prose to make syntax easily understood [11]. In tables 2.1 and 2.2, you can see the pros and cons of Graph database.

2.4 Relational database vs Graph database

Tables 2.1 and 2.2, are comparison tables between Relational database and Graph database in terms of their pros and cons.

Relational database	Graph database
• **Simplicity:** a relational data model is simpler than the hierarchical and network model. • **Structural Independence:** relational database is only concerned with data and not with a structure. • **Easy to use:** as the concept of tables consisting of rows and columns is quite natural and simple to understand. • **ACID support in relational database:** it is to make sure that you could trust that once the data was committed, it would be accessible to future queries.	• **Performance:** as data volume grows with more connections or relationships, performance of retrieving data will be more complex in traditional database technologies while in graph database performance stays constant with an increase in data volume. • **Flexibility:** graph database is schema and index free, so there is no need to remodel the database whenever there is a change instead, you can simply add to the existing structure without effecting the functionality. • **Time:** querying data will be just traversing the graph so the time will not be effected by the complexity of the query. • **Querying is human readable** Cypher is designed to be human readable, it is based on English prose to make syntax easily understood.

Table 2.1: Relational database vs Graph database in terms of advantages of using them [31] [10]

Relational database	Graph database
• The tables are human readable but as soon as you normalize the data to eliminate duplication or inconsistencies many fields start referencing auto-generated foreign keys and the tables become more difficult to understand or maintained without joining complicated queries [3]. • Relational databases can sometimes become complex as the amount of data grows, and the relations between pieces of data become more complicated. Complex relational database systems may lead to isolated databases where the information cannot be shared from one system to another [10]. • The design of Relational database, is so symmetric as retrieving data for a query requires looking in every row in each table included in the query ,beside the fact that we need to define everything in the query. For example, assume X is a person and we want to get all the people that X considers as his/her friend or the opposite, we want to retrieve all people who consider X as their friend. That kind of relations are so expensive because we need to look-up in all the rows to get them and if we go more deep in queries it will become more complex and it will need more computations. In addition to the time taken to get all kind of relations between the two entities.	• As data grows the corresponding graph becomes more visually confusing and it will not be easy on user to distinguish between nodes and edges.

Table 2.2: Relational database vs Graph database in terms of disadvantages of using them

2.5 Visualizing Large Graphs

Big data is a complex field, we always need to use visualization techniques in order to understand what is happening. For example, assume we need to make a graph representing humans life. We will need to know where they were born, when they were born, countries they visited before, people they met. Each person we analyze is connected to other people which means more relations and bigger graph; thus, it will be very difficult to untangle the relationships. Additionally, some people maybe similar in some things but different in others which also makes it difficult to optimize the graph. Human brain processes visual information 60,000 times faster than text, and 90 percent of the information transmitted to the brain is visual [17]. Graph visualization is very important in different fields as it makes it easy to present connected real world data beside the additional value for the data the graph shows. Representing The full graph makes us see the full image so we can easily see the connections between data in the areas of interests and come up with new conclusions.

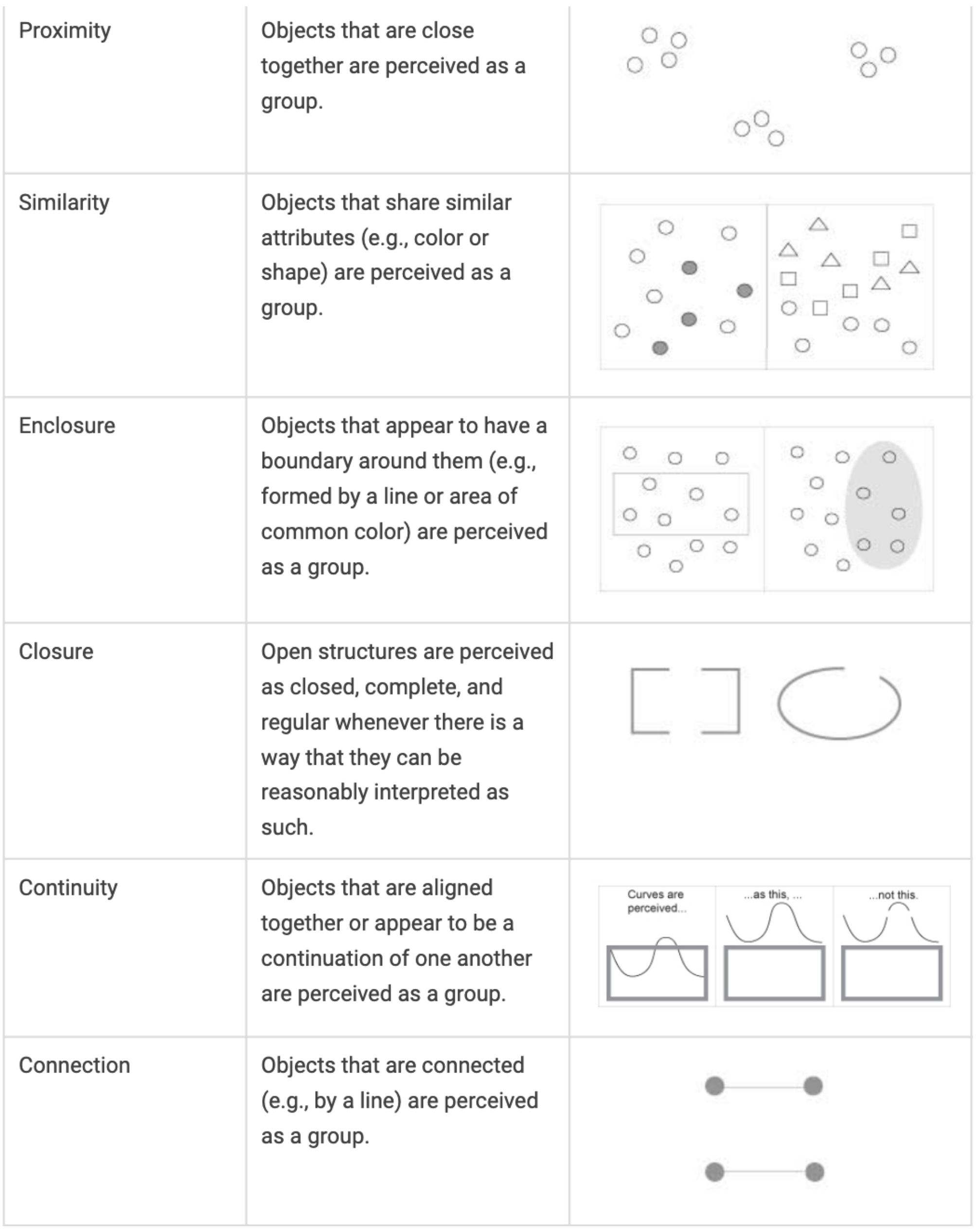
Our problem with Graph database that as data grows the corresponding graph becomes more visually confusing and it will not be easy on user to distinguish between nodes and edges. Limitations of humans perception abilities makes human unable to understand or analyze bigger graph, despite the simplicity of the graph. Gestalt School of Psychology made contributions to the science of perception, they observed that "We organize what we see in particular ways in an effort to make sense of it" [18]. The result was series of Gestalt principles of perception, it is still respected today as accurate description of visual behaviour, You can find few of the principles in 2.2.

Visual encoding is influenced by visual channels that can dramatically affect representational power such as: length, color, position, shape and size of nodes, or thickness of an edge representing a relationship. Neo4j has limited options of nodes sizes and colors, and we can not change edges length or thickness according to its property value so, When we apply it on Big Data at a point there will be similarities and that will confuse the user and may make it even impossible to understand the graph. The solution to our problems is to use other techniques that can help us with this issue.

There are two categories for graph visualization tools.

- Embeddable tools with built-in Neo4j connections (NeoVis, Popoto.js)).
- Embeddable libraries without direct Neo4j connection(Vis.js, D3.js).

In figure 2.2, there are some of Gestalt principles.



Proximity	Objects that are close together are perceived as a group.	
Similarity	Objects that share similar attributes (e.g., color or shape) are perceived as a group.	
Enclosure	Objects that appear to have a boundary around them (e.g., formed by a line or area of common color) are perceived as a group.	
Closure	Open structures are perceived as closed, complete, and regular whenever there is a way that they can be reasonably interpreted as such.	
Continuity	Objects that are aligned together or appear to be a continuation of one another are perceived as a group.	Curves are perceived... ...as this, ...not this.
Connection	Objects that are connected (e.g., by a line) are perceived as a group.	

Figure 2.2: Gestalt principles [18]

Chapter 3

Charts

In this chapter we will discuss how to choose the right chart for presenting your retrieved data, different python plotting libraries (Matplotlib, Seaborn, Bokeh and Plotly.py). We will see the results of applying Plotly.py and Seaborn on a movie database query to compare between them and show different charts of Plotly.py resulted from other queries on the same database. At the end there is two comparison tables between the four libraries Matplotlib, Seaborn, Bokeh and Plotly.py.

First solution for data visualization is plotting the required data on Charts[1] so user can easily analyze data and understand the results. First we need to know more about graphs and how to select the best chart for your query.

3.1 Charts Types

There are four basic presentation types that you can use to present your data [19]
In order to choose the best chart for your data, you need to answer the following questions [19] .

- How many variables do you want to show in a single chart?
- How many data points will you display for each variable?
- Will you display values over a period of time, or among items or groups?

Data presentation Types :

- **Comparison Graphs**, we use it to compare sets of values.
  Example,

[1]**Charts** is a graphical representation of data where, the data is represented on symbols as bars in bar chart, lines in line chart or slices in pie chart

  - **Line chart** is frequently used for comparing changes of the output in an interval scale where intervals are equal in size. Example, Visualizations of data over a period of time
  - **Bar chart** is a horizontal column chart, used to visualize patterns in the data and to compare outputs between different groups.
  - **Area diagram** is the area under the line. It is used to present accumulative values[2] change over time, like item stock, number of employees or savings accounts.

- **Relationship graphs**, we use it to see The relations between the variables.
  Example,
  - **Scatter chart** is used to show the relationship between two different variables and whether one variable correlates[3] to another or not. Also, it is used to show the data distribution.
  - **Bubble chart** is used to display three dimensions of the data. It can be used in case you want to add another dimension to the scatter plot.
- **Distribution graphs**, we use it to show the distribution of a set of values.
  Example,
  - **Column histogram** is used to present distribution and relationships of a single variable over a set of categories. Example, distribution of grades on school exam.
  - **Scatter chart** is used to show the data distribution or the clustering trends as it visualizes where most of the data seems to bunch up and can clearly display the outliers.
- **Composition graphs**, we use it when we want to show how various parts of the data make up the whole data.
  Example,
  - **Pie chart** is used to represent the date in percentages, to visualize a part to whole relationship or the composition of the data.
  - **Stacked area chart** is an extension of the basic Area charts, used to show changes or the evolution of the value of several groups in the same plot.

In 3.1, we see a diagram that helps in choosing the best chart for a given data.

[2] **Accumulative value** is the gradual growth of value by successive additions.

[3] **Correlates** is having a mutual relationship or connection, in which one thing affects or depends on another

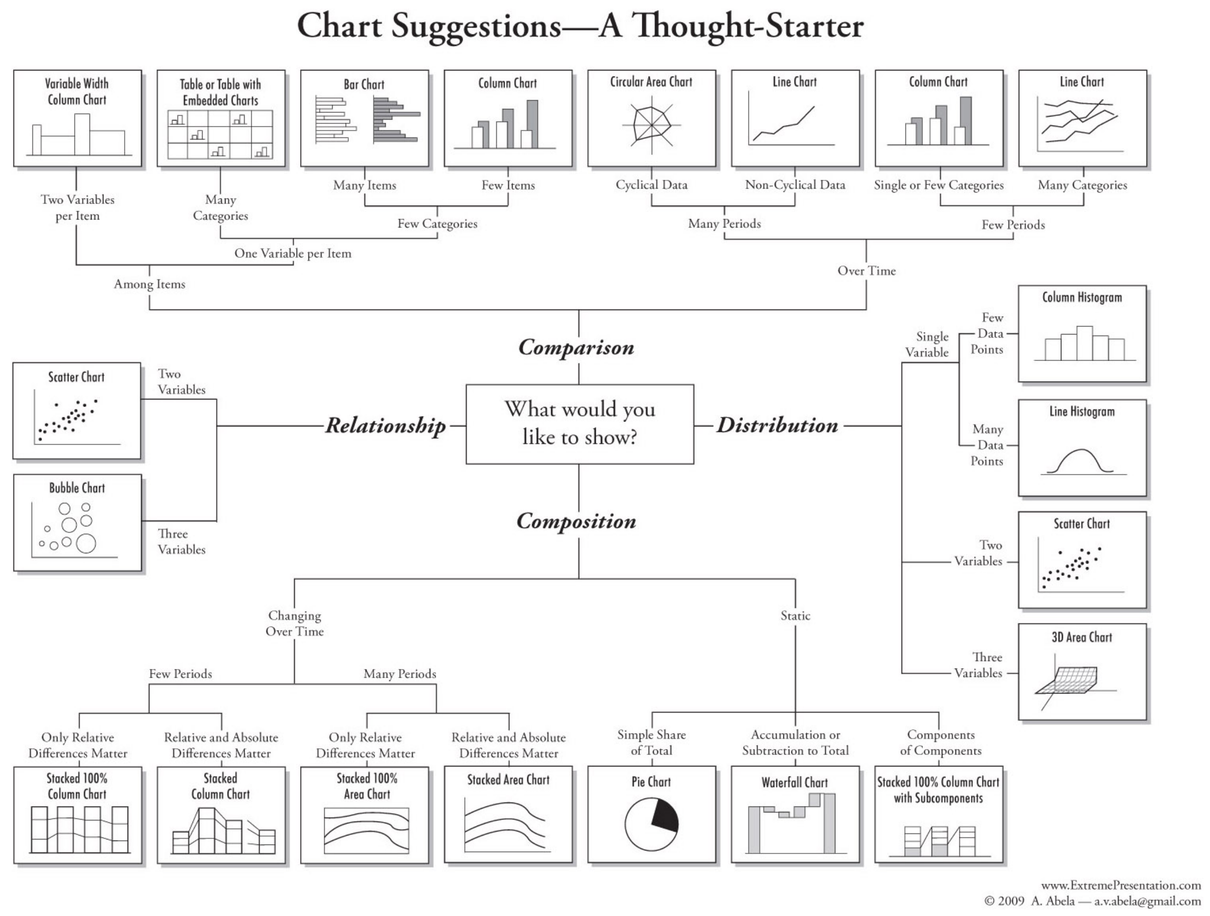


Figure 3.1: Choosing the best chart for a given data, by Dr. Andrew Abela

There are lots of different libraries for plotting charts but in this thesis, we will discuss four different plotting python libraries.

3.2 Matplotlib

Matplotlib "is a python 2D plotting library which produces publication quality figures in a variety of hard-copy formats and interactive environments across platforms" [21]. In other words it is used to create 2d graphs and plots using python scripts. Matplotlib is organized in a hierarchy of nested Python objects[21].

A Figure object is the outermost container or box which has one or more axes [32]. There is a difference between Axis and Axes. Axes is the area where your plot appear, a given figure can contain one or more axes, but axes object can only be in one Figure. Axis is the axis of the graph like x-axis, y-axis and a z-axis in case of 3d plots. Below the axes in the hierarchy you can control smaller objects like line styles, font properties, formatting axis, images and texts. Matplotlib has a module called **Pyplot** which provides the state-machine interface to the underlying object-oriented plotting library. The state-machine implicitly and automatically creates figures and axes to achieve the desired plot [21].Matplotlib is a static data visualization library which means that user cannot interact with the resulted plot like zoom in, zoom out, select certain part to analyze or download it but, by adding one command(%matplotlib notebook) in jupyter notebook you will be able to zoom in/out, re-size-able and download. In 3.2, we can see Matplotlib Hierarchy.

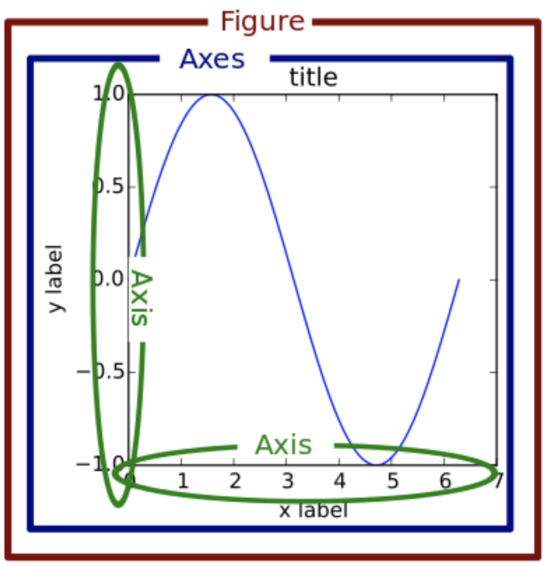


Figure 3.2: This Diagram shows Matplotlib Hierarchy [32]

Lets try a dummy example to see, the code structure for Matplotlib in python to make a simple chart and the design of charts in Matplotlib.

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.show()
%matplotlib notebook
```

In 3.3, we can see the result of applying the dummy example.

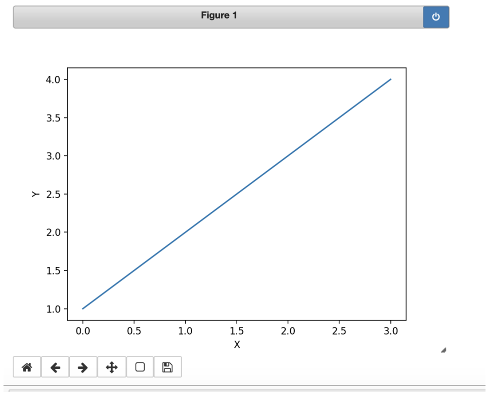


Figure 3.3: Dummy Example for Matplotlib on notebook

In table 3.1, we can see the pros and cons of Matplotlib.

Pros	Cons
• Gives you the control on tiny details. • Simple hierarchical structure. • Seaborn and Pandas were built on top of Matplotlib.	• The documentation needs to be more simple and up to date. Some online examples can take 70% fewer lines of code in their modern version. • Static data visualization library. • Needs too much manual work to plot complex graphs.

Table 3.1: Pros and cons of using Matplotlib [32]

"Matplotlib is extremely powerful but with that power comes complexity" Chris Moffitt

3.3 Seaborn

Seaborn is a python data visualization library built on top of matplotlib and is integrated with **Pandas**[4][33]. It provides high-level interfaces for drawing attractive and informative statistical graphics. Seaborn uses *Matplotlib* to draw plots behind the scenes. Many tasks can be done using just Seaborn but in some cases to be able to customize more, it might

[4]**Pandas** is an open source library for python used as a data analyzing and data structure tool

be necessary to use Matplotlib directly [33]. Seaborn is also a static data visualization library but by adding one command(%matplotlib notebook) in jupyter notebook you will be able to zoom in/out, re-size-able and download.
Lets try a dummy example to see, the code structure for Seaborn to make a simple chart and the design of charts produced by Seaborn.

```python
import seaborn as sns; sns.set()
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib notebook
df=pd.DataFrame()
df['X']=[1,2,3,4]
df['Y']=[1,2,3,4]
ax = sns.lineplot(x="X", y="Y", data=df)
```

In 3.4, we can see the result of applying the dummy example.

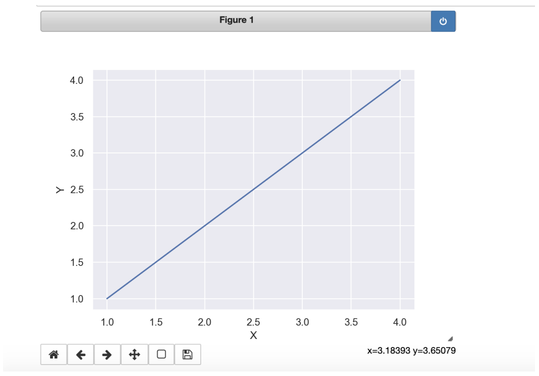


Figure 3.4: Dummy Example for Seaborn on notebook

Pros	Cons
• Less manual work than Matplotlib.	• Static data visualization library.
• Has a built in themes for styling matplotlib graphics.	• Need to know matplotlib to tweak and edit Seaborns defaults.
• Works well with NumPy [5] and Pandas data structures.	

Table 3.2: Pros and cons of Seaborn [14]

"If *Matplotlib* tries to make easy things easy and hard things possible, *Seaborn* tries to make a well-defined set of hard things easy too" *Michael Waskom*

3.4 Bokeh

Bokeh "is an interactive visualization library that targets modern web browsers for presentation"[5]. *Bokeh* has multiple languages bindings like **Python**, **R** [6], **Lua** [7] and **Julia** [8] [29]. All of these bindings act as an input for BokehJS which is a javascript library which then presents data on modern web browsers [29]. In 3.5, we can see the process flow of how Bokeh presents data to a web browser.

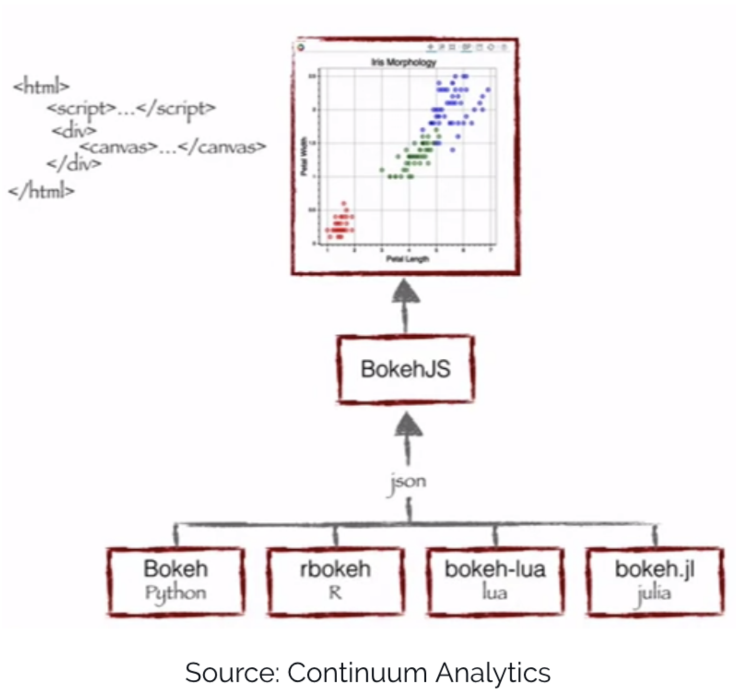


Figure 3.5: The process flow of how *Bokeh* helps to present data to a web browser [29]

Bokeh produces elegant and interactive Visualizations with high performance over large or streaming data [5].It is used to create interactive plots, dashboards, and data applications. It integrates also with Pandas by using Column Data Source class. There are built-in tools that can be included on widget box attached to the plot that makes the user interact easily with the graph in order to zoom-in, zoom-out, selecting ...etc.
Lets try a dummy example to see, the code structure for Bokeh to make a simple chart and the design of charts produced by Bokeh.

[6]**s a programming language and free software environment for statistical computing and graphics**

[7] **Lua**lightweight, multi-paradigm programming language designed primarily for embeddable scripting language for applications

[8]**Julia** is programming language designed for high-performance numerical analysis and computational science.

```
import bokeh
from bokeh.models import Title
import numpy as np
from bokeh.models import Circle, ColumnDataSource, Line, LinearAxis,
    Range1d
from bokeh.plotting import figure, output_notebook, show
from bokeh.core.properties import value

output_notebook()
data = {'X': [1, 2, 3, 4],
        'Y': [1,2,3,4]}
x=[1,2,3,4]
y=[1,2,3,4]
tooltips = [
    ("x", "$x"),
    ("y", "$y"),
]

p = figure(title="X", plot_width=300, plot_height=300,title_location="
    left", x_range=(0, 4), y_range=(0, 5),
           tools='hover,wheel_zoom', tooltips=tooltips)
p.add_layout(Title(text="Y", align="center"), "below")

p.line(x="X",y="Y",source=ColumnDataSource(data=data))
show(p)
```

In 3.6, we can see the result of applying the dummy example.

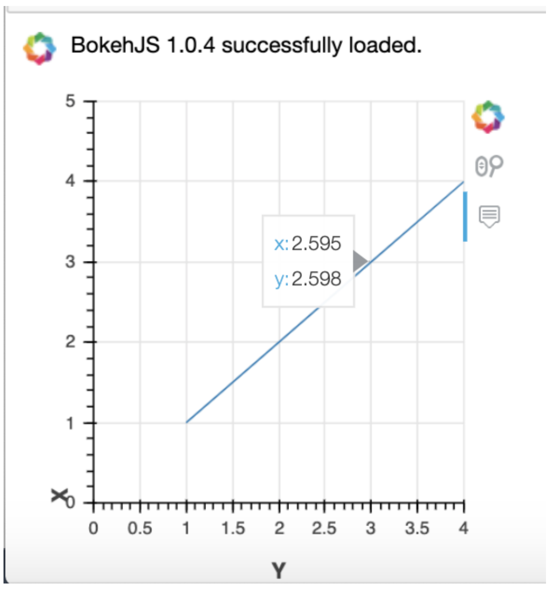

BokehJS 1.0.4 successfully loaded.

Figure 3.6: Dummy example for *Bokeh* on jupyter notebook

Pros	Cons
• Interactive data visualizing library with, different styling option. • Easily build complex statistical plots through simple commands. • Has multiple language bindings like Python, R, Lua and Julia. • Works well with NumPy and Pandas. • Has **Tooltip**[9] tool. • Can be embedded to **Flask**[10] and **Django**[11].	• The documentation of Bokeh does not have enough examples. • Lots of manual work is required which is time consuming to look for the commands. • Labeling axis needs some manual work to put the label in a good position.

Table 3.3: Pros and cons of Bokeh [29]

3.5 Plotly.py

Plotly.py is an interactive data visualization library built on top of **Plotly.js**, which is built on **D3.js** [25]. It has api wrappers for R, Julia and many other language but, not all languages have available online examples [20]. It supports wide range of visualization types like financial statistical visualization, geographic use cases and even some advanced three dimensional use cases. It is an open source technology that can also work offline without requiring an account. It supports also Pandas and NumPy but, you will need to install cufflinks which is used to connect Plotly.py with Pandas. Plotly.py‘s documentation is direct, and there are more than one example for every type of graph, and also the syntax is more simple and direct than Bokeh but, sometimes you will need also to write a couple of lines to improve the plot. On top of Plotly is **Dash**. Dash is an open library source used for building data visualization applications with highly custom user interfaces using only Python [22]. Dash uses only Python instead of all the needed protocols to build interactive web-based applications. Dash‘s applications are rendered in the web browser so you can later deploy your applications to the server [22].

Lets try a dummy example to see, the code structure for Plotly.py to make a simple chart and the design of charts produced by Plotly.py.

```python
import plotly.graph_objs as go
import plotly as py
import pandas as pd
import numpy as np
import plotly.graph_objs as go
py.offline.init_notebook_mode(connected=True)
layout=go.Layout(
    title="Dummy Example",
    yaxis=dict(title='X'),
    xaxis=dict(title='Y'))

x=[1,2,3,4]
y=[1,2,3,4]
trace1=go.Scatter(
    x=x,
    y=y,
    mode='lines'

)

#Create the figure with the layout and the data of x axis and y axis
fig=go.Figure(data=[trace1],layout=layout)
py.offline.iplot(fig)
```

In 3.7, we can see the result of applying the dummy example.

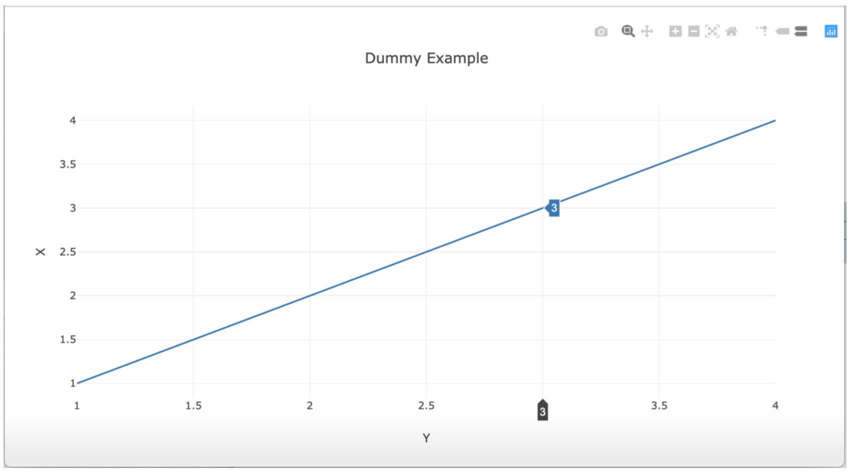


Figure 3.7: Dummy Example for Polotly.py on notebook

Pros	Cons
• It is an interactive data visualization library. • Uses declarative programming [12]. • Has api wrappers for **R**, **Julia**, and many other language. • Supports *NumPy* and *Pandas*. • Has Tooltip tool. • More direct than Bokeh in labeling the axis. • Does not need a lot of manual work as Bokeh does, and the syntax is easy to memorize.	• You need to install cufflinks to connect *Plotly.py* with *Pandas*. • The user interface of *Plotly.py* website is confusing and needs to be more clear and direct. • The documentation's introduction is not informative.

Table 3.4: Pros and cons of Plotly.py [24]

3.6 Plotly.py outputs

We will see the results of applying Plotly.py and Seaborn on a movie database query to compare between them. In the appendix you will find the full code of the first query with the description check section A.1 page 42 in the appendix.
In 3.8, Plotly.py output on a movie database query.

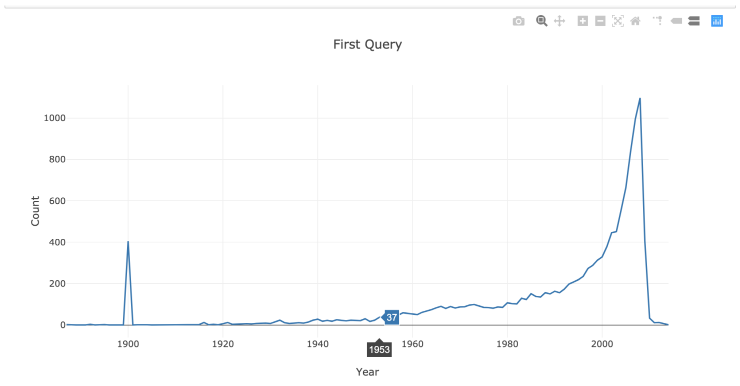


Figure 3.8: The resulted graph of the first query by Plotly.py

In 3.9, Seaborn output on the same query.

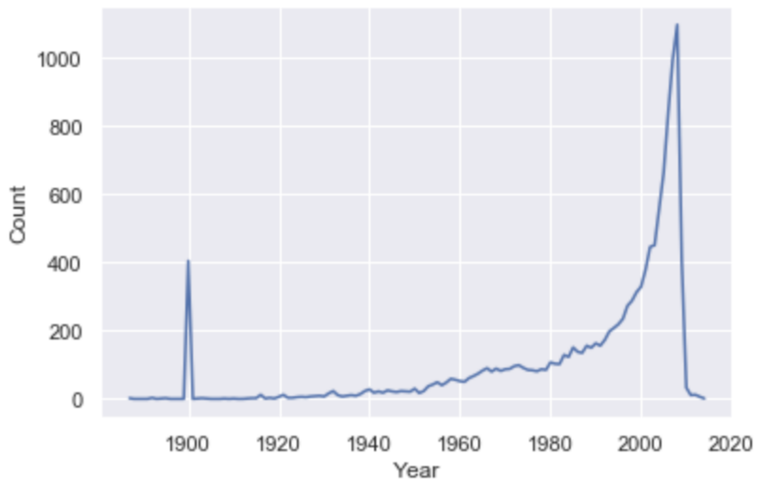


Figure 3.9: The resulted graph of the first query by Seaborn

As we can see the difference between the two graphs. Seaborn graph contains just the resulted plot but you can not interact with the graphs while, in Plotly.py graph beside the interaction there is the Tooltip tool.

3.10, 3.11 and 3.12 are different output charts of different queries on the same database.

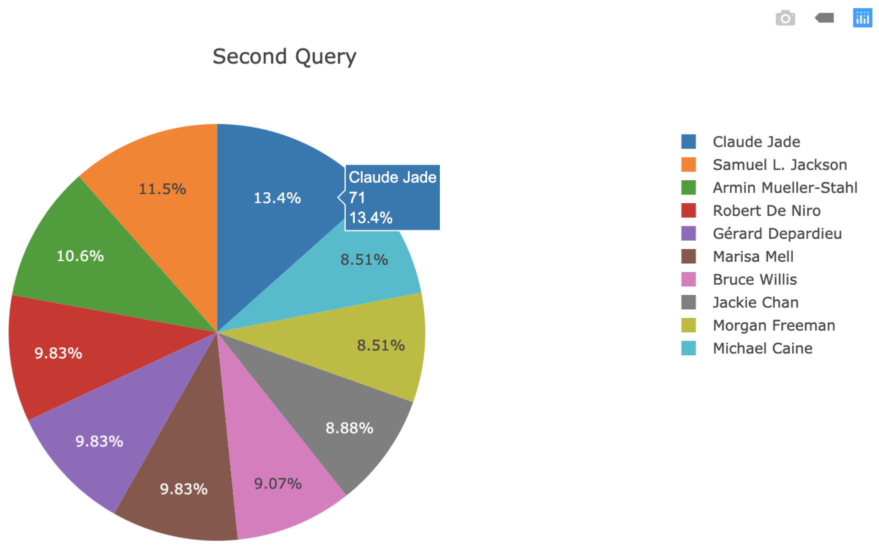


Figure 3.10: Top 10 Actors with maximum count of movies

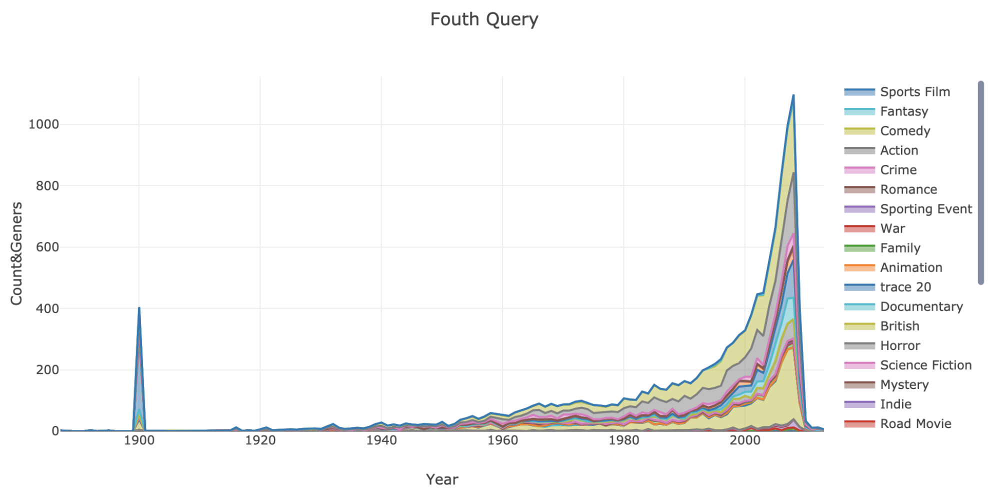


Figure 3.11: Distribution of movies according to genre over years

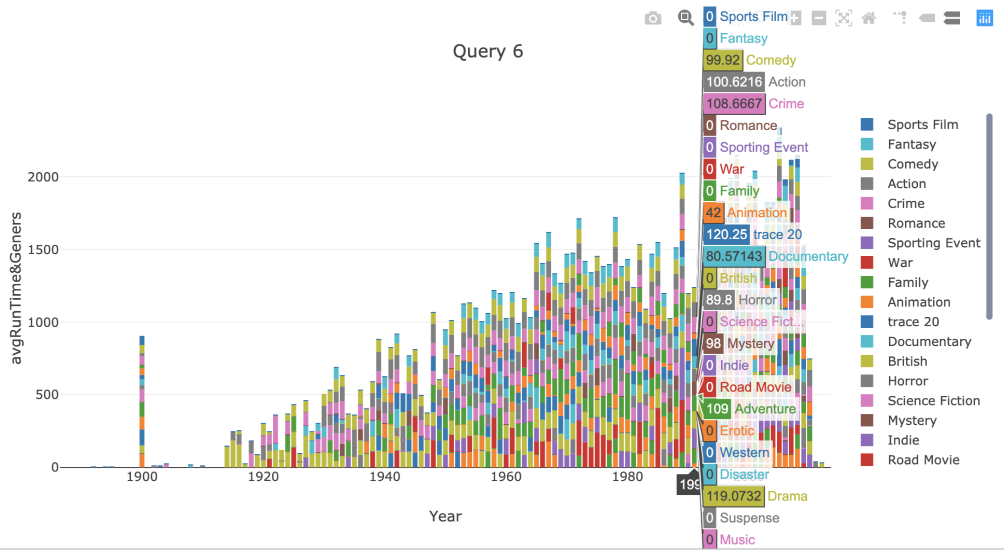


Figure 3.12: Average running time of films based on genre over time

In tables 3.5 and 3.6, we compare between the four python libraries in terms of some supported charts in each and technology review

Name	**Matplotlib**	**Seaborn**	**Bokeh**	**Plotly**
Base Technology	Python	Python	Python	D3.js
Last update	23.06.2017	01.07.2018	01.04.2019	07.05.2019
REST Interface	Yes	Yes	Yes	Yes
Licence	PSF	BSD	MIT, BSD	MIT
Stack overflow	39507	3370	3075	4504
Direct support for Neo4j	Yes	Yes	Yes	Yes
Supported languages	Python library	Python library	Python, Julia, Lua and R	Python, Ruby, R, Julia and more

Table 3.5: Technology Review table

Name	**Matplotlib**	**Seaborn**	**Bokeh**	**Plotly**
Scatter Chart	Yes	Yes	Yes	Yes
Histogram	Yes	Yes	Yes	Yes
Bar chart	Yes	Yes	Yes	Yes
Line	Yes	Yes	Yes	Yes
Pie charts	Yes	No	Yes	Yes
Area Chart	No	Yes	Yes	Yes
Horizontal bar chart	Yes	Yes	Yes	Yes
Bubble chart	Yes	Yes	No	Yes
gantt charts	No	No	No	Yes
population pyramid charts	No	No	No	Yes
Log plots	Yes	No	No	Yes
Spectrogram plots	Yes	No	No	No

Table 3.6: Graphs supported by each Technology

Chapter 4

NeoVis

In this chapter we will talk about Vis.js which is embeddable library without direct Neo4j connection and Neovis which is embeddable tool with built-in neo4j connection and built on top of Vis.js.

4.1 Vis.js

Vis.js is dynamic, browser based visualization library designed to be easy to use, handle large amounts of dynamic data, and enable manipulation of and interaction with the data [15]. The library consists of different components. Network component is the one used to display network of nodes and edges it is easy to use and have numerous customizations available for styling nodes, labels, animations, coloring, grouping, and others. The library consists of [15].

- **Data-Set** used to manage unstructured data using, add, update, and remove data. Listen for changes in the data.
- **Graph2d** draws graphs and bar charts on an interactive time-line
- **Graph3d** creates interactive, animated 3d graphs like lines, dots and block dimensional graph.
- **Network** creates interactive, animated 3d graphs, surfaces, lines, dots and block styling out of the box.
- **Timeline** creates a fully customizable, interactive time-line with items and ranges.

In table 4.1, we will discuss the pros and cons of Vis.js.

Pros	Cons
• Widely used with lots of online examples. • Supports three dimension graphs, custom shapes, styles, colors, sizes, images, and more. • Provide user interaction with the network like mouse events, navigation buttons and the pop-ups. • Handles physics simulation, moving the nodes and edges to show them clearly.	• It does not have direct connection with *Neo4j*. • You need to explicitly define nodes and relationships • User can not query the data or interact with the graph except moving the nodes and relations.

Table 4.1: Pros and cons of Vis.js [6]

4.2 NeoVis.js

NeoVis.js is “Graph visualization tool powered by vis.js with data from Neo4j” [23]. It is designed to combine between Neo4j‘s javascript driver(to connect and fetch data from neo4j) and javascript visualization library called **Vis.js**. NeoVis is new library released on second of April 2018. Its is considered as the first step for solving visualizing big graphs problem as it connects instantly with neo4j so you can query the database directly from Neovis. It is very simple to use and easy to learn since, it is built with considering Neo4j property graph model. Controlling coloring styles based on relationships, labels and properties is defined in a single configuration object.

There is six main Features for NeoVis

- Connect to Neo4j instance to get live data
- User specify labels and properties to be displayed
- User specify Cypher query to populate
- Specify edge property for edge thickness
- Specify node property for node size
- Specify node property for community / clustering

[0]**Clustering** is grouping nodes by making nodes that are more similar, more to near to each than other different nodes.

In table 4.2, we will discuss the pros and cons of Neovis.js.

Pros	**Cons**
• Simple to use and easy to learn • Connects to Neo4j instantly to get live data • User specify labels and properties to be displayed • Specify edge property for edge thickness • Specify node property for clustering • Specify node property for node size • Can make hierarchical for the result hubsize or directed	• Documentation needs to be more detailed • No enough examples available online • Should inherit more graph visualization features from Vis.js like, Configuring popover and assigning URL images.

Table 4.2: Pros and cons of NeoVis [23]

In tables 4.3 and 4.4 we will compare between Neovis, Vis.js and Neo4j web interface.

NeoVis	**Neo4j web interface**
• Specify edge property for edge thickness • Specify node property for node size • Specify node property for Clustering • Can make hierarchical for the result hubsize or directed	• Popover is more interactive. • On writing COL there is auto-suggestion • Different output types graph, table, text or code

Table 4.3: NeoVis vis Neo4j in terms of output results

NeoVis	**Vis.js**
• Connect directly with Neo4j • You can query data easily • You can control the thickness of edge depending on a relationship property • You can control the size of node depending on a node property • You can Specify node property for Clustering • Can make hierarchical for the result hubsize or directed • Console debug error messages are meaningful and easy to understand, you need to enable it.	• Supports configuring popovers • Supports three dimension graphs. • Has more graph visualization features like more custom shapes, styles, colors, sizes, images, and more. • Support adding more interaction features to nodes like handling mouse and touch events as well as the navigation buttons and the pop-ups. • Widely used and lots of different online examples are available.

Table 4.4: NeoVis vis Vis.js

Now lets test Neovis on Game of thrones database. You will find the code with the description for the resulted graph in the appendix A.3 page 49. In 4.1 is the graph of Game of throne database. We can see the clustering as each character with in the same community has the same node color. We can also see the different node sizes according to a node property. The thickness of the edges also differ according to a relation property. In 4.2 is the same graph but with enabling hubsize hierarchy and we can see the popover[1] with the properties of the node on moving cursor on the node. There is also another technology with embeddable neo4j connection called Popoto.js, if you want to know more about it check the appendix section A.2 page 44.

[1]**Popover** is a pop-up box that appears when the user clicks on an element, Can be used to show properties of a node when you click on it.

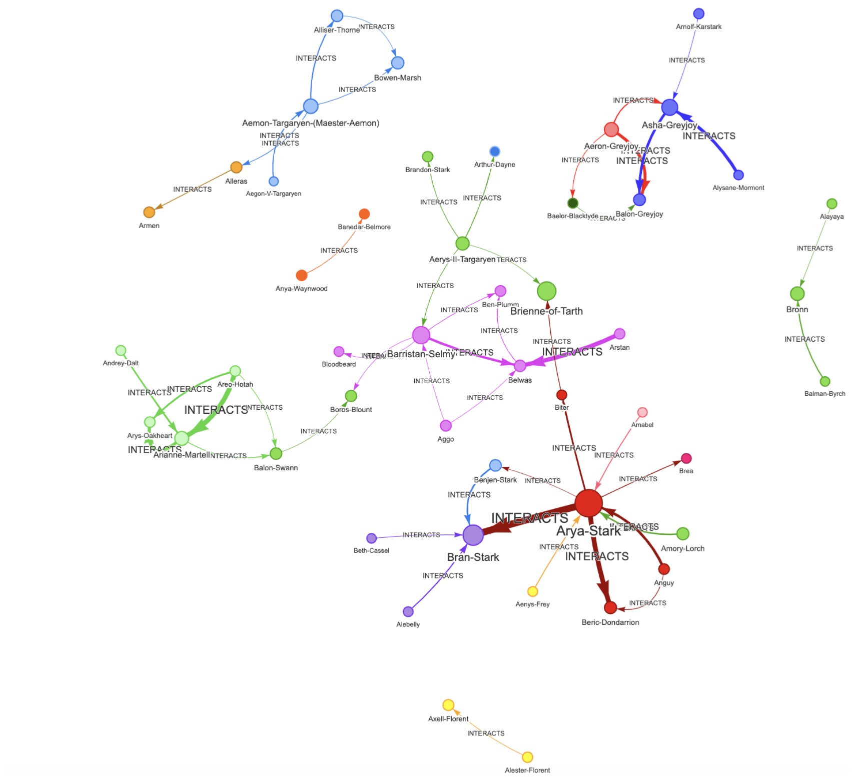


Figure 4.1: Game of throne graph database

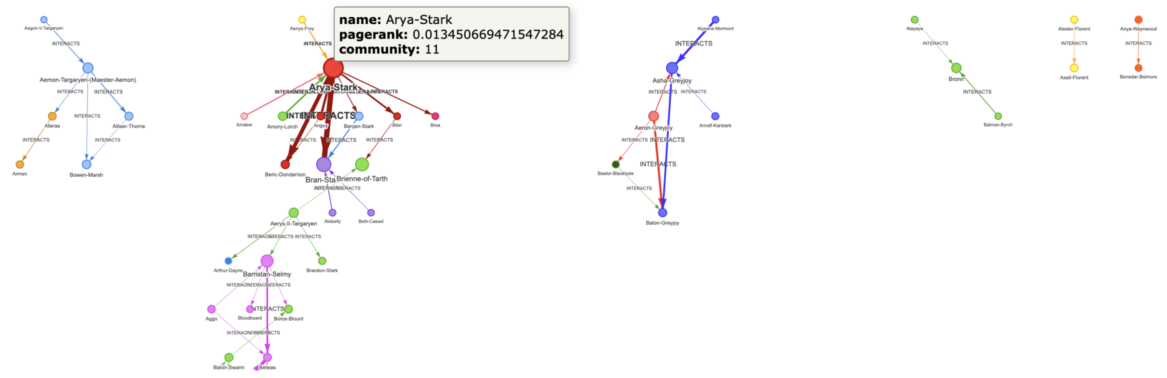

name: Arya-Stark
pagerank: 0.013450669471547284
community: 11

Figure 4.2: The same graph as 4.1 with enabling hubsize hierarchy

Chapter 5

D3

In this chapter, we will talk about D3.js and some of its graph layouts. We will talk about Force directed graphs, Collapsible Force Layout, Tidy Tree and Cluster Dendrogram. At the end there will be comparison table between D3.js and Neovis.

D3.js (Data-Driven Documents) is an open source javascript library for creating visual dynamic and interactive data visualizations in web browser using **Html**, **Css** and **Svg**[1] [6]. D3.js is embedded within an **HTML** web-page. It uses pre-built javascript functions to select elements, creates *SVG* objects, style them, or add transitions, dynamic effects or tooltips to them [6]. *D3.js* makes you have great control over the final visual result. *Neo4j* movie example uses d3.js. For graphs the best thing about D3.js that it offers different layouts for graphs the nearest layouts for our needs is **Force directed graph** and **Collapsible Force Layout** but there is also different layouts that D3.js supports that will be better for visualizing some special cases. In order to apply the D3.js layouts the procedures are nearly the same in all. First you need to initiate some configurations, pass data and then the layout function calculates the necessary positions that‘s it. We will talk about four of D3.js, Force directed graphs, Collapsible Force Layout, **Tidy Tree** and **Cluster Dendrogram**.

5.1 Force directed graphs

Force directed graph is one of D3.js layouts for visualizing graphs. It is network graph consists of nodes and links. In order to create Force directed graph, first we need to set the configuration. Positioning of nodes is simulated by forces using **D3-force**. D3-force is a module that uses Verlet integration [2] for simulating physical forces on nodes [39]. Forces are simply functions that modifies nodes positions or velocities. You can apply

[1]**SVG** is an XML-based vector image (computer graphics images that are defined in terms of 2D points) format for two-dimensional graphics with support for interactivity and animation

[2]**Verlet integration** is a numerical method used to integrate Newton's equations of motion [39]

classical physical force like gravity or you can apply forces using geometric constraints like keeping the nodes within a certain bounding box [39]. There is a link force that pushes linked nodes together or a part according to your desired distance for more different kind of forces that D3-force support you need to check this cite [39]. Second, we need to pass the data. For data we need to pass it in an array of nodes, and a separate array of links. We can connect between the nodes by links in two ways. The links can be reference of the source and target nodes or the index of the nodes in the array. Then we pass the nodes and links arrays into the layout function.

In 5.1, we see an example for Force directed graph on simple database of species and animals. As we can see in 5.1 the clustering feature, the species have red color and the animals have gray color. In 5.2, we see the effect of clicking on mammal node, all of the nodes that are not connected to mammal node disappear and the colors change. This is the power of D3.js it makes you play around with different visualizing features. You will find the code with the description for the resulted graph in the appendix A.4 page 50.

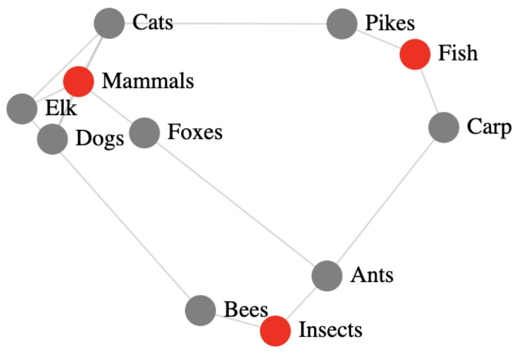


Figure 5.1: An example for Tidy tree in D3.js [8]

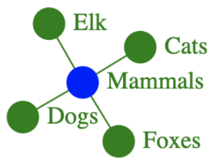


Figure 5.2: An example for Cluster Dendrogram in D3.js [7]

5.2 Collapsible Force Layout

Collapsible Force Layout is one of D3.js layouts for visualizing graphs. It is same as Force directed graphs but with more feature. The internal nodes are collapsible which means, if you click on an inner node the outer nodes coming out of this node will disappear. I see that it is really nice feature for optimizing graph. In figures 5.3 and 5.4, we can see the effect of clicking on a node has inner nodes.

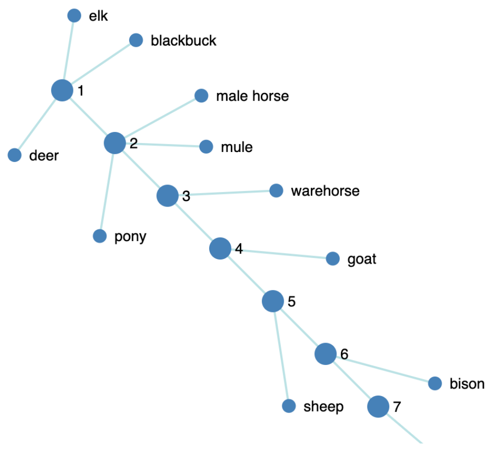


Figure 5.3: The graph is like any normal graph with nodes and links in 5.3, we will see the effect of clicking on node 5 [4].

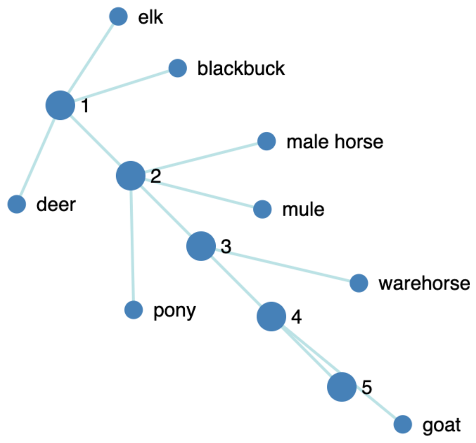


Figure 5.4: After clicking on node 5 all its inner nodes disappeared [4].

5.3 Tidy Tree and Cluster Dendrogram

Tidy tree in D3.js is considered to be one of D3.js hierarchical layouts. It is implemented by ReingoldTilford tidy algorithm [30]. It is consists of nodes and links also but, it shows the relation between nodes in parent-child concept. We have the root node which represents the starting point, comes under it its children and each child is a parent to other nodes and so on.
Cluster Dendrogram is also one of D3.js hierarchical layouts. It has the same structure as Tidy tree but the difference between them is that in Cluster dendrogram all of the leaves are on the same level. In 5.5 and 5.6, we can see the difference between tidy tree and cluster dendrogram structure notice that the leaves of the tree does not have to be on the same level.

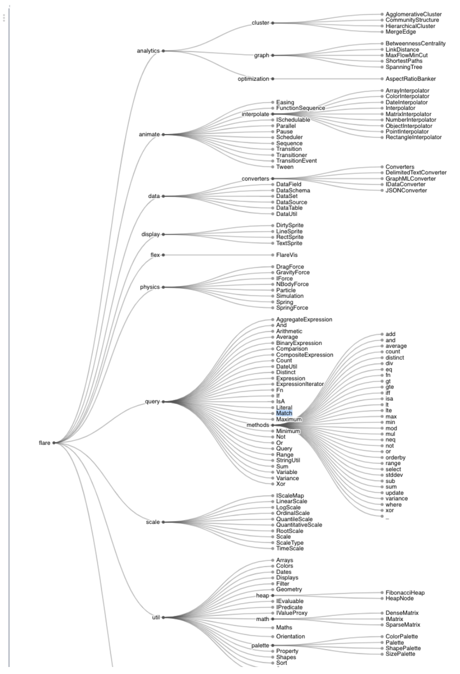


Figure 5.5: An example for Tidy tree in D3.js [8]

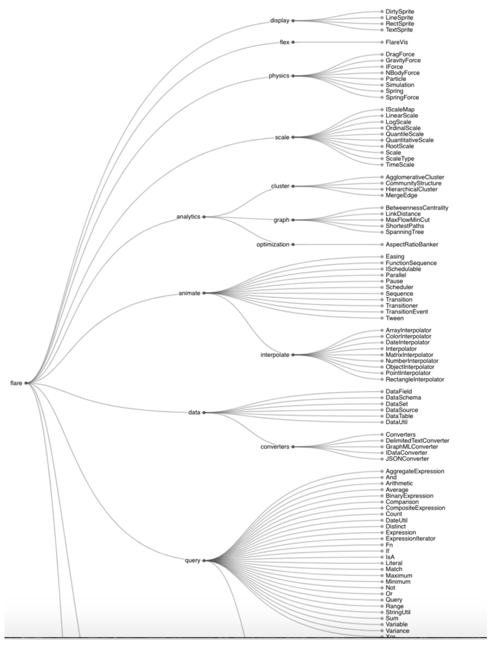


Figure 5.6: An example for Cluster Dendrogram in D3.js [7]

In table 5.1, we will discuss the pros and cons of D3.js.

Pros	**Cons**
• Support different graph visualizing layouts. • Support different styling features. • Support different types of forces for nodes and links. • Specify relation thickness and node size depending on their properties. • Widely used and lots of different examples available online • The documentation is clear	• It does not have direct connection with Neo4j. • You need to explicitly define everything for the graph • User can not query the data or interact with the graph except moving the nodes and relations.

Table 5.1: Pros and cons of D3.js [6]

In table 5.2, we will compare between D3.js and NeoVis.

D3.js	**NeoVis**
• Support different graph visualizing structures • Support different styling features like assigning images for nodes • Support different types of forces for nodes and links. • Widely used and lots of different examples available online • The documentation is clear	• Connect directly with Neo4j • You can query data easily

Table 5.2: D3.js vis NeoVis [6]

Chapter 6

Recommendation

In this thesis our main concern is regarding a new data management technology, graph databases. Despite how powerful this technology is, visualizing big data graphs is a challenging issue. Humans cannot understand massive graphs as it will be confusing for them. We tried to look at the problem from two different perspectives. First, by asking ourselves what is the easiest way to represent the big data in so users can easily analyze it and compare the results. We found no better answer than representing it in charts. Second, is asking ourselves why visualizing big data graphs is difficult for humans. For this question we used some Gestalt principles to help us understand how the human perception is affected by a number of visual factors. There are some tools, applying some of these principles, that we can use to add more features to the graphs, making it easier for users to understand.

6.1 First solution

Charts are powerful for presenting data especially for statistical ones. Instead of representing the data in graphs, we will represent it in charts, so user can easily analyze it. We talked about four different python libraries for plotting graphs.

- **Matplotlib** is a powerful plotting library. It is static, so users can not interact with the resultant graph. Matplotlib gives you full control over the tiniest details of the chart, but you will need lots of manual work.
- **Seaborn** is built on top of Matplotlib, it is simpler than Matplotlib and is also a static visualization library. Although Seaborn is simpler than Matplotlib, it is quite limited. If you want to edit its defaults you will need to have some knowledge of Matplotlib.
- **Bokeh** is an interactive visualization library allowing users to interact with the resultant graph. Bokeh supports more than one programming language, not just python. The problem with Bokeh is that it also requires some manual work.

- **Plotly.py** is an interactive visualization library too. Plotly also supports different programming languages, but it is the easiest one compared to other visualization libraries.

If you check table 3.6, you will notice that none of the libraries support all of the charts. There is no best library to use that depends on your needs. However, if you need the fastest and easiest library with interactive option for basic charts, Plotly.py will be your best choice. If you are a master in coding and eager about tiny details concerning the design of charts without caring about if it is interactive or not, your best choice is Matplotlib.
I suggest creating a tool that combines the pros of all four libraries while having a direct built-in connection to Neo4j because all of the four technologies need some manual work to get the axes data manually from Neo4j.

6.2 Second solution

Representing data in charts allow us to easily analyze it. However, not all types of data can be represented on charts. At some point, we will need to see the full graph. There are different tools that add more features to the graph making it more understandable for users. We talked about different tools that will add more features to the graph based on some of Gestalt's principles so user can easily understand the resultant graph.

- **Neovis.js** is an embeddable tool with built-in Neo4j connection. It connects with Neo4j instantly, so you will just need to add some couple of code lines to connect with neo4j and get live data. It helps you in specifying edge property for edge thickness, node property for node size and node property for clustering. It is a graph visualization tool powered by Vis.js but it does not inherit all graph visualization features from Vis.js like, configuring popover, assigning URL images to the nodes and adding more interaction features like navigation buttons, handling mouse events and so on.

- **D3.js** is an embeddable library without direct Neo4j connection. It supports widely different styling features, has different graph visualizing layouts and supports different types of forces for nodes and links to make the network more realistic. The problem with D3.js that you need lots of manual work to draw and define everything for the graph as it has no direct connection with Neo4. You also can not query data in D3.js, you will need to get the resultant graph first from Neo4j and manually design it on D3.js.

I recommend to use D3.js as nowadays, it is widely used and it has much more features and wide types of forces that you can add to the graph making it more realistic, so it will be more understandable for users. If Neovis inherits more graph visualization features from Vis.js and supports adding different forces in the graph, it will be the best option because it has direct connection with Neo4j, so we can directly query data from Neovis.

Chapter 7

Conclusion

Graph databases enable us to make new discoveries in different fields. Graph theory was introduced as a solution for Seven Bridges of Knigsberg problem by Euler as it helped him to see the full image of the problem, this shows how powerful graphs are. The only problem with graph database is that when we apply it on a big data the resultant graph will be so huge, and that is difficult even on programmers to understand. In this thesis, our aim was to find a solution for this issue. We introduced two different solutions.

The first solution is, instead of visualizing big graphs, we can represent results on charts so users can easily understand and analyze them. We talked about four different python plotting libraries Matplotlib, Seaborn, Bokeh and Plotly.py. Each one of them has its own advantages and your choice depends on your needs as, the four libraries do not support all types of charts. Check tables 3.6 and 3.5 to aid you in choosing the best library that suits your needs. However, the easiest interactive plotting library with the least manual work of them is Plotly.py.

In the second solution we avoid using charts to represent the results because not all types of data can be represented on charts. We will use different tools that generate graphs that are visually understandable. We talked about three of these tools Neovis.js, D3.js and Vis.js. Although, Neovis is the easiest option as it has built-in Neo4j connection, using Vis.js or D3.js is better because they have way wider graph visualization features than neovis.

In conclusion, there is no solution better than the other and it all depends on the type of the data. However, if we can visualize the full graph, that will always be the better approach as we can identify hidden relationships allowing us to unravel great mysteries in our world.

Appendix

Appendix A

Codes of the examples and Popoto.js

A.1 Plotly.py Example

This code is implemented using jupyter-Notebook all the Queries are made on the large movie database on neo4j. The Movie Database is a traditional data-set for graph databases, similar to IMDB [1]. it contains actors, directors that are related through movies they collaborated in together, also users who rated this movies. The next code is the code of the first Query which is get The number of produced movies each year. Explanation is provided as comments in the code.

```python
#import pacakges and set enviroment
from neo4j import GraphDatabase
import pandas as pd
import numpy as np
import numpy as np
%matplotlib inline
import plotly as py
import plotly.graph_objs as go
import datetime
py.offline.init_notebook_mode(connected=True)
%config IPCompleter.greedy=True

#Create intiate driver for your database
url = "bolt://localhost:7687"
driver = GraphDatabase.driver(url, auth=("neo4j", "123456"))
#First Query No of Movies per year

nodeType="Movie"
nodeA1="m.title"
nodeA2="m.releaseDate"
query1="Match(m:"+nodeType+")return "+nodeA1+","+nodeA2

```

[1]**IMDB** IMDb (Internet Movie Database) is an online database of information related to movies, television shows and actors [36]

```
#intiate session to run the query
with driver.session() as graphDB_Session:
        nodes = graphDB_Session.run(query1)

#nodeA1A2 -> list of [Movie name, year of release]
nodeA1A2=[]

#in the movies data base there is a mistake in the date of release for
   Rodan movie
#filling nodeA1A2 list
for node in nodes:
    if(type(node[nodeA2])==str):
        hellNo=datetime.datetime.fromtimestamp(int(node[nodeA2])/1000).
   year
        if(node[nodeA1]=='Rodan'):
            nodeA1A2.append([node[nodeA1],1956])
        else:
            nodeA1A2.append([node[nodeA1],hellNo])

#Sort the list according to year of release
nodeA1A2.sort(key=lambda x: x[1])

#x-> x Axis data, y->Y Axis data
x=[]
y=[]
i=nodeA1A2[0][1]
j=0

#filling x and y lists with count in y and year in x
while(i<=nodeA1A2[len(nodeA1A2)-1][1]):
    count=0
    while(j<len(nodeA1A2)and i==nodeA1A2[j][1]):
        count+=1
        j+=1
    x.append(i)
    y.append(count)
    i+=1

#This part for plotting on plotly The Layout of the plot
layout=go.Layout(
    title="First Query",
    yaxis=dict(title='Count'),
    xaxis=dict(title='Year'))



#Type of the graph "Line"

trace1=go.Scatter(
    x=x,
    y=y,
    mode='lines'

)

#Create the figure with the layout and the data of x axis and y axis
fig=go.Figure(data=[trace1],layout=layout)
py.offline.iplot(fig)
```

Lets add some couple of lines to see what is the difference between static data visualization graphs and interactive data visualization graphs.We will use the same code but we will add couple of lines instead of plotting on Plotly.py part

```
import seaborn as sns; sns.set()
import matplotlib.pyplot as plt
df=pd.DataFrame()
df['Year']=x
df['Count']=y
ax = sns.lineplot(x="Year", y="Count", data=df)
```

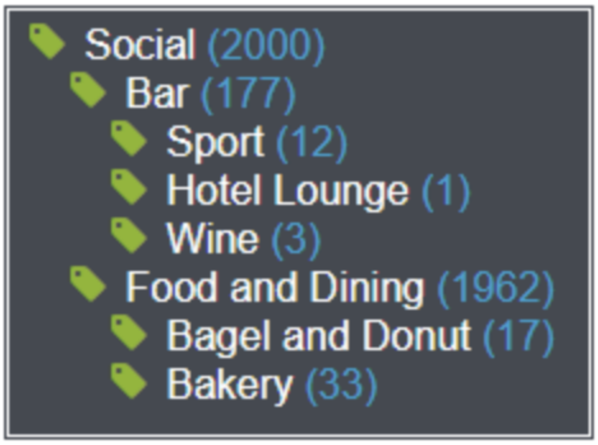
A.2 Popoto.js

Popoto.js is a JavaScript library built upon D3.js released on 8Th of March 2018. *Popoto.js* is designed to create interactive and customized visual query builder for *Neo4j* graph databases [12]. Popoto.js is so simple and easy for users who does not have a background about Cypher or graph database, The user just needs to know about simple ways to generate graph queries which will be translated into Cypher and run on the database. First we need to know that the library consists of various components. Popoto.js five components are [13]

- **Graph Component** Is an interactive interface designed to build queries for users. The Graph is made of nodes and links connects it.

- **Toolbar** Is a list of actions available in the graph container.

- **Taxonomy** It is generated from the node configuration and display all searchable labels, The label can be displayed hierarchical if parent and children properties are defined in node configuration.



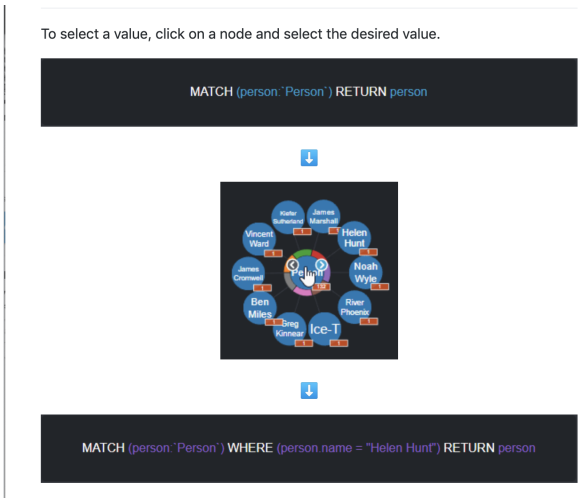
Social (2000)
Bar (177)
Sport (12)
Hotel Lounge (1)
Wine (3)
Food and Dining (1962)
Bagel and Donut (17)
Bakery (33)

Figure A.1: Example for Taxonomy container [13]

- **Query** Displays a text or Cypher representation of the graphical query.
- **Result** Displays the results matching the graph query.

Now lets know some basic actions that will generate graph queries [13]

- **Select value** in order to select a value the user just need to click on the desired node, and then select the desired value.



To select a value, click on a node and select the desired value.

```
MATCH (person:`Person`) RETURN person
```

```
MATCH (person:`Person`) WHERE (person.name = "Helen Hunt") RETURN person
```

Figure A.2: As we see when the user click on the node *Match(person:Person)return person* will be generated and when he select the desired value Match(person:Person)where (person.name=Helen Hun)t return person will be generated [13]

- **Remove Section** in order to remove right click on the node with the desired value, if the node has multiple values the last one will be removed, for example if we want to remove a node of label person right click on person if person has multiple nodes it will delete the last one added

- **Add Relation** all of the relations will be around the node just click on the desired one.

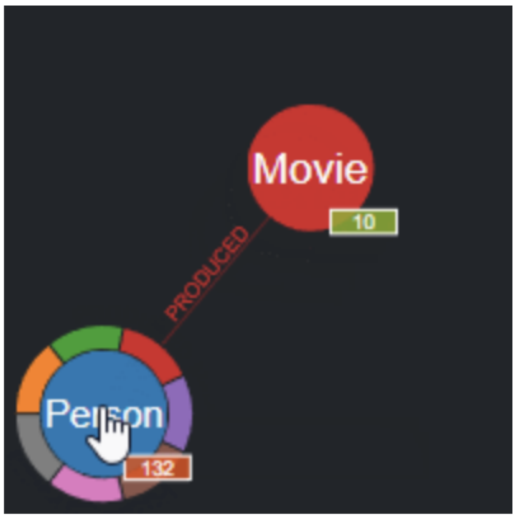


Figure A.3: Around the node different relations just click the one you want and The relation the will be added [13]

- **Delete Relation** In order to delete relation just click on the link corresponding to the relation

- **Create and constraints** To perform and between two value, add several relations and choose the value for each relation added

- **Create or Constraints** Same as *Create and constraints* but instead of creating more than one relation just create one with different values

- **Next value** To navigate through different values of a node, click on the arrow it will take you to the next/previous values

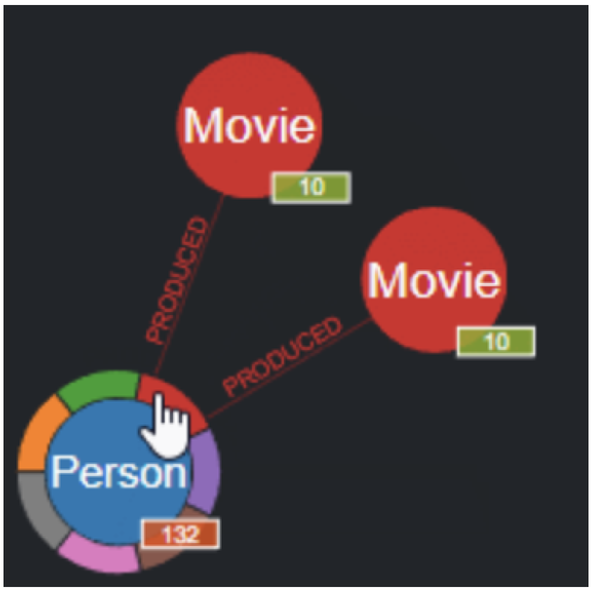


Figure A.4: If you want to and between two values click twice on the relation if you want to and between more than two click more [13]

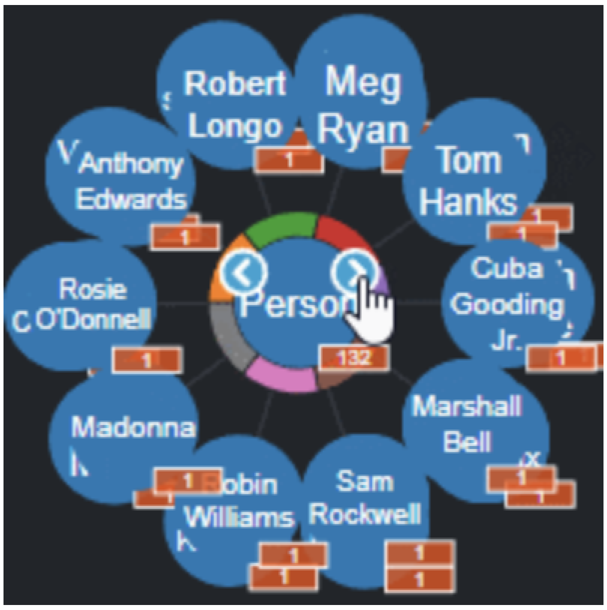


Figure A.6: As we can see by clicking the arrows you can navigate through the values of the desired node [13]

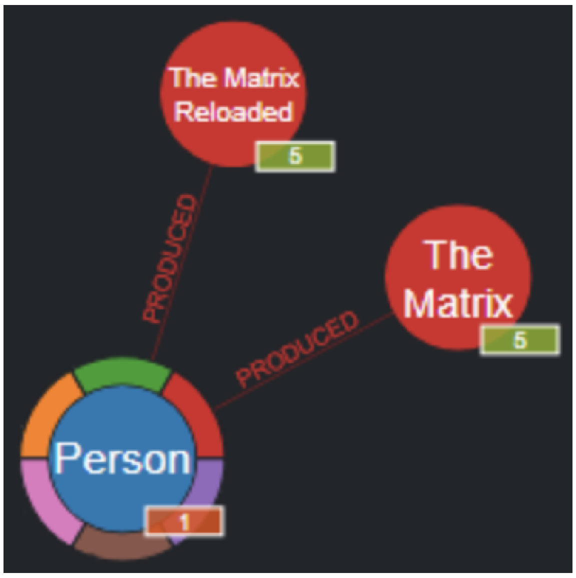


Figure A.5: Assign desired values for the created relations [13]

Pros	Cons
• Simple to use and easy to learn • Connects to Neo4j • No need to learn Cypher in order to query database	• Documentation made me feel lost needs to be more simple with defining the html code steps • There is no enough examples available online • Can not specify node property for URL of image for node

Table A.1: Pros and cons of *Popoto.js* [13]

A.3 NeoVis example

In order to use Game of Thrones database you need to open neo4j create new graph and run the following block of codes separately in Neo4j.
First bock is for Uploading characters and creating relationship with property weight.

```
LOAD CSV WITH HEADERS FROM "https://raw.githubusercontent.com/
   mathbeveridge/asoiaf/master/data/asoiaf-all-edges.csv" AS row
MERGE (src:Character {name: row.Source})
MERGE (tgt:Character {name: row.Target})
MERGE (src)-[r:INTERACTS]->(tgt) ON CREATE SET r.weight = toInteger(row
   .weight)
```

Second setting community property for each character to use it in clustering, setting pagerank property to use it for node size.

```
LOAD CSV WITH HEADERS FROM "https://raw.githubusercontent.com/
   johnymontana/neovis.js/master/examples/data/got-centralities.csv" AS
     row
MATCH (c:Character {name: row.name})
SET c.community = toInteger(row.community),
    c.pagerank  = toFloat(row.pagerank)
```

To visualize the full graph.

```
MATCH p=(:Character)-[:INTERACTS]->(:Character)
RETURN p
```

NeoVis code with description.

```
<html>
  <head>
    <title>Graph Visualization Vis.js</title>
    <script src="https://rawgit.com/neo4j-contrib/neovis.js/master/dist
   /neovis.js"></script>
    <script
          src="https://code.jquery.com/jquery-3.2.1.min.js"
          integrity="sha256-hwg4gsxgFZhOsEEamdOYGBf13FyQuiTwlAQgxVSNgt4
   ="
          crossorigin="anonymous"></script>
  </head>
  <script>
    function draw(){
      //1-Connect to Neo4j instance to get live data
    var config= {
      container_id:"viz",
      server_url:" bolt://localhost:7687",
      server_user:"neo4j",
      server_password:"1234567",
      //2-User specify labels and properties to be displayed
      labels:{
        // 6-Specify node property for node size, Node size is proportional to the Character's pagerank score
        "Character":{
          "caption":"name",
          "size":"pagerank",
          //5-Specify node property for community / clustering,  Node color will be determined by the community property in Character
          "community":"community"
        }
      },
      relationships: {
        //4-Specify edge property for edge thickness, Thickness will be proportional to the weight property on the INTERACTS relationship.
        "INTERACTS": {
          "thickness": "weight",
          "caption": true }
      },
      //3-User specify Cypher query to populate
      initial_cypher:"MATCH (n)-[r:INTERACTS]->(m) RETURN * limit 50",
      arrows:true,
        // Can make hierarchical for the result, The default is directed
        hierarchical:true,
        hierarchical_sort_method:"hubsize",
        //Help u in debugging errors, Error messages are easy to understand
        console_debug:true
      }
      viz = new NeoVis.default(config);
            viz.render();
            console.log(viz);
        }
    </script>
</head>
<body onload="draw()">
<div id="viz"></div>
Cypher query: <textarea rows="4" cols=50 id="cypher"></textarea><br>
<input type="submit" value="Submit" id="reload">

</body>
<script>
    $("#reload").click(function() {
        var cypher = $("#cypher").val();
        if (cypher.length > 3) {
            viz.renderWithCypher(cypher);
        } else {
            console.log("reload");
            viz.reload();
        }
    });
</script>
</html>
```

A.4 D3.js example

Force directed graph layout code with description.

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript" src="http://d3js.org/d3.v2.js"></script>
<style type="text/css">
.link { stroke: #ccc; }
.nodetext { pointer-events: none; font: 10px sans-serif; }
</style>
</head>
<body>
<h3><a href="http://stackoverflow.com/questions/10899725/d3-js-force-
    directed-graph-with-support-for-drag-and-drop-to-make-selected-node"
    >http://stackoverflow.com/questions/10899725/d3-js-force-directed-
    graph-with-support-for-drag-and-drop-to-make-selected-node</a></h3>
<script type="text/javascript">

var w = 960,
    h = 500

var vis = d3.select("body").append("svg:svg")
    .attr("width", w)
    .attr("height", h);

d3.json("graph.json", function(json) {
    var force = self.force = d3.layout.force()
        .nodes(json.nodes)
        .links(json.links)
        .gravity(.05)
        .distance(100)
        .charge(-100)
        .size([w, h])
        .start();

    var link = vis.selectAll("line.link")
        .data(json.links)
        .enter().append("svg:line")
        .attr("class", "link")
        .attr("x1", function(d) { return d.source.x; })
        .attr("y1", function(d) { return d.source.y; })
        .attr("x2", function(d) { return d.target.x; })
        .attr("y2", function(d) { return d.target.y; });

    var node_drag = d3.behavior.drag()
        .on("dragstart", dragstart)
        .on("drag", dragmove)
        .on("dragend", dragend);

    function dragstart(d, i) {
        force.stop() // stops the force auto positioning before you start dragging
    }

    function dragmove(d, i) {
        d.px += d3.event.dx;
        d.py += d3.event.dy;
        d.x += d3.event.dx;
        d.y += d3.event.dy;
        tick(); // this is the key to make it work together with updating both px,py,x,y on d !
    }

    function dragend(d, i) {
        d.fixed = true; // of course set the node to fixed so the force doesn't include the node in its auto positioning stuff
        tick();
        force.resume();
    }


    var node = vis.selectAll("g.node")
        .data(json.nodes)
      .enter().append("svg:g")
        .attr("class", "node")
        .call(node_drag);

    node.append("svg:image")
        .attr("class", "circle")
        .attr("xlink:href", "https://github.com/favicon.ico")
        .attr("x", "-8px")
        .attr("y", "-8px")
        .attr("width", "16px")
        .attr("height", "16px");

    node.append("svg:text")
        .attr("class", "nodetext")
        .attr("dx", 12)
        .attr("dy", ".35em")
        .text(function(d) { return d.name });

    force.on("tick", tick);

    function tick() {
      link.attr("x1", function(d) { return d.source.x; })
```

```
          .attr("y1", function(d) { return d.source.y; })
          .attr("x2", function(d) { return d.target.x; })
          .attr("y2", function(d) { return d.target.y; });

      node.attr("transform", function(d) { return "translate(" + d.x +
   "," + d.y + ")"; });
    };


});

</script>
</body>
</html>
```

Appendix B

Lists

List of Figures

List of Tables

Bibliography

[1] M Ali-ud-din Khan, Muhammad Uddin, and Navarun Gupta. Seven v's of big data understanding big data to extract value. In *Proceedings of the 2014 Zone 1 Conference of the American Society for Engineering Education*, pages 1–5. IEEE, 04 2014.

[2] Michele Grimaldi Andrea De Mauro, Marco Greco. A formal definition of Big Data based on its essential features. *Library Review*, 65(13):122–135, 2016.

[3] Ashish. Evolution to graph databases and neo4j. https://medium.com/@ashish_fagna/evolution-to-graph-databases-and-neo4j-486a18914eb3, July 2018. [Online; accessed 26-July-2019].

[4] Ajs Block. Sample 4(collapsible force layout w/json file). https://bl.ocks.org/ajmistu/749df84a81441218e2ead3301a2a2e5c, June 2017. [Online; accessed 27-July-2019].

[5] Bokeh Development Team. *Bokeh: Python library for interactive visualization*, 2014. [Online; accessed 27-July-2019].

[6] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, December 2011.

[7] Mike Bostock. Cluster dendrogram. http://tiny.cc/lo2baz, December 2017. [Online; accessed 27-July-2019].

[8] Mike Bostock. Tidy tree. https://observablehq.com/@d3/tidy-tree, December 2017. [Online; accessed 27-July-2019].

[9] Dr Edgar F. Codd. A Relational Model of Data for Large Shared Data Banks . *Magazine Communications of the ACM*, 13:377–387, 06 1970.

[10] Guru99 community. Relational data model in dbms: Concepts, constraints, example. https://www.guru99.com/relational-data-model-dbms. [Online; accessed 26-July-2019].

[11] Neo4j community. Cypher query language. http://tiny.cc/y5xiaz. [Online; accessed 26-July-2019].

[12] Neo4j community. Graph visualization tools. http://tiny.cc/8n2baz. [Online; accessed 27-July-2019].

[13] Popoto.js community. Popoto.js. https://github.com/Nhogs/popoto/wiki, May 2018. [Online; accessed 27-July-2019].

[14] Seaborn community. An introduction to seaborn. https://www.tutorialspoint.com/seaborn/. [Online; accessed 26-July-2019].

[15] Vis.js community. Vis.js. https://visjs.org/. [Online; accessed 27-July-2019].

[16] S. Gill Williamson Edward A. Bender. Lists, decisions and graphs. *University of California at San Diego*, 12 2010.

[17] Harris Eisenberg. Humans process visual data better. http://www.t-sciences.com/news/humans-process-visual-data-better, September 2015. [Online; accessed 27-July-2019].

[18] Stephen Few. Data visualization for human perception. http://tiny.cc/it1baz. [Online; accessed 27-July-2019].

[19] Janis Gulbis. How to pick the right chart type. https://eazybi.com/blog/data_visualization_and_chart_types/, March 2016. [Online; accessed 26-July-2019].

[20] Jeff Hale. Its 2019 make your data visualizations interactive with plotly. http://tiny.cc/230baz, October 2019. [Online; accessed 26-July-2019].

[21] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[22] Plotly Technologies Inc. Collaborative data science. https://plot.ly, 2015. [Online; accessed 26-July-2019].

[23] Johnymontana. Neovis.js. https://github.com/neo4j-contrib/neovis.js/. [Online; accessed 27-July-2019].

[24] Will Koehrsen. Introduction to interactive time series visualizations with plotly in python. http://tiny.cc/c60baz, December 2018. [Online; accessed 26-July-2019].

[25] Will Koehrsen. The next level of data visualization in python. http://tiny.cc/c10baz, January 2019. [Online; accessed 26-July-2019].

[26] Douglas Laney. 3D Data Management: Controlling Data Volume, Velocity, and Variety. *META Group Research IEEE*, 4:1–3, 2001.

[27] Ripon Patgiri and Arif Ahmed. Big data: The vs of the game changer paradigm. In *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 17–24. IEEE, December 2016.

[28] Turtorials point community. Relation data model. https://www.tutorialspoint.com/dbms. [Online; accessed 26-July-2019].

[29] Sunil Ray. Interactive Data Visualization using Bokeh. http://tiny.cc/b20baz, August 2015. [Online; accessed 26-July-2019].

[30] E. M. Reingold and J. S. Tilford. Tidier drawings of trees. *IEEE Trans. Softw. Eng.*, 7(2):223–228, March 1981.

[31] Bryce Merkl Sasaki. Why graph technology is the future. https://neo4j.com/blog/why-graph-databases-are-the-future/, July 2018. [Online; accessed 26-July-2019].

[32] Brad Solomon. Python plotting with matplotlib (guide). https://realpython.com/python-matplotlib-guide/. [Online; accessed 26-July-2019].

[33] Michael Waskom, Olga Botvinnik, Drew O'Kane, Paul Hobson, Joel Ostblom, Saulius Lukauskas, David C Gemperline, Tom Augspurger, Yaroslav Halchenko, John B. Cole, Jordi Warmenhoven, Julian de Ruiter, Cameron Pye, Stephan Hoyer, Jake Vanderplas, Santi Villalba, Gero Kunter, Eric Quintero, Pete Bachant, Marcel Martin, Kyle Meyer, Alistair Miles, Yoav Ram, Thomas Brunner, Tal Yarkoni, Mike Lee Williams, Constantine Evans, Clark Fitzgerald, Brian, and Adel Qalieh. mwaskom/seaborn: v0.9.0 (july 2018). https://seaborn.pydata.org/, July 2018. [Online; accessed 26-July-2019].

[34] Wikipedia contributors. Big data. https://en.wikipedia.org/w/index.php?title=Big_data&oldid=907766383, 2019. [Online; accessed 26-July-2019].

[35] Wikipedia contributors. Codd's 12 rules. https://en.wikipedia.org/w/index.php?title=Codd%27s_12_rules&oldid=906795138, 2019. [Online; accessed 26-July-2019].

[36] Wikipedia contributors. Imdb. https://en.wikipedia.org/w/index.php?title=IMDb&oldid=907296381, 2019. [Online; accessed 27-July-2019].

[37] Wikipedia contributors. Relational database. https://en.wikipedia.org/w/index.php?title=Relational_database&oldid=908014166, 2019. [Online; accessed 26-July-2019].

[38] Wikipedia contributors. Topology. https://en.wikipedia.org/w/index.php?title=Topology&oldid=903581937, 2019. [Online; accessed 26-July-2019].

[39] Wikipedia contributors. Verlet integration. https://en.wikipedia.org/w/index.php?title=Verlet_integration&oldid=906103946, 2019. [Online; accessed 27-July-2019].