BUILDING KNOWLEDGE GRAPHS WITH PYTHON

Project report submitted in partial fulfillment of the
requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering/Information Technology

By

Ngawang Choega (191507)
Rahul Sharma (191555)

Under the supervision of

Dr. Ravindara Bhatt
to

Department of Computer Science & Engineering and
Information Technology
Jaypee University of Information Technology
Waknaghat, Solan-173234, Himachal Pradesh

CANDIDATE’S DECLARATION

I hereby declare that the work presented in this report entitled **Building Knowledge Graphs with Python”** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from July 2022 to May 2023 under the supervision of **Dr. Ravindara Bhatt** (Associate Professor in Computer Science & Engineering / Information Technology).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

(Student Signature)
Ngawang Choega, 191507

(Student Signature)
Rahul Sharma, 191555

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

(Supervisor Signature)
Dr. Ravindara Bhatt
Associate Professor
Computer Science & Engineering / Information Technology
Dated:

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: …………………………

Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | Paper

Name:__________________________ **Department:**________________ **Enrolment No** ________

Contact No.__________________________ **E-mail.**________________________________

Name of the Supervisor: ____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): __________________

____

____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at……………….(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	**Excluded**	**Similarity Index (%)**	**Generated Plagiarism Report Details** (Title, Abstract & Chapters)	
	• All Preliminary Pages • Bibliography/Images/Quotes • 14 Words String		Word Counts	
Report Generated on			Character Counts	
		Submission ID	Total Pages Scanned	
			File Size	

Checked by
Name & Signature **Librarian**

……

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com

AKCNOWLEDGEMENT

Firstly, I express my heartiest thanks and gratefulness to almighty God for His divine blessing makes us possible to complete the project work successfully.
I am really grateful and wish my profound my indebtedness to Supervisor Dr. Ravindara Bhatt, Associate Professor, Department of CSE/IT Jaypee University of Information Technology, Waknaghat. Deep Knowledge & keen interest of my supervisor in the field of "Research Area" to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stage have made it possible to complete this project.

I would like to express my heartiest gratitude to Dr. Ravindara Bhatt, Department of CSE, for his kind help to finish my project.

I would also generously welcome each one of those individuals who have helped me straight forwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.
Finally, I must acknowledge with due respect the constant support and patience of my parents.
Project Group No.: 101

Student Name: Ngawang Choega
Rollno.: 191507

Student Name: Rahul Sharma
Rollno.: 191555

TABLE OF CONTENT

Title	**Page No.**
Declaration	i
Plagiarism Certificate	ii
Acknowledgement	iii
Table of Content	iv
List of Abbreviations	v
List of Figures	vi
Abstract	viii
Chapter-1 Introduction	1
Chapter-2 Literature Survey	11
Chapter-3 System Development	22
Chapter-4 Performance Analysis	53
Chapter-5 Conclusions	62
References	65

LIST OF ABBREVIATIONS

KG- Knowledge Graph

SVO- Subject Verb Object

SPO- Subject Predicate Object

RDF- Resource Description Framework

AI- Artificial Intelligence

NLP- Natural Language Processing

KGE- Knowledge Graph Embedding

GAN- Generative Adversarial Network

GPU- Graphical Processing Unit

IDE- Integrated Development Environment

DBMS- Database Management System

APOC- Awesome Procedure on Cypher

GDS- Graph Data Science Library

API- Application Programming Interface

NER- Named Entity Recognition

SVM- Support Vector Machine

CLI- Command Line Interface

SQL- Structured Query Language

ML- Machine Learning

TSE- Total Squared Error

LIST OF FIGURES

Fig No.	Title	Page No
1.1	Relationship between entities	1
1.2	NLP considerations for knowledge graph creation	2
3.1	Workflow of the model designing phase	27
3.2	Sample knowledge graph shown in Neo4j platform	29
3.3	Example to explain NER	37
3.4	SVO triples in graph data	40
3.5	Example of a knowledge graph with SVO triples	40
3.6	Workflow diagram of model development phase14	40
3.7	Importing all the required libraries	41
3.8	Adding SVO triples and performing NLP on the graph	42
3.9	Querying Google's knowledge graph using an API key	42
3.10	Helper functions to remove special character	43
3.11	Helper functions to create SVO triples	43
3.12	Wikipedia data of Barack Obama	44
3.13	More helper functions used in the code	45
3.14	Three more helper functions are used in this section	45
3.15	Overview of the workflow in NLP only approach	46
3.16	Fetching out phonetic spellings of target nodes	46
3.17	More helper functions used	47
3.18	Creating lists of tuples representing node and edge list	48
3.19	Creating node list used to populate the graph	48
3.20	Deduping the words for NLP	48
3.21	Connecting Neo4j with google colab	49
3.22	Function to measure cosine similarity between words	49
3.23	Finding out the cosine similarity values	50
3.24	Setting the Neo4j environment to populate the graphs	50
3.25	Getting the total number of nodes	50

3.26	Figure explaining word2vec algorithm	51
3.27	Figure explaining node2vec algorithm	52
4.1	Creating a variety of X variables for the model	53
4.2	Helper function to run SVM	54
4.3	Confusion matrix showing us model performance	54
4.4	Confusion matrix showing us the performance of our model	55
4.5	Confusion matrix showing us the performance of our model	56
4.6	Displacy to get Wikipedia summary paragraph	57
4.7	Relation between source name and target name	58
4.8	Connecting to Neo4	59
4.9	Setting up Neo4j environment	59
4.10	Dimensionality reduction for the sake of visualization	60

ABSTRACT

Data comes in a variety of forms. Data is useful if we are able to extract information from it and it becomes difficult when we are working on a large scale. Here, comes the knowledge graphs. The demands of emergency management are well-suited to the rich, adaptable, and uniform ways that knowledge graphs portray data. They build upon the standards, resources, methods, and methods for semantic data and computation which are already in place. Natural-language texts are a form of data source that would provide unique analytic issues, particularly those gathered via social media like Twitter. Knowledge graphs have been created using Python with tools like Neo4j that employ the Cypher query language for databases. A knowledge graphs is a well labelled graph where the labels have a well-defined meaning. It has many components such as nodes, edges and labels. A common example is the voice assistant and the search engines.

The knowledge graphs help in providing better search results on search engine and voice assistant also works on the natural language processing to understand the query of the person and provide him the best solution possible. Descriptions have formal semantics that enable both humans as well as computers to process them effectively and unambiguously; they build upon one another to form a network in which each entity represents a fraction of the description of the entities related to it; diverse data is connected and defined by semantic links. One entity is a part of another entity's definition. The graph is created by this connecting. Example: A is B, B is C. C has D. A has D. A Q&A "knowledge base" regarding a software product, for example, does not comprise a knowledge base (KG) because it lacks formal structure and semantics. A fundamental idea in discrete mathematics that finds use in all branches of computer technology is the directed labelled graph. In order to fully understand the use of the Cypher query language in the field of AI, we have built knowledge graphs with Neo4j in this project. Visual pattern matching and relationship comparison are made possible via the Cypher query language. It is evident that knowledge graphs have such a promising future in the field of technology innovation.

Chapter-1
INTRODUCTION

1.1 Introduction

A knowledge graph, a type of graph representation, depicts the relationships between items in the actual world, such as objects, occasions, circumstances, or concepts. After storage in a graph database, the data is eventually displayed as a graph structure. The interconnected descriptions of these items are readily stored in these knowledge graphs. Over time, knowledge graphs have proven to be an efficient abstraction for classifying structured knowledge online and offering a means of integrating the data obtained from a variety of different data sources. A knowledge graph is an appropriately labeled graph with labels that have clear definitions. A knowledge graph, which comprises nodes, edges, and labels, has several parts. A node is any entity, such as a person, business, computer, location, or organisation. A network is a connection between two mobile phones, whereas a client relationship is one between a business and a client. The relationship of interest between a pair of nodes is captured by the edge that links them. For instance, a friendship is a relationship between two individuals. The label describes the connection between two nodes, including a friendship, client, or network.

Formally, a subset of the cross product of N * L* N provides us the knowledge graph given a set of nodes N and a set of labels L. Each element of this set, referred to as a triple, can be seen in illustration below.

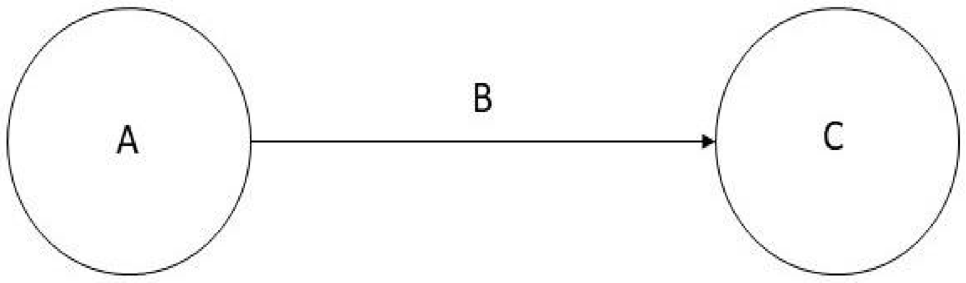
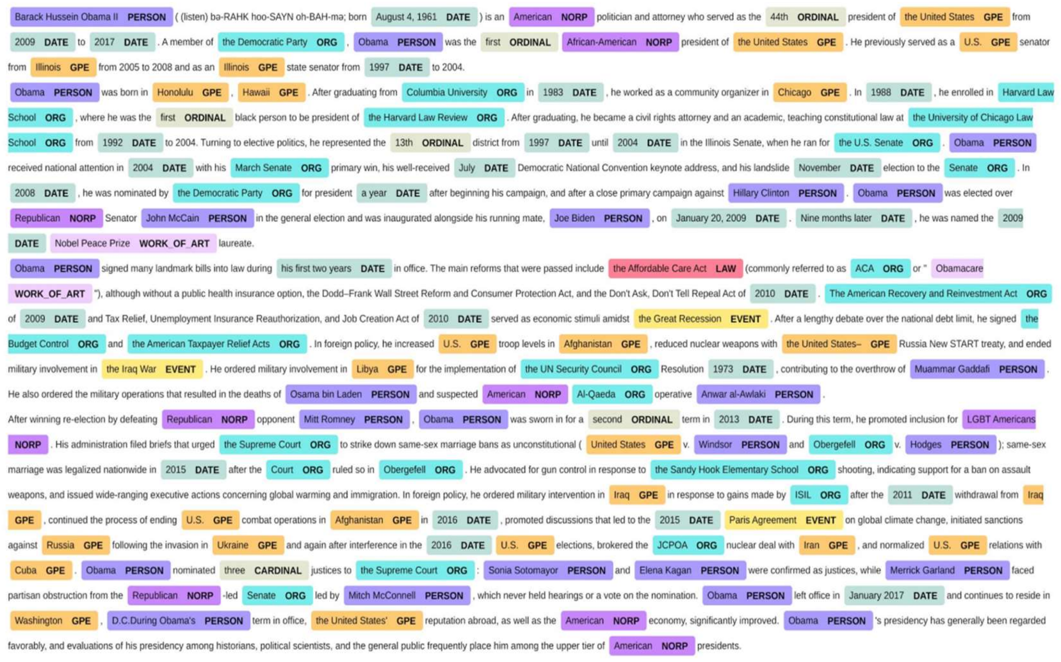


Fig 1.1: Relationship between the two entities

A directed labeled graph is a 4-tuple G = (N, E, L, f), where N is a set of nodes, E ⊆ N × N is a set of edges, L is a set of labels, and f: E→L, is an assignment function from edges to labels. An assignment of a label B to an edge E= (A, C) can be viewed as a triple (A, B, C). The knowledge graph gets richer with every new data added into it. Data, graphs, and semantics all collaborate to create a knowledge graph with rich, dynamic context. A knowledge graph is the best tool for business data integration since it can explain real-world context machines. Data is united through the power of graphs to endlessly link concepts instead of by joining tables to integrate the data. Verbs can be a bit difficult to detect via NLP: Named Entity Recognition (NER) SVO / SPO triples


Barack Hussein Obama II PERSON ((listen) bə-RAHK hoo-SAYN oh-BAH-mə; born August 4, 1961 DATE) is an American NORP politician and attorney who served as the 44th ORDINAL president of the United States GPE from 2009 DATE to 2017 DATE . A member of the Democratic Party ORG , Obama PERSON was the first ORDINAL African-American NORP president of the United States GPE . He previously served as a U.S. GPE senator from Illinois GPE from 2005 to 2008 and as an Illinois GPE state senator from 1997 DATE to 2004.

Obama PERSON was born in Honolulu GPE , Hawaii GPE . After graduating from Columbia University ORG in 1983 DATE , he worked as a community organizer in Chicago GPE . In 1988 DATE , he enrolled in Harvard Law School ORG , where he was the first ORDINAL black person to be president of the Harvard Law Review ORG . After graduating, he became a civil rights attorney and an academic, teaching constitutional law at the University of Chicago Law School ORG from 1992 DATE to 2004. Turning to elective politics, he represented the 13th ORDINAL district from 1997 DATE until 2004 DATE in the Illinois Senate, when he ran for the U.S. Senate ORG . Obama PERSON received national attention in 2004 DATE with his March Senate ORG primary win, his well-received July DATE Democratic National Convention keynote address, and his landslide November DATE election to the Senate ORG . In 2008 DATE , he was nominated by the Democratic Party ORG for president a year DATE after beginning his campaign, and after a close primary campaign against Hillary Clinton PERSON . Obama PERSON was elected over Republican NORP Senator John McCain PERSON in the general election and was inaugurated alongside his running mate, Joe Biden PERSON , on January 20, 2009 DATE . Nine months later DATE , he was named the 2009 DATE Nobel Peace Prize WORK_OF_ART laureate.

Obama PERSON signed many landmark bills into law during his first two years DATE in office. The main reforms that were passed include the Affordable Care Act LAW (commonly referred to as ACA ORG or " Obamacare WORK_OF_ART "), although without a public health insurance option, the Dodd–Frank Wall Street Reform and Consumer Protection Act, and the Don't Ask, Don't Tell Repeal Act of 2010 DATE . The American Recovery and Reinvestment Act ORG of 2009 DATE and Tax Relief, Unemployment Insurance Reauthorization, and Job Creation Act of 2010 DATE served as economic stimuli amidst the Great Recession EVENT . After a lengthy debate over the national debt limit, he signed the Budget Control ORG and the American Taxpayer Relief Acts ORG . In foreign policy, he increased U.S. GPE troop levels in Afghanistan GPE , reduced nuclear weapons with the United States– GPE Russia New START treaty, and ended military involvement in the Iraq War EVENT . He ordered military involvement in Libya GPE for the implementation of the UN Security Council ORG Resolution 1973 DATE , contributing to the overthrow of Muammar Gaddafi PERSON . He also ordered the military operations that resulted in the deaths of Osama bin Laden PERSON and suspected American NORP Al-Qaeda ORG operative Anwar al-Awlaki PERSON .

After winning re-election by defeating Republican NORP opponent Mitt Romney PERSON , Obama PERSON was sworn in for a second ORDINAL term in 2013 DATE . During this term, he promoted inclusion for LGBT Americans NORP . His administration filed briefs that urged the Supreme Court ORG to strike down same-sex marriage bans as unconstitutional (United States GPE v. Windsor PERSON and Obergefell ORG v. Hodges PERSON); same-sex marriage was legalized nationwide in 2015 DATE after the Court ORG ruled so in Obergefell ORG . He advocated for gun control in response to the Sandy Hook Elementary School ORG shooting, indicating support for a ban on assault weapons, and issued wide-ranging executive actions concerning global warming and immigration. In foreign policy, he ordered military intervention in Iraq GPE in response to gains made by ISIL ORG after the 2011 DATE withdrawal from Iraq GPE , continued the process of ending U.S. GPE combat operations in Afghanistan GPE in 2016 DATE , promoted discussions that led to the 2015 DATE Paris Agreement EVENT on global climate change, initiated sanctions against Russia GPE following the invasion in Ukraine GPE and again after interference in the 2016 DATE U.S. GPE elections, brokered the JCPOA ORG nuclear deal with Iran GPE , and normalized U.S. GPE relations with Cuba GPE . Obama PERSON nominated three CARDINAL justices to the Supreme Court ORG : Sonia Sotomayor PERSON and Elena Kagan PERSON were confirmed as justices, while Merrick Garland PERSON faced partisan obstruction from the Republican NORP -led Senate ORG led by Mitch McConnell PERSON , which never held hearings or a vote on the nomination. Obama PERSON left office in January 2017 DATE and continues to reside in Washington GPE . D.C.During Obama's PERSON term in office, the United States' GPE reputation abroad, as well as the American NORP economy, significantly improved. Obama PERSON 's presidency has generally been regarded favorably, and evaluations of his presidency among historians, political scientists, and the general public frequently place him among the upper tier of American NORP presidents.

Fig 1.2: NLP considerations for knowledge graph creation

Barack Hussein Obama II, an attorney and politician from the United States, governed over the nation as its 44th president from 2009 to 2017.

Knowledge graphs combine elements of various data management paradigms:

database, because structured queries may be used to examine the information.

due to the fact that they may be used to investigate any other network data structure.

Due to their formal semantics, knowledge bases may be utilised to interpret data and deduce novel facts. Knowledge graphs are essential to the representation of information extracted using natural language processing. In terms of improving predictions, domain knowledge is fed into machine learning models using knowledge graphs. The finest foundation for data integration, unification, linking, and reuse is provided by knowledge graphs, that are represented in RDF. It's because they combine:

Expressivity: The Semantic Web stack's standards, RDF(S) and OWL, enable the fluent representation of a diverse variety of data and content types, encompassing taxonomies and vocabularies, metadata of every kind, reference data, and master data. It is simple to model provenance and other structured metadata thanks to the RDF extension.

1. Performance: Every specification has been considered carefully and tested to guarantee effective management of graphs containing billions of facts and attributes.
2. Interoperability**:** Data serialisation, access (SPARQL Protocol for end-points), management (SPARQL Graph Store), and federation all have different requirements. Globally unique identifiers make publishing and data integration easier.
3. Standardization: All of the foregoing is defined through the W3C community process to ensure that the needs of many actors, from logicians to experts in enterprise data management and system operations teams, are met.

Ontologies and formal semantics

Ontologies provide the core of the formal semantics of a knowledge graph. They could be thought of as the data schema for the graph. They function as a formal agreement between both the knowledge network's producers and users over the meaning of things inside. A user may be a separate person or a computer programme that must interpret the data in a reliable and correct way. Ontologies preserve a broad knowledge of the meanings associated with the data.

When formal semantics are being used to create and interpret the underlying in a knowledge graph, incorporating classes, there are many representations and modelling tools accessible. The entity description often includes a categorization of the entity in reference to a class hierarchy. When dealing with business data, for instance, there can be classes for Person, Organization, and Location. Agent is a superclass that both organizations and individuals may be using. Typically, there are numerous different subclasses of location, such as country, city, populous place, etc. The concept of class was taken from object-oriented design, where each entity normally belongs to exactly one class. Kinds, which provide details about the nature of the link, such as buddy, related, rival, are frequently used to determine relationships between entities. Formal definitions of relationship types are also possible. For instance, the parent-of connection is the opposite of the child-of relationship and both are particular instances of the symmetric relative-of relationship. or specifying that the words subsidiary and subregion are related in a transitive sense. Categories that describe a particular element of an entity's semantics might be attached to it, such as "Big four consultants" or "XIX century composers." A book may fall under more than one of these categories at once, like "Books about Africa," "Bestseller," "Books by Italian Authors," "Books for Kids," etc. Taxonomy is used to describe and arrange the categories. Free text summaries. A "human-friendly language" description is often included to help explain the design objectives for the entity and enhance search.

1.2 Problem Statement

These days, search and even semantic search are just lacking. In order to comprehend a world that is becoming increasingly complex, users want information that is condensed and simple to consume. This begs for a promising technique of accumulating and presenting data that draws on a pool of facts and information. This essay begins by examining the typical process and difficulties encountered by teams beginning the extremely difficult effort of gathering and arranging all of the current information about the planet. There is little doubt that

Deep Learning is grabbing the show in Silicon Valley. But in reality, it's a completely different creature that nourishes and enriches the daily results recommended by your personal search engine. As a simple click search, many efforts are performed to figure out what your inquiry is about. After all, Google's primary purpose depends on promptly and properly responding appropriately to users' requests. How can you convey to your system at some point to your users—the relationship between machine learning and data mining? or that American linguist Noam Chomsky? What about Mount Makalu's height? And that natural language processing comprises word embedding? Simple: You determine the remedies and inform them of them. Or at least that is Knowledge Graphs' main principle (KG).

These interconnected knowledge units power and improve other backend functionalities in in addition to the straightforward searching and showing of information about entities:

- naming items in context and trying to clean up confusion
- expanding data to boost semantic search
- tying up entities with content and data sources
- Engine for relevant information inputs
- User interfaces concentrating on entities
- Deductive logic

A knowledge graph is self-descriptive since it offers a centralized hub to locate the material and comprehend its meaning. The knowledge graph and the word semantics are linked because the significance of the data is included in graph together with the data. Knowledge graphs contribute value because they provide the following:

1. Context: By integrating various types of information into an ontology and having the flexibility to implement additional derived knowledge as desired, knowledge graphs give context to algorithms. The majority of standard knowledge graphs may leverage many forms of raw data at once.

2. Efficiency: Knowledge graphs enable computational efficiencies for querying stored data, leading in effective use of data for creating insights, once needed entities and relations are accessible.
3. Explainability: By including the meaning of entities that are available inside the graph itself, large networks containing entities and interactions offer answers to the problem of intuitiveness. Knowledge graphs thus become naturally interpretable.Knowledge graphs can be utilized in a variety of scenarios. There are particular use cases for each domain in which the graph is now being built. Google's knowledge graph focuses on search engine marketing and providing people with vital information. If you performed a Google search with the term "interstellar," you would see details on the film as well as suggestions for other space sci-fi flicks that are comparable to it.
There would be some scientific publications about interstellar space or just a dictionary defining the term. Google intends to perceive the context of the user's search query in order to display the most appropriate outcomes first. Technological developments exist that function as voice assistants and provide suggestions. The system is more precise and productive thanks to a knowledge graph.

1.3 Objectives

The knowledge graph, which seek to enhance how people search for information, makes the difficult task of searching and exploring easier since there is a huge amount of information available about a person, entity, or thing in the form of data, sounds, videos, and images.
The idea behind the knowledge graph is therefore to make searching smart enough to easily detect associations between two or more items in order to grasp what a person is looking for and deliver relevant results. Since the knowledge graph uses actual facts and associations to supplement distinctive traits, it can more effectively react to inquiries and help us find what we're looking for since it looks for genuine, real-world items. These facts and data inside the knowledge graph are based on a variety of publicly available data that also presents an

opportunity for linking disparate data to derive insight. For instance, by looking at customer reviews and ratings, websites may use search engine optimization to increase search traffic to their websites and help businesses build their credibility. Machine learning and knowledge representation using knowledge graphs are developing quickly with each passing year. Adopting knowledge graphs may help in representing the entities and relationships with tremendous reliability and explainability as machine learning tools and techniques advance throughout time to do various jobs with enormous accuracy and precision. The quality of the findings and the potential of machine learning techniques may both be boosted by combining knowledge graph with machine learning.

The potential advantages of using knowledge graph into machine learning include the following:

1. Data inadequacy

A significant amount of information must be available in order to train an ML model, however in cases when there is a lack of data, Knowledge Graph can be used to advance training data. As an illustration, you might use the name of the entity from the real training data in place of the name of an entity with a comparable data type. This allows for the creation of numerous examples utilising the knowledge graph.

2. Zero-shot learning

Machine learning algorithms have undoubtedly become cleverer over time, but if some classes of properly labeled data are missing, they will not be able to discern apart two identical objects. This is known as zero-shot learning in machine learning. Knowledge can be a powerful strategy to solve this issue since it allows for the coupling of ML model conclusions with knowledge graph consequences. As an illustration, consider the picture data where the training data's scenario categories are absent.

Explaining the conclusions reached by machine learning models and the inferred descriptions of these predictions is one of the key problems that needs to be solved. Knowledge Graph can solve this problem by mapping the interpretations and explanations to some underrecognized nodes in the graph and encapsulating the decision-making process.

In conjunction to this, the following difficulties can be overcome:
Engineers, developers, and researchers will benefit from understanding and using knowledge graphs more if a consistent set of appropriate applications is used while creating them.
When using knowledge integration and unstructured data together, it might be challenging to determine whether the entities are real.

3. Explicability
Integrated with the real-world entities included in Knowledge Graph as described in the data. Though ML algorithms might be able to fix this, the insights these algorithms produced depended on the nature of the data, and because there are so many different datasets, integrating information becomes very difficult.
Despite this, knowledge is constantly evolving. For instance, a user who uses a knowledge graph to keep track of patient health records may find that the information they have reserved at one point in time is incorrect or inaccurate at a later date. This raises the question of how to accommodate this evolving nature of knowledge.

Businesses are increasingly resorting to knowledge graphs as a more efficient methods of bridging the gap between the data world and the business world. Knowledge graphs, when combined with allied AI technologies like machine learning and natural language processing, are opening up new possibilities for gathering and analysing data and are swiftly emerging as a crucial part of contemporary data systems.

1.4 Methodology

Datasets from many sources, many of which have varied structural properties, are frequently used to build knowledge graphs. Multiple forms of data structures are possible when schemas, identities, and context are combined. Identity categories the underlying nodes appropriately, schemas give the foundation for the knowledge graph, and context establishes the environment in which that knowledge is found. Words having many meanings can be distinguished using these elements. This makes it possible for the tools to distinguish between apple as in brand and apple as a fruit, like Google's search engine algorithm.

Knowledge graphs driven by machine learning employ natural language processing (NLP) to provide a comprehensive representation of nodes, edges, and labels through a process called as semantic enrichment. When data is put through this procedure, knowledge graphs are capable of recognizing individual items and understand the connections between various things. Then, further relevant datasets with a similar nature are compared to this working knowledge and combined with it. Once complete, a knowledge graph enables search and question-answering systems to retrieve and reuse complete answers to particular queries.

Through the establishment of linkages between data points which may not have already been recognized, the data integration efforts focused on knowledge representation also can help in the creation of new knowledge.

1.5 Organization

- Data acquisition: This involves gathering data from various sources and preparing it for processing. This can include scraping web pages, querying databases, parsing documents, etc.
- Data pre-processing: Once the data is acquired, it needs to be cleaned and transformed into a format suitable for building a knowledge graph. This can involve removing duplicates, normalizing data, extracting relevant information, etc.
- Knowledge extraction: This involves using natural language processing (NLP) techniques to extract knowledge from the pre-processed data. This can include identifying entities, relationships, and events from text.
- Knowledge representation: The extracted knowledge needs to be represented in a format that is suitable for building a knowledge graph. This can include creating nodes for entities and relationships, defining properties, etc.
- Graph construction: Once the knowledge is represented, it needs to be added to the knowledge graph. This involves creating nodes, edges, and properties, and defining their relationships.
- Graph analysis: The knowledge graph can be analyzed using various techniques such as graph algorithms, clustering, and visualization.
- Performance analysis measures: Finally, we find out the accuracy measures, errors in our knowledge graphs and compared the results.

Chapter-2
LITERATURE SURVEY

2.1 Introduction

With the use of AI in everyday products companies are trying to build technology that improves the accuracy of existing models. A common example can be taken of the recommendation system that are present in almost every product we use. E-commerce websites, movie recommendation system, search engines also use recommendation to improve the search results. Also, the voice assistants we use such as Siri or Alexa are also based on AI and knowledge graphs adds an edge to the technology used. We have come across different research papers, books, journals and websites to understand better about knowledge graphs.

2.2 Related Work

[1] Yan, J., Wang, C., Cheng, W., Gao, M. and Zhou, A., 2018.

Autonomous knowledge collection and interpretation are difficult tasks for both robots and even humans because of the fragmented nature of information on the Internet and the multitude of data sources it is provided in. One of the best ways to integrate knowledge has recently acquired favour in both corporate and academic circles: knowledge graphs. Using approaches for knowledge graph generation, which integrate the information into knowledge that is represented in a graph, it is possible to mine information from organised, semi-structured, or even unstructured data sources. Knowledge graphs may manage data in a way that is also simple to use, understand, and maintain.

[2] Ayinuer, N., Ruxianguli, A. and Yasen, A., 2022.

Now that we live in the big data era, unstructured data is like a massive sea of smoke. Unstructured data makes up a large component of each line of work, and its analysis requires a significant number of human resources. This article will cover how to visualise unstructured data using Python for large amounts of unstructured data. After processing the data, it will be imported into the Neo4j database using Cypher syntax, after which a knowledge graph of the relationships between objects, organisations, and characters will be generated and made visible.

[3] Ive, Julia, Yue Zhang and Zhiyang Teng 2022.

Machines can now evaluate and produce natural language data thanks to the field of natural language processing (NLP), which combines computer science, linguistics, cognitive science, and machine learning (ML). The multi-disciplinary character of NLP draws specialists from a variety of disciplines, most of whom have experience with linguistics and machine learning. Because the field is heavily practice-based, traditional NLP textbooks tend to be task-specific and to dive into detail into the linguistic quirks of ML approaches to NLP. Additionally, they frequently present techniques that mainly use deep learning or classic ML. This book gives an introduction to natural language processing from a machine learning perspective, presenting statistical and learning - based models, generative and discriminative models, supervised and unsupervised models, and other important approaches and algorithms used in the area.

[4] Achichi, Manel, Zohra Bellahsene, and Konstantin Todorov 2017

Legato is an autonomous data linking system that works with datasets that comprise blocks of resources that are both highly varied in terms of descriptions and yet quite similar in their content. The Legato findings from the 2017 Ontology Alignment Evaluation Initiative's Instance Matching track are presented in this study using the SEALS platform. Legato took part in both of the instance matching track's sub-tracks. We give a brief overview of the Legato architecture, explain the many methods the system utilized to complete the data linking job, and exhibit and discuss how the system aligned with the other tools taking part in the 2017 assessment campaign.

[5] Hogan A, Blomqvist E, Cochez M, d'Amato C, Melo GD, Gutierrez C, Kirrane S, Gayo JE, Navigli R, Neumaier S, Ngomo AC 2021.

Source selection, which involves deciding which endpoints are pertinent to consider while evaluating a particular query, is a crucial step in federated query execution systems. Before the federated query, the strategic sourcing process is carried out separately by running SPARQL ASK queries, upgrading catalogues and indexes, or gathering heuristic data as a pre-processing stage. However, in some domains, such as the sense of the Linked Open University, these strategies have drawbacks. The DCAT metadata language, on the other hand, enables a publisher to describe datasets and data services in a catalogue using a common model and vocabulary that makes it easier for users to consume them. Additionally, data summarizations are indeed a compact way to convey essential dataset information. Additionally, the client-server interactions may be automated with the use of RDF Web APIs thanks to the Hydra hypermedia vocabulary and Hydra API Documentation. In order to make the source selection process easier, this study focuses on leveraging the former semantic vocabularies in conjunction with a context-based united well-accepted language. A case study in the framework of the Linked Open University was provided to clarify our concept. The case study demonstrated that our proposal enables the selection of the appropriate sources for each triple pattern without adding extra processing complexity through the use of SPARQL ASK queries. As a result, it is tailored to both new query interfaces and established query interfaces such as SPARQL endpoints.

[6] Ji, Shaoxiong, Shirui Pan, Erik Cambria, Pekka Marttinen, and S. Yu Philip 2021

Human knowledge offers a formal comprehension of the universe. Knowledge graphs, which describe the structural links among things, are being utilized in an increasing number of research that emphasize on cognitive and human-level intelligence. In the survey, we provide a comprehensive examination of the knowledge graph that encompasses the following key areas of study: Knowledge-aware applications, knowledge representation learning, knowledge acquisition and completion, a temporal knowledge graph, and so on. We also summarise recent developments and future research directions to help future studies. On these subjects, we suggest new taxonomies and a full-view classification. The four components of knowledge graph embedding are representation space, scoring function, encoding models, and supplementary data. Reviewing embedding techniques, route inference, and logical rule reasoning for acquiring knowledge, specifically knowledge graph completeness. We continue to investigate a number of cutting-edge subjects, such as metarelational learning, common sense, and temporal knowledge graphs. We also offer a selected selection of sets of data and open-source libraries on various tasks to help future studies on knowledge graphs. We now have a comprehensive outlook on a number of intriguing research topics.

[7] Noy, Natasha, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor 2019.

The knowledge graphs of five different tech firms are examined in this article, along with the connections and variations in how each company built and used the graphs and the difficulties that all information-driven businesses currently face. The variety of uses for knowledge graphs presented here ranges from search to product details to social networks.

[8] Liu, Haibo, Guoyi Jiang, Linhua Su, Yang Cao, Fengxin Diao, and Lipeng Mi 2020.

China's foreign power projects are expanding as part of "The Belt and Road," which is related to a lot of project information being disseminated. The administration of international power projects may be facilitated by the creation of power project knowledge graphs based on graph databases like Neo4j, as well as by having an intuitive grasp of the links between projects for future overall planning. According to the knowledge graph of international power projects, businesses may characterise the geographical distribution features of the Belt and

Road countries to better grasp the potential for future power investment development in other nations.

[9] Yang, Xu, Ziyi Huan, Yisong Zhai, and Ting Lin 2021.

Due to its strong recommendation impact, knowledge graph-based personalised recommendations have recently attracted the attention of academics. In this essay, we investigate knowledge graphs-based tailored recommendations. Prior to studying the knowledge graph development process, we finish building the movie knowledge graphs. We also utilise Neo4j graph database to vividly display and store the movie data. Then, utilising the knowledge of the nearby feature structures of the entities in the knowledge graph, we enhanced the algorithm using a cross-training approach to study the classical translation model TransE algorithm in knowledge graph representation learning technology. Additionally, the TransE algorithm's negative sampling procedure has been enhanced. The experimental findings demonstrate that the enhanced TransE model can vectorize entities and relations more precisely. This study culminates with the construction of a recommendation model using knowledge graphs, ranking learning, and neural networks. We provide two knowledge graph-based recommendation models: a neural network recommendation model and a Bayesian personalised recommendation model (KG-BPR) (KG-NN). We compare the outcomes of embedding the semantic information of entities and relations in knowledge graphs into vector space using the revised TransE approach. The BPR model and neural network, respectively, use item entity vectors holding external knowledge information to make up for the item's own lack of knowledge information. On the MovieLens-1M data set, the experimental analysis is completed. The experimental findings demonstrate that the two recommendation models put forward in this study may significantly increase recommendation accuracy, recall, F1 value, and MAP value.

[10] Dessì, Danilo, Francesco Osborne, Diego Reforgiato Recupero, Davide Buscaldi, and Enrico Motta 2021.

New problems are created at the same time as innovations are brought about by the ongoing expansion of scientific knowledge. One of these has to do with the fact that managing and annotating the large number of published articles has made it difficult to analyse, necessitating manual labour. To save time for academics, research policy makers, and businesses while browsing, analysing, and forecasting scientific research, new technology infrastructures are required. Large networks of items and relationships, or knowledge graphs, have shown to be a successful solution in this area. Scientific knowledge graphs concentrate on the academic realm and often include metadata about the authors, locations, organisations, study subjects, and citations of research articles. The knowledge offered in the study publications is not explicitly represented in the knowledge graphs of the present generation, nevertheless. As a result, we offer a novel architecture in this work that makes use of NLP and ML techniques to extract entities and connections from research papers and combine them into a massive knowledge graph. Using a variety of cutting-edge Natural Language Processing and Text Mining tools, we I address the challenge of knowledge extraction; (ii) describe a method for integrating the entities and relationships produced by these tools; and (iii) demonstrate the benefit of such a hybrid system over other approaches and (vi) as a chosen use case, we created a scientific knowledge graph with 109,105 triples, which we pulled from 26,827 publication abstracts on the Semantic Web. We anticipate that because our method is universal and adaptable to any area, it will make it easier to manage, analyse, disseminate, and process scientific knowledge.

[11] Dou, Jinhua, Jingyan Qin, Zanxia Jin, and Zhuang Li 2018.

Intangible cultural heritage (ICH) is a country's precious historical and cultural treasure. National culture must be handed down and protected if it is to flourish in a sustainable manner. Many different pieces of intangible cultural heritage may be found in China. As information technology advanced, government agencies and organisations providing public cultural services created ICH database resources, although the majority of these databases were scattered worldwide. Massive data management, storage, and analysis are hampered by some traditional database systems. A significant amount of data has also been generated at the same time as the growth of digital intangible cultural assets. The large and fragmented structure of the data makes it difficult for the general population to immediately understand essential information. To help knowledge management and the general public, we presented the intangible cultural heritage knowledge graph as a solution to these issues. Experts in intangible cultural heritage and knowledge engineers worked together to establish the ICH domain ontology, which governs the idea, attribute, and connection of ICH information. Natural Language Processing (NLP) technology was used in this work to extract domain knowledge from large amounts of ICH text data. For Chinese intangible cultural heritage, a knowledge base based on domain ontology and examples was built, and a knowledge graph was created. Based on the ICH knowledge network, the patterns and traits underlying intangible cultural heritage were illustrated. The organisation, administration, and conservation of intangible cultural heritage knowledge may be supported by the knowledge graph for ICH. Additionally, the general public may easily find the connected knowledge and acquire the ICH expertise. The preservation and transmission of intangible cultural resources are aided by the knowledge graph.

[12] Miller, Justin J 2013.

Relational database systems (RDBMS) can now be replaced with graph databases (GDB) (RDBMS). Applications that may be expressed in a much more natural manner include those in chemistry, biology, semantic web, social networking, and recommendation engines. Relational database systems (Oracle, MySQL) and graph databases (Neo4J) will be compared, with an emphasis on elements such data structures, data model features, and query capabilities. The inherent and modern limits of some of the existing comparisons and contrasts between implementations of graph and relational databases will also be discussed.

[13] Guu, Kelvin, John Miller, and Percy Liang 2015.

Compositional questions like "What languages are spoken by persons living in Lisbon?" may be resolved using path queries on a knowledge network. However, missing facts (edges) in knowledge graphs frequently prevent route searches from working. By embedding knowledge graphs in vector spaces, recent models for knowledge base completeness infer missing information. We demonstrate that although these models may be used recursively to respond to path questions, they have cascading flaws. This drives the development of a new "compositional" training objective, which significantly enhances the capacity of all models to respond to route inquiries, sometimes more than doubling accuracy. We further show that compositional training functions as a unique type of structural regularisation on a common knowledge base completion task, consistently enhancing performance across all base models (decreasing mistakes by up to 43%) and obtaining new state-of-the-art outcomes.

[14] Fensel, Dieter, U. Simsek, Kevin Angele, Elwin Huaman, Elias Kärle, Oleksandra Panasiuk, Ioan Toma, Jürgen Umbrich, and Alexander Wahler 2020.

The building of knowledge graphs, including manual, semi-automatic, and automatic approaches as well as their implementation, validation, and integration into knowledge graphs, are all covered in this book along with the tools and methods that enable information providers to do so. Additionally, it offers lifecycle-based methods for the semi-automated and automatic curation of these graphs, including techniques for knowledge graph evaluation, mistake correction, and enrichment with additional static and moving resources. Knowledge graphs are defined in Chapter 1, with less emphasis on mathematical accuracy and more on the effects of different techniques. Chapter 2 describes the creation, application, upkeep, and deployment of knowledge graphs. As a resource for open and service-oriented dialogue systems, Chapter 3 then shows pertinent application layers that can be constructed on top of such knowledge graphs and demonstrates how inference may be used to build views on such graphs. Applications of knowledge graph technology for e-tourism and use cases for other verticals are covered in Chapter 4. Finally, Chapter 5 offers a summary and outlines possible future paths. In the supplementary appendix, a domain specification's abstract syntax and semantics are introduced. This allows schema.org to be customized for different domains and jobs. The book analyses a number of pilot projects with an emphasis on conversational interfaces, showing how to utilize knowledge graphs for e-marketing and e-commerce in order to demonstrate the practical application of the methods offered. It is designed for advanced academics and professionals who need a quick overview of knowledge graphs and how to use them.

2.3 Summary of Literature Review

We have discussed models by which data can be structured as graphs, representations of schema, identity, and context, techniques for utilizing deductive and inductive knowledge, and methodologies for the formation, enrichment, quality assessment, and improvement of knowledge. A knowledge graph is defined as a graph of data intended to acquire and express knowledge of the real world. Knowledge graphs have drawn a lot of interest from many organizations and companies as well as various research communities. This interest is attributable, in large part, to the universality of the issue that knowledge graphs seek to solve: the integration and value extraction from many data sources at scale, whether in the context of a specific organization, community, or more general collections of human knowledge. The main finding of knowledge graphs is that, for describing and integrating many types of large-scale data, graphs offer a straightforward, flexible, intuitive, and still effective abstraction. This understanding is not brand-new, but rather has matured with the introduction of knowledge graphs. Graphs have historically been utilized to represent data and information in a variety of domains, including Graph Algorithms and Theory, Graph Databases, Information Extraction, Information Representation, Machine Learning, the Semantic Web, and others. Knowledge graphs may now take use of these developments since they have been merged. As a result, using a graph to represent data opens up a "tool-box" of languages, approaches, and systems that may be applied to combine and extract value from data in large scale.

There are currently a number of graph query languages that, in contrast to previous NoSQL options, are completely functional and enable both relational algebra and cutting-edge capabilities like navigational searches that can match pathways of any length. There are currently a variety of graph databases and user interfaces that support these query languages. While a thorough (relational-like) schema is not necessary for graphs to represent data, several conceptions of graph schemata have been developed in an effort to evaluate, condense, and define the semantics of graphs. Contextual frameworks for graphs, such as different reification alternatives, annotated graph frameworks, etc., may be used to express and reason about the extent of truth of knowledge included in the graph with regard to the time, space, provenance, confidence level, etc. Deductive forms of reasoning may be supported over graphs by using ontologies and/or rules. Through materialization or query rewriting, these methods can automatically provide access to the implicit information that the graph involves in addition to encoding a machine-readable consensus on its meaning. With graph parallel frameworks capable of applying such algorithms at a wide scale, graph algorithms, such as centrality measures, community identification, and clustering, may be deployed to the data to get insights into significant entities or edges, close-knit sub-graphs of entities, and more. It is now possible to apply machine learning natively across graphs in the context of a number of tasks, including classification, question answering, recommendations, and more thanks to recent and ongoing breakthroughs in knowledge graph embeddings and graph neural networks.

To retrieve formal, declarative assumptions that capture high-level trends and may be used to derive additional information in an understandable manner, a knowledge graph may be researched using rule and logic mining techniques. Graph-based information extraction may be used to extract and/or update a knowledge graph from legacy sources of text and semi-structured data, whilst

graph-based mapping languages make it simpler to incorporate multiple sources of legacy structured data into the knowledge graph.
Ontology engineering and learning tools, strategies, and approaches can help further direct the development of an ontology for the knowledge network, recording a consensus over its semantics and facilitating access to implicit knowledge via deductive reasoning.
When using one of the many tools and frameworks that are available to aid in performing such assessments, quality dimensions and metrics for knowledge graphs allow for systematically assessing the readiness of the knowledge graph for its anticipated applications, both qualitatively and quantitatively. As a result, choosing to describe data as a graph creates a "tool-box" of languages, methodologies, and systems that may be used to integrate and extract value from data at a wide scale. There are currently a number of graph query languages that, in contrast to previous NoSQL options, are completely functional and enable both relational algebra and cutting-edge capabilities like navigational searches that can match pathways of any length. There are currently a variety of graph databases and user interfaces that support these query languages.

Chapter-3
SYSTEM DESIGN & DEVELOPMENT

In the first stage, we will search for textual data and extract all of the pertinent information from that specific data. After that, we will upload the data to a local database we've created in our Neo4j desktop application. We will use Natural Language Processing to construct a knowledge graph for the preceding literature input (NLP). Natural Language Processing (NLP), an automated approach of text analysis, is based on a variety of concepts and tools. As it is such a dynamic and significant field of research and progress, there isn't a single description that can be accepted by everyone, but there are a number of components that would be included in any definition from a qualified candidate. Natural Language Processing (NLP) is a conceptually backed collection of computer technologies for evaluating and presenting naturally produced texts at one or more stages of linguistic analysis in order to provide human-like language processing for a range of tasks or applications.

This definition's many elements can be further examined. The phrase "spectrum of computational procedures" should be used sparingly because there are several ways to carry out various sorts of language analysis. For instance, "naturally occurring texts" might be written in any language, literary genre, or style. The messages might be spoken or written. The sole requirement is that they must be communicated in a language in which users converse.

Furthermore, the material being analysed should be taken from real usage rather than being created especially for the analysis. The phrase "levels of linguistic analysis" refers to the several forms of language processing that are known to be active when people produce or comprehend language. This notion will be further discussed in Section 2. It is thought that since each level provides a unique type of meaning, humans often employ all of these levels. However, as can be seen from the disparities between the various NLP applications, different NLP systems utilise different levels of language analysis, or combinations of levels. This also creates a lot of uncertainty among non-specialists as to what NLP truly is because any system that makes use of any subset of these levels of analysis may be described to as employing NLP. Therefore, whether the system uses "poor" or "powerful" NLP may really determine what distinguishes them.

It is clear from "Human-like language processing" that NLP is regarded as a field within Artificial Intelligence (AI). The fact that NLP aims for performance similar to that of a human being makes it legitimate to classify it as an AI discipline, although its whole lineage does depend on a number of other disciplines. NLP is not often considered as a goal in and of itself, with the possible exception of AI researchers, as shown by the statement "for a range of activities or applications." Others use NLP as a tool to complete a certain job. As a result, NLP is used by Information Retrieval (IR) systems as well as Machine Translation (MT), Question-Answering, and other techniques.

NLP aims to "achieve human-like language processing," as mentioned above. The word "processing" was chosen with great care; "understanding" should not be used in its stead. Because, despite the fact that the area of NLP was once known as Natural Language Understanding (NLU) in the early days of AI, it is generally acknowledged that, despite being the objective of NLP, real NLU has not yet been achieved. A complete NLU system could:

1. Paraphrase an input text
2. Translate the text into another language

Answer questions about the contents of the text 4. Draw inferences from the text

NLU is still the objective of NLP, even though that NLP has made some significant progress toward achieving goals 1 through 3. This is due to the fact that NLP systems cannot, by themselves, infer meaning from text. There are more specific objectives for NLP, many of which are connected to the specific application being used. For instance, the objective of an NLP-based IR system is to respond to a user's actual information needs by giving more accurate, detailed information. The NLP system's goal in this instance is to represent the user's query's actual meaning and purpose, which they may convey as naturally in common English as if they were chatting to a reference librarian. In order to locate a real match between need and answer, regardless of how either is stated in

their surface form, the contents of the documents that are being searched will also be represented at all of their levels of meaning. We impose the vectorization approach on the precomputed knowledge graphs in the next step of the model developing phase. To do this, the knowledge graph must be embedded in the vector space. Knowledge graph embeddings are a type of vector representation for the nodes and connections in a knowledge graph that preserves both the graph's natural structure and its ability to be reasoned about. To utilize graphs as an input to machine learning methods, vectorization or embeddings (numerical representation of elements and relations in a graph) are required. We require various methods to learn the numerical representations of knowledge graphs because we consider them differently from other types of graphs (or embeddings). Knowledge graph embeddings (KGE) may be created in a variety of methods, which we can broadly divide into three categories:
Translation based methods:
Here, embeddings are created using distance-based functions in Euclidean space. The head and related vectors may be combined to equal the tail vector using a simple technique. The equation can be written as h + r t. The TransE algorithm is used. There are further iterations of the algorithm, although they only make minor changes. TransH, TransR, TransD, TransSparse, and TransM are a few examples.

1. Factorization based methods:
Based on the concept of tensor factorization, the first algorithm employing this method to be developed was RESCAL. When n is the number of entities and m is the number of relations, a three-way tensor is defined as n x n x m. The tensor has a value of 1 if there is a relationship between the entities, and 0 otherwise. This tensor is factorised in order to determine the embeddings. For big graphs, this is typically computationally costly. Algorithms based on RESCAL's concept, such as DistMult, HolE, ComplEx, and QuatE, solve the issue.

2. Neural network-based methods:

It is hardly surprising that neural networks are used to find knowledge graph embeddings given its widespread use nowadays. An algorithm called Semantic Matching Energy describes an energy function that is used to give a triple a value using neural networks. An energy function is used in a neural tensor network; however, a bilinear tensor layer is used in place of the typical linear layer. Convolutional neural networks, such as ConvE, restructure the numerical representations of entities and relations into the form of a "picture," use convolution filters to extract the features, and then train the resulting embeddings. To compute the knowledge graph embeddings, we may also use models like HittER and KBGAN that are based on Transformer and are inspired by GAN.

We have several Python libraries, such as: to implement these algorithms.

- LibKGE
- LibKGE
- PyKEEN
- GraphVite
- AmpliGraph

Structure of KGE algorithm

To develop an algorithm to calculate the knowledge graph embeddings, there are several fundamental elements that are common. Below is a list of a few of these recommendations:

1. Negative generation:

This is a theory about creating corrupted or negative triples in a knowledge network. Triples that are not present in the original graph are referred to as negative triples. These can be produced at random or using other methods, such as Bernoulli negative sampling.

2. Scoring function:

It is a function which encapsulates the triple and produces a value or a result. We may say that a triple is valid if the score is high, and a triple is negative if the score is low. One of the crucial aspects that make up the KGE algorithm is the scoring function.

3. Loss function:

We utilize a loss function while training this method since it is treated as an optimization problem. The scores of the positive and negative triples are used to calculate the loss in this loss function. We also use an optimizer in order to reduce the loss as much as possible. Cross entropy loss, pairwise margin-based hinge loss, etc. are just a few examples of loss functions utilised in this context.

We'll be using a number of Data Science and Machine Learning techniques on the knowledge graph after we've created the graph embeddings for it. The last two stages of model design—vectorizing the knowledge graph, constructing graph embeddings, and implementing data science and machine learning techniques to the graph database under the areas of basic ML and the Graph Data Science Library.

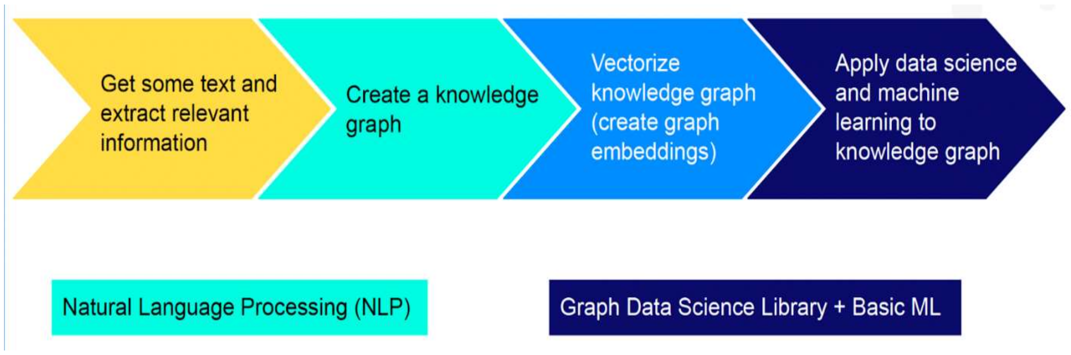

Get some text and extract relevant information
Create a knowledge graph
Vectorize knowledge graph (create graph embeddings)
Apply data science and machine learning to knowledge graph
Natural Language Processing (NLP)
Graph Data Science Library + Basic ML

Figure 3.1: Workflow of the model designing phase.

Tools and technologies used:

1. Google colab:

The two biggest trends in computer science right now are deep learning and machine learning. A lot of pupils are attempting to learn it and employ it in fantastic projects. Everyone is aware that no amount of reading, watching, or studying will be helpful if you don't put what you've learnt into practise on your own. However, you need very high-end hardware for your system to be able to manage such a demand. Additionally, not everyone has the money to purchase a laptop with these characteristics. So what are their alternatives for studying and implementing machine learning? Google Colab is the answer. To provide free access to GPUs and TPUs for anybody who needs it to construct a machine learning or deep learning model, Google established Google Colab. Google Colab is a more sophisticated variation of Jupyter Notebook. Using a web browser or an Integrated Development Environment, Jupyter Notebook is a programme that enables editing and execution of Notebook documents (IDE). You will deal with Notebooks rather than files.

Google colab features

The interesting features that each contemporary IDE offers are abundant in Google Colab, in addition to many others. Below is a list of some of the more fascinating aspects.

- Interactive lessons for learning neural networks and machine learning.
- Create and run Python 3 programmes without a local setup.
- Use the Notebook to run terminal commands.
- Import data from outside resources like Kaggle.
- Your notebooks should be saved to Google Drive.
- Google Drive notebook imports.
- No-cost cloud computing, GPUs, and TPUs.
- Integrate with Tensor Flow, PyTorch, and Open CV.
- Directly import or publish to/from GitHub.

2. Python 3:

At the National Research Institute for Mathematics and Computer Science in the Netherlands, Guido van Rossum created Python, a high-level, interpreted scripting language. Version 1.0 was released in 1994 after the original version was published in the alt. sources newsgroup in 1991.

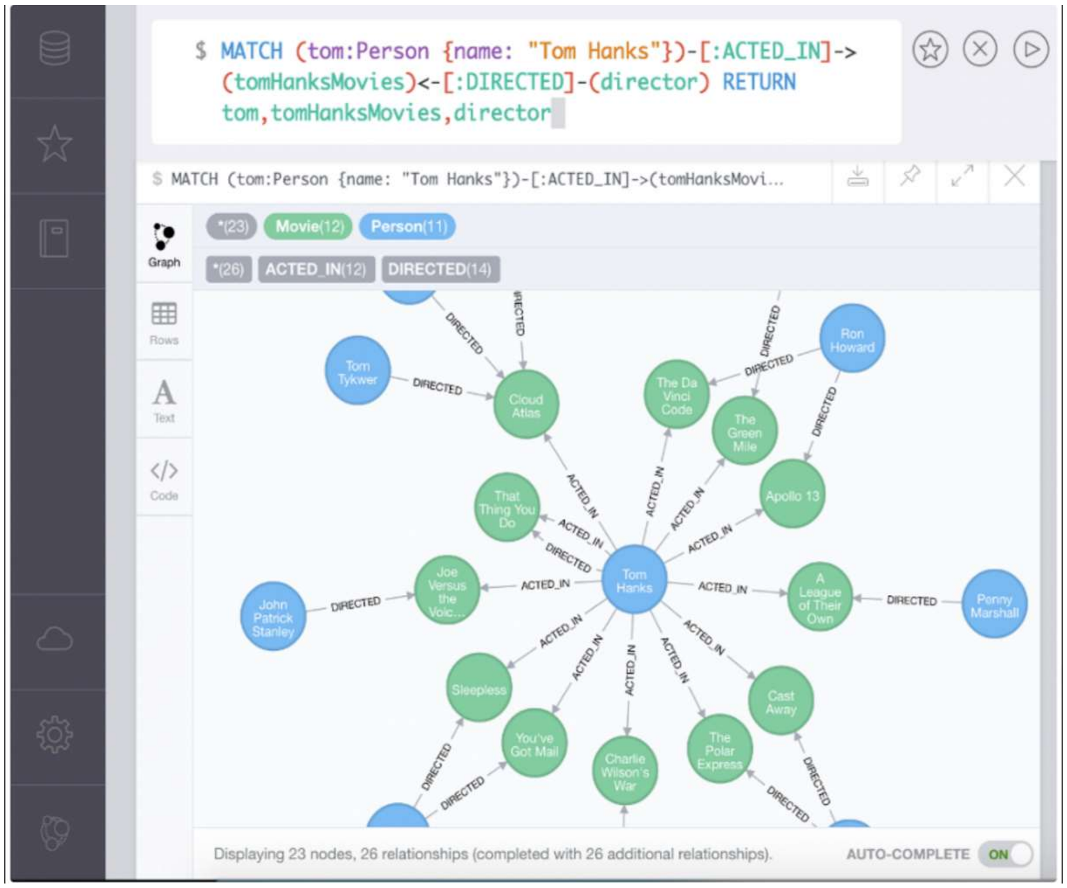
The 2.x series of versions predominated the market since the release of Python 2.0 in 2000 until December 2008. The development team then published version 3.0, which included a few very modest but significant changes that were not backward compatible with the 2.x versions. Python 2 and Python 3 are quite similar to one another, and several Python 3 features have been backported to Python 2. However, they are still incompatible in general. Python 2 and Python 3 have both been kept up to date on a regular basis with new releases for each. The most current versions as of this writing are 2.7.15 and 3.6.5. However, Python 2 has an official End of Life date of January 1, 2020, after which it will no longer be maintained. It is advised that you concentrate on Python 3 if you are new to the language, as this lesson will do. The Python community still refers to Guido as the BDFL (Benevolent Dictator for Life), and the language is still upheld by a core development team at the Institute.

By the way, the name Python really comes from the British comedy group Monty Python's Flying Circus, a favourite of Guido's and probably still is. It has nothing to do with snakes. The Python documentation frequently contains allusions to various Monty Python movies and skits.

3. Neo4j:

Neo4j is the most used graph database management system globally (DBMS). Additionally, it ranks one of the most popular Database management systems overall and NoSQL database systems.

Neo4j uses graphs to store and display data. Nodes and the connections between them serve as the representation of data. Relational databases like MS Access, SQL Server, MySQL, etc. are very different from Neo4j databases (as with any graph database). Tables, rows, and columns are used in relational databases to hold data. They also display data in a table-like format. Neo4j does not store or display data using tables, rows, or columns.



```
$ MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED_IN]->
  (tomHanksMovies)<-[:DIRECTED]-(director) RETURN
  tom,tomHanksMovies,director
```

```
$ MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED_IN]->(tomHanksMovi...
```

Displaying 23 nodes, 26 relationships (completed with 26 additional relationships).

Figure 3.2: Sample knowledge graph shown in Neo4j platform.

Neo4j is excellent for storing data with a large number of interconnected relationships. That's where graph databases can really shine. In actuality, graph databases, such as Neo4j, are far superior to relational databases in handling relational data.

This is partly because the graph model doesn't often call for a predetermined schema. Before loading the data, the database structure need not be created (like you do in a relational database). In Neo4j, the structure is the data. A "schema-optional" DBMS is Neo4j. However, the ability to build relationships is the primary factor making Neo4j preferable for relational data. Relationships are the foundation of Neo4j.

Setting up primary key/foreign key restrictions to specify which fields can relate to which data is not necessary. Simply add any relationship between any two nodes whenever you need to using Neo4j. For social networking apps like

Facebook, Twitter, etc., Neo4j is therefore ideally suited. However, Neo4j also excels in a wide range of other fields. Here are some of the primary applications for which Neo4j is useful:

- Social networks
- Realtime product recommendations
- Network diagrams
- Fraud detection
- Access management
- Graph based search of digital assets
- Master data management

Awesome Procedures on Cypher (APOC)

Awesome Procedures on Cypher is the acronym for this system. Developers had to create their own functions and procedures prior to the release of APOC in order to enable common functionality that Cypher or the Neo4j database did still not offer. There might be a lot of redundancy if each developer writes his or her own version of these routines.

As a result, one of our Neo4j engineers developed the APOC library as a common utility library for functions and procedures. This made it possible for developers working across platforms and sectors to use a standardised library for routine tasks and create their own functionality only when necessary for business logic or use-case-specific requirements. The APOC library is thought to be the most popular and substantial Neo4j extension library. More than 450 standard procedures are included, including capability for utilities, conversions, graph updates, and other things. They are well-supported and relatively simple to use in Cypher queries or to execute as standalone functions. Make careful to verify APOC to determine whether the adhoc function already exists before starting to develop it for your application.

Graph Data Science Library (GDS)

Common graph algorithms are easily implemented, parallelized, and provided as Cypher procedures via the Neo4j Graph Data Science (GDS) package. Additionally, GDS has machine learning pipelines that may be used to train predictive supervised models to address various graph-related issues, such as predicting relationships that are missing.

API tiers:

Cypher functions and procedures are included in the GDS API. Each of them falls into one of three maturity categories:

- Production-quality
    - Indicates that the feature has been tested with regards to stability and scalability.
    - Features in this tier are prefixed with gds.<operation>.
- Beta
    - Indicates that the feature is a candidate for the production-quality tier.
    - Features in this tier are prefixed with gds.beta.<operation>.
- Alpha
    - Indicates that the feature is experimental and might be changed or removed at any time.
    - Features in this tier are prefixed with gds. alpha. <operation>.

Algorithms

To calculate metrics for graphs, nodes, or links, graph algorithms are utilised.

They can offer information about pertinent graph entities (centralities, rankings), or innate structures like communities (community-detection, graph-partitioning, clustering). In many iterative graph algorithms, the graph is regularly traversed for computation utilising random walks, breadth- or depth-first searches, or pattern matching. Many of the techniques also have considerable computational complexity as a result of the exponential rise of alternative pathways with increasing distance. Fortunately, there are effective techniques that make use of certain network topologies, memorize previously examined areas, and parallelize processes. These optimizations have been used whenever practical. Numerous algorithms that are described in depth in the chapter on algorithms are part of the Neo4j Graph Data Science library.

2.1. Algorithm traits

GDS algorithms employ several characteristics of its input network in distinct ways (s). Our term for this is algorithmic characteristics. The implementation of an algorithm to provide well-defined outcomes in line with an algorithm trait is said to support the trait. These characteristics of algorithms exist:

1. Directed

The algorithm is well-defined on a directed graph.

2. Undirected

The algorithm is well-defined on an undirected graph.

3. Homogeneous

The algorithm will act uniformly and as if all nodes and connections in its input graph(s) were of the same kind. When examining the algorithm's output, it is important to take into consideration the fact that the network may contain different types of nodes or links.

4. Heterogeneous

The algorithm is capable of distinguishing between nodes and/or associations of various types.

5. Weighted

The algorithm allows for the weighting of node and/or relationship parameters to be adjusted. These values, which also are given by the nodeWeightProperty, nodeProperties, and relationshipWeightProperty configuration parameters, might economic growth rate, time, capacity, or other domain-specific attributes. By default, the algorithm will value each node and/or link equally.

6. Graph catalogue

GDS represents the graph information in a specific graph format in order to perform the algorithms as quickly as feasible. As a result, an in-memory graph catalogue must be loaded with the graph data from the Neo4j database. Graph reconstructions, which also provide features like filtering on node labels and connection types, can be used to manage the quantity of information loaded.

7. Cypher query language

Neo4j's graph query language is called Cypher Query Language. On the graph databases, it permits consumers to carry out a variety of CRUD activities (create, read, keep updating, and delete). Due to the fact that various graph databases have their own structured query language, it cannot be implemented with all of them. The most widely used graph query language is Gremlin, which is used by certain databases.

The flexibility of learning and mastering the Cypher query language is a plus. It is quite declarative and expressive. The syntax of Cypher makes it simple to identify links and node patterns in the graph. Cypher is a language for expressing visual patterns in graphs that was inspired by SQL. Without a comprehensive explanation of how to execute it, it enables users to specify what they wish to choose, insert, edit, or remove from our graph data. To handle the necessary create, read, update, and delete activities, Cypher may be utilized to develop expressive and effective queries.

8. spacy library

SpaCy is a Python Natural Linguistic Processing (NLP) framework that is open-source, free, and has a tonne of built-in features. It's getting more and more common for NLP data processing and analysis. Unstructured textual data is created on an enormous scale, thus it's crucial to analyse it and extract insights from it. You must portray the facts in a way that computers can comprehend in order to accomplish that. You can accomplish this using NLP.

9. NLP and SpaCy

NLP is a branch of artificial intelligence that analyses how computers and human languages interact. The process of obtaining meaning from human languages for computers through analysis, comprehension, and NLP. NLP offers a variety of implications, including the ability to extract insights from unstructured textNLP helps to extract insights from unstructured text and has several use cases, such as:

- Automatic synthesis
- acknowledgment of named entities
- mechanisms for resolving queries
- Sentimental evaluation

SpaCy is a Python NLP toolkit that is open-source and free. It is created to produce information extraction or natural language processing systems and is built in python. It offers a clear and approachable API and is intended for usage in production.

10. Pywikibot

A Python module and framework called Pywikibot is used to automate tasks on MediaWiki websites. It was originally developed for Wikipedia but is currently utilised on many other MediaWiki wikis as well as other Wikimedia Foundation projects. The project began in 2003, and the current version of the core code is 7.4.0. It has complete API use, is current with new MediaWiki capabilities, and has a Pythonic package structure. However, it also operates with versions of MediaWiki 1.23 or before. When used with a reference implementation of Python, Pywikibot is compatible with Microsoft Windows, macOS, and Linux. Any other operating system with a suitable installation of Python should also be able to use it. Python 3.5.3 or higher is presently required to execute the bot, although Python 3.6 or higher is suggested. To verify if you have Python installed and to determine its version, just type "python" at the CMD

11. Wikipedia python package

The most essential information source is the Internet. If we have an internet connection, we can access all knowledge with only one click. So it's important to understand how to get the precise information from the right sources. Data scraping is the practise of gathering information from numerous sources. Each of us has used Wikipedia. It is a place rich in knowledge. The biggest website on the internet, Wikipedia, has a tonne of information. It is an open-source platform that uses a wiki-based editing system to be maintained by a community of volunteer editors. It is an encyclopaedia that is bilingual. Python has a Wikipedia module (or API) that could be employed to extract data from Wikipedia pages. We can get and interpret information from Wikipedia thanks to this module. It functions as a little scrapper and can only scrape a certain amount of data, to put it simply. We must first install this module on our local PC before we can use it.

Model Development

Method 1: The NLP only approach

In this methodology, the model development procedure will be divided into many stages and shown via a workflow chart. The most crucial step in every data science endeavour will be taken as our initial action. the pre-processing or cleaning of the textual data, which will have a lot of redundant and useless information. A data scientist is reported to do roughly 80% of their work during the initial data cleansing phase. Clean data is essential since it will eventually impact how well the model we are trying to construct performs. Cleaning text of sounds is the initial stage in text analytics. You could be questioning what text noise is. Anything that won't be helpful in our study is considered noise, according to a minimum structure. I can split noise into four categories if I try:

Common Entities- Stop words (is, the, etc.), URLs, hashtags, punctuation, numbers, and other such things.

Slangs: Frequently used terms not found in dictionaries spelling and grammar mistakes Various Keywords

Text scrubbing not only aids in removing noise or redundant information but also decreases the number of dimensions in the data and simplifies the machine learning model. The noise will now be eliminated one by one. I'll demonstrate how you can rapidly clean up text by developing user-defined algorithms.

Maintain uniformity

1. Fixing encoding- You might have noted that as we are working with tweets, we have already imported and transferred electronically in the ISO-8859-1 format. Every language has a unique encoding, such as ASCII for English, BIG5 for Chinese, and Latin for West Europe. Organizing them into a typical and distinctive format is usually a smart idea. 'UTF8' is a frequently used format.
2. Change casing- It is generally beneficial to change the text to lower case to preserve uniformity.

Obtaining all named entities in the textual data is the following stage in this procedure (For instance it could be name, organization, animal, place, etc). Knowing precisely what a named entity is can help you better comprehend named entity recognition. Any type of text data may contain numerous names for actual objects, and such names may be regarded as named entities in the data. The named entity is denoted by the name of any person, place, or object in the data. Virat Kohli, India, the MacBook Pro, as well as other things that can have names are examples of named entities. A named entity, to put it more properly, is a depiction of the correct name of any object. Virat Kohli is the name of a cricket player, as was indicated in the example above. Finding and categorising significant chunks of data (entities) in text is a technique known as named entity recognition (NER), sometimes known as entity stacking, extraction, or identification. An entity is any term or set of words that frequently refers to the same thing. Every acknowledged object is assigned to a certain category. For instance, NER machine learning (ML) models may recognise the phrase "super.AI" in a text and classify it as a "Company." India is a nation, and MacBook Pro is a piece of technology (Thing). This article examines the fundamentals of NER, as well as some high-level use scenarios and how you may utilise it in your project or company.

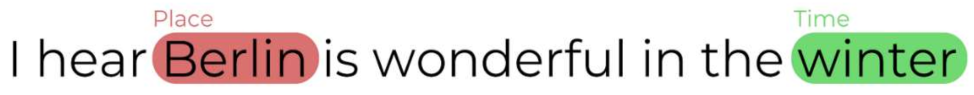


Figure 3.3: Example to explain NER.

Working of NER

At the heart of any NER model is a two-step process:

1. Detect a named entity
2. Categorize the entity

Beneath this lie a couple of things.

The first step is identifying a word or a set of words together that form an entity. Each phrase in the trio of tokens "The Great Lakes" comprises a separate entity and a token. Inside-outside-beginning tagging is frequently employed to indicate the start and end of an item. We'll go into
more detail about this in a later section. Entity categories must be created for the second phase. Here are some typical entity types:

1. Person
   - E.g., Elvis Presley, Audrey Hepburn, David Beckham
2. Organization
   - E.g., Google, Mastercard, University of Oxford
3. Time
   - E.g., 2006, 16:34, 2am
4. Location
   - E.g., Trafalgar Square, MoMA, Machu Picchu
5. Work of art
   - E.g., Hamlet, Guernica, Exile on Main St.

These are but a few instances. To fit your work, you may design your own entity categories and offer detailed criteria for which entities fall under which categories when there is uncertainty or task-specific ontologies. A model needs training data to understand what relevant entities are and are not, as well as how to classify them. The accuracy of the model in performing the task will increase with the relevance of the training data to the task. It could have problems surfing Twitter if you instruct your model on Victorian Gothic literature. You may use your entities and categories to classify data and construct a training dataset after they are built (our named entity recognition data programme can do this for you automatically). Then, using this training dataset, you train an algorithm to predictably categorise your content.

Use of NER

NER is suitable for any circumstance where a high-level summary of a sizable body of material is beneficial. With NER, you may swiftly group texts based on their relevance or similarity and understand the subject or theme of a body of material at a look.

Here are some notable NER use cases:

1. Human resources
   - Speed up the hiring process by condensing resumes of applicants; enhance internal operations by classifying employee complaints and queries
2. Customer support
   - Minimize response times by classifying user requests, grievances, and enquiries, then prioritising keywords.
3. Search and recommendation engines
   - By condensing explanation material, testimonies, and debates, search results and suggestions may be made timelier and more relevant.
   - Here, one significant success story is Booking.com.
4. Content classification
   - By recognising the topics of interest of blog posts and news items, one could surface material more quickly and obtain insights into trends.

5. Health care

- By eliminating critical information from lab results, you may raise patient care standards and decrease workloads.
- Roche uses radiological and pathology information to do this.

6. Academia

- Increase the rate with which students and researchers can locate pertinent information by summarising papers and archive material and emphasising key phrases, subjects, and themes.
- NER is used by European, the EU's digital portal for cultural heritage, to make old newspapers searchable.

Starting a NER project or business is simple if you think it could be beneficial. You may start with a number of strong open-source programmes, including as NLTK, SpaCy, and Stanford NER, each of which has benefits and drawbacks that we will discuss in more detail later.

But first, we need to develop a pertinent labelled dataset on which to train the model. Only then can you begin utilising one of these packages to build a model. Herein lies the value of super.AI. Using the named entity recognition data programme, you send us the raw text of your text together with the appropriate entities and categories.

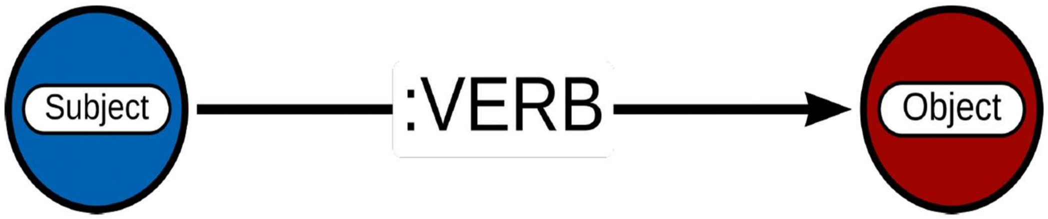


Figure 3.4: SVO triples in graph data.

Whenever you combine SVO triples, you might develop a multi where nodes stand in for entities (all subjects and objects) and directed edges represent relationships. The direction of the edge shows whether an entity occurs as a subject or an object (edge points from subject to object) (also called edge labels).

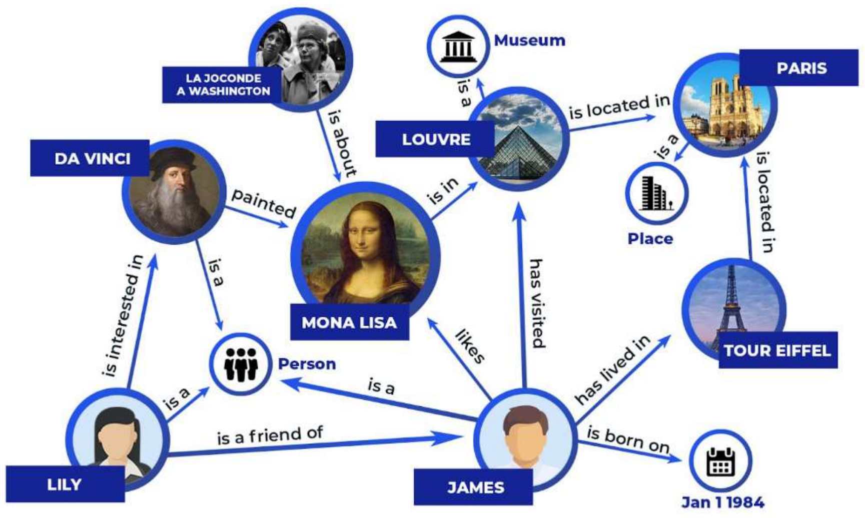


Figure 3.5: Example of a knowledge graph with SVO triples.

In the aforementioned illustration, it is evident that Da Vinci is a subject who created the Mona Lisa through painting (it is a verb-it makes a connection between the two specified things) (Object). The actual construction of the knowledge graph is the last stage in this approach.

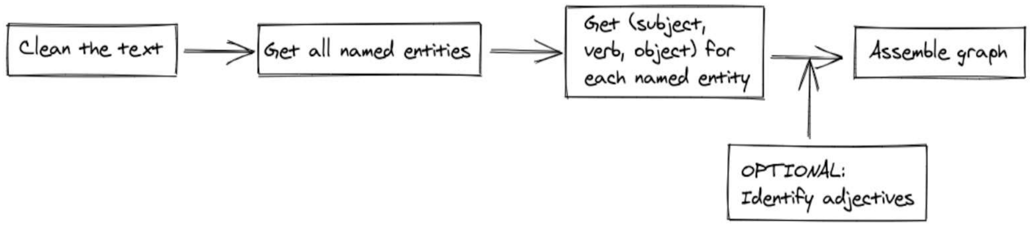

Clean the text
Get all named entities
Get (subject, verb, object) for each named entity
Assemble graph
OPTIONAL: Identify adjectives

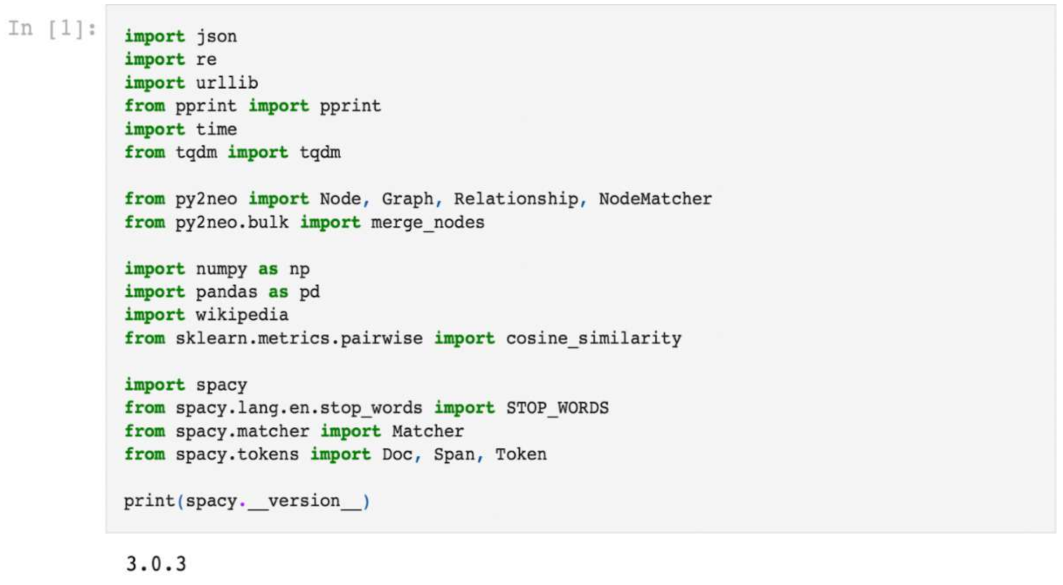
Figure 3.6: Workflow diagram of the model development phase.

Implementation

Importing all the necessary packages and libraries.

Configure spacy

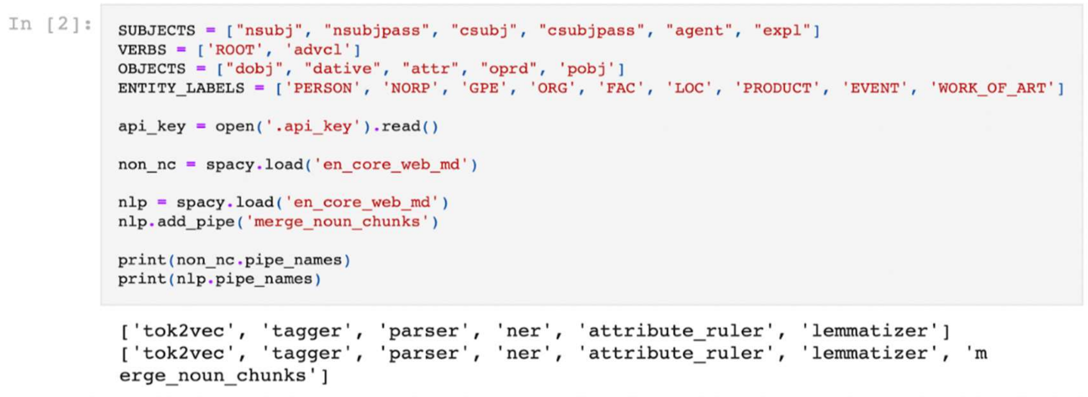
We have to load certain models before we can use spacy. Their simple core library, en_core_web, which has a 20 MB or less download size and offers strong, fundamental functionality, serves as the foundational model. However, this fundamental model has a flaw in that it lacks complete word vectors. It has context-sensitive tensors instead. You may still use it for tasks like text similarity, but if you want to utilise spacy to make accurate word vectors, pick a bigger model like en_core_web modern core web as the smaller models are not well known for accuracy. Although it is outside the purview of this workshop, you can also employ a number of third-party models. Usually, we use the following command to load the models. Download en_core_web md using python3 -m spacy. Although it has already been completed in the container, you may add additional models by either adding a cell to this notebook or using the CLI.



```
In [1]:  import json
         import re
         import urllib
         from pprint import pprint
         import time
         from tqdm import tqdm

         from py2neo import Node, Graph, Relationship, NodeMatcher
         from py2neo.bulk import merge_nodes

         import numpy as np
         import pandas as pd
         import wikipedia
         from sklearn.metrics.pairwise import cosine_similarity

         import spacy
         from spacy.lang.en.stop_words import STOP_WORDS
         from spacy.matcher import Matcher
         from spacy.tokens import Doc, Span, Token

         print(spacy.__version__)

         3.0.3
```

Figure 3.7: Importing all the required libraries.

Now that we have the data from Wikipedia, we will use natural language processing to analyse it. SVO triples are then added to the graph.

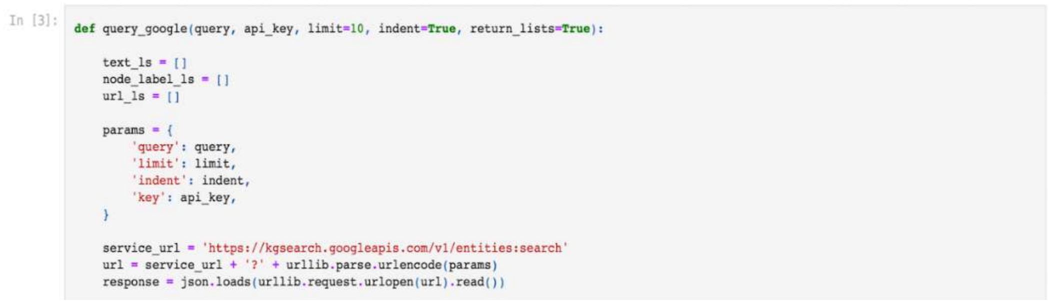


```
In [2]:  SUBJECTS = ["nsubj", "nsubjpass", "csubj", "csubjpass", "agent", "expl"]
         VERBS = ['ROOT', 'advcl']
         OBJECTS = ["dobj", "dative", "attr", "oprd", 'pobj']
         ENTITY_LABELS = ['PERSON', 'NORP', 'GPE', 'ORG', 'FAC', 'LOC', 'PRODUCT', 'EVENT', 'WORK_OF_ART']

         api_key = open('.api_key').read()

         non_nc = spacy.load('en_core_web_md')

         nlp = spacy.load('en_core_web_md')
         nlp.add_pipe('merge_noun_chunks')

         print(non_nc.pipe_names)
         print(nlp.pipe_names)

         ['tok2vec', 'tagger', 'parser', 'ner', 'attribute_ruler', 'lemmatizer']
         ['tok2vec', 'tagger', 'parser', 'ner', 'attribute_ruler', 'lemmatizer', 'm
         erge_noun_chunks']
```

Figure 3.8: Adding SVO triples and performing NLP on the graph.

Query google knowledge graph

You need an API key to access the Google Knowledge Graph, which permits you 100,000 free read requests each day for each project.

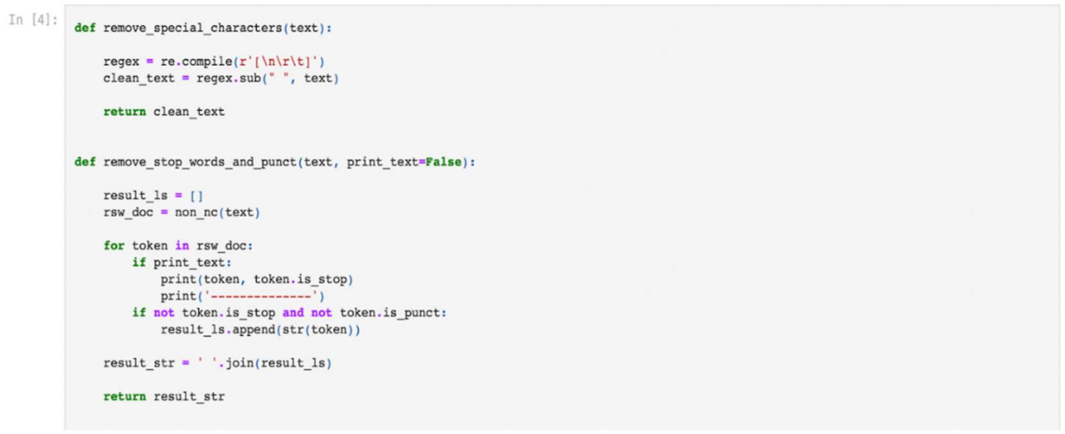


```
In [3]: def query_google(query, api_key, limit=10, indent=True, return_lists=True):

            text_ls = []
            node_label_ls = []
            url_ls = []

            params = {
                'query': query,
                'limit': limit,
                'indent': indent,
                'key': api_key,
            }

            service_url = 'https://kgsearch.googleapis.com/v1/entities:search'
            url = service_url + '?' + urllib.parse.urlencode(params)
            response = json.loads(urllib.request.urlopen(url).read())
```

Figure 3.9: Querying Google knowledge graph using an API key.

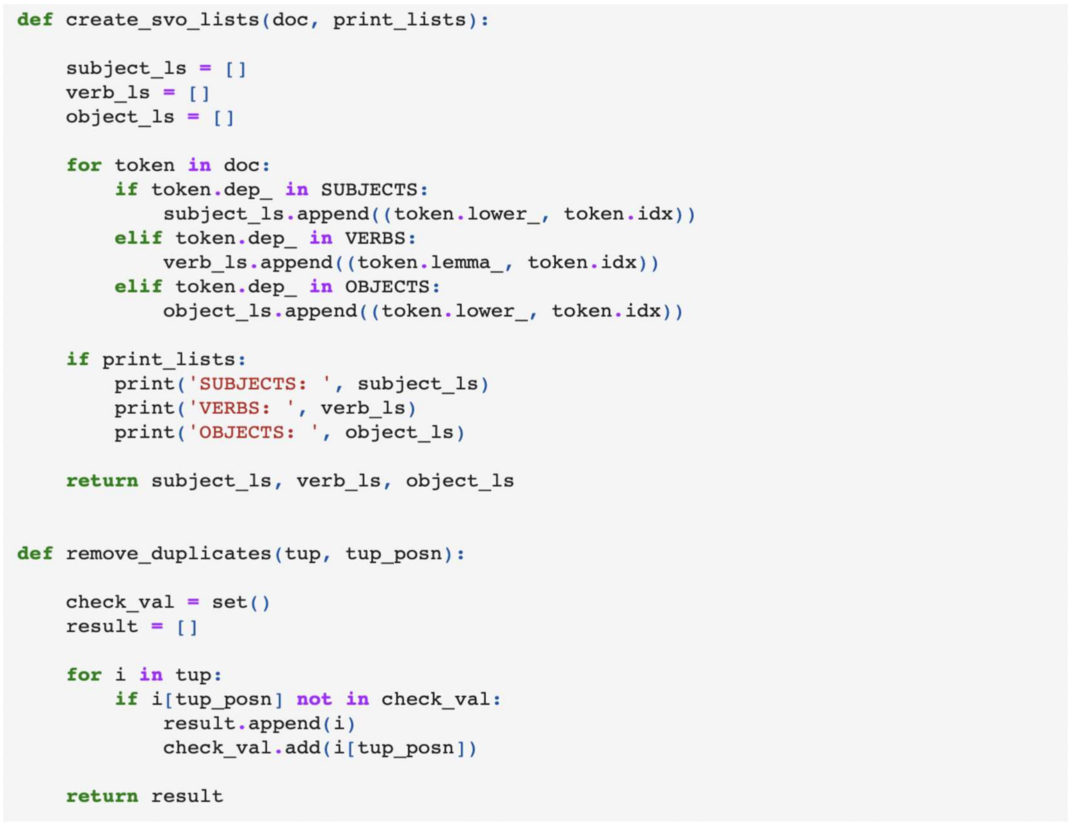
Helper functions for text cleaning

We need to clean up a lot, as with any data science effort. The following routines, such as remove special characters, remove special characters. Delete punctuation and commas (remove stop words and punctations). Dates are removed. Overlapping records (remove duplicates). Throughout this process, duplicates frequently appear, and we will spend a lot of time fighting them. Then, we can address create svo triples, which is the core issue.



```
In [4]: def remove_special_characters(text):

            regex = re.compile(r'[\n\r\t]')
            clean_text = regex.sub(" ", text)

            return clean_text


        def remove_stop_words_and_punct(text, print_text=False):

            result_ls = []
            rsw_doc = non_nc(text)

            for token in rsw_doc:
                if print_text:
                    print(token, token.is_stop)
                    print('--------------')
                if not token.is_stop and not token.is_punct:
                    result_ls.append(str(token))

            result_str = ' '.join(result_ls)

            return result_str
```

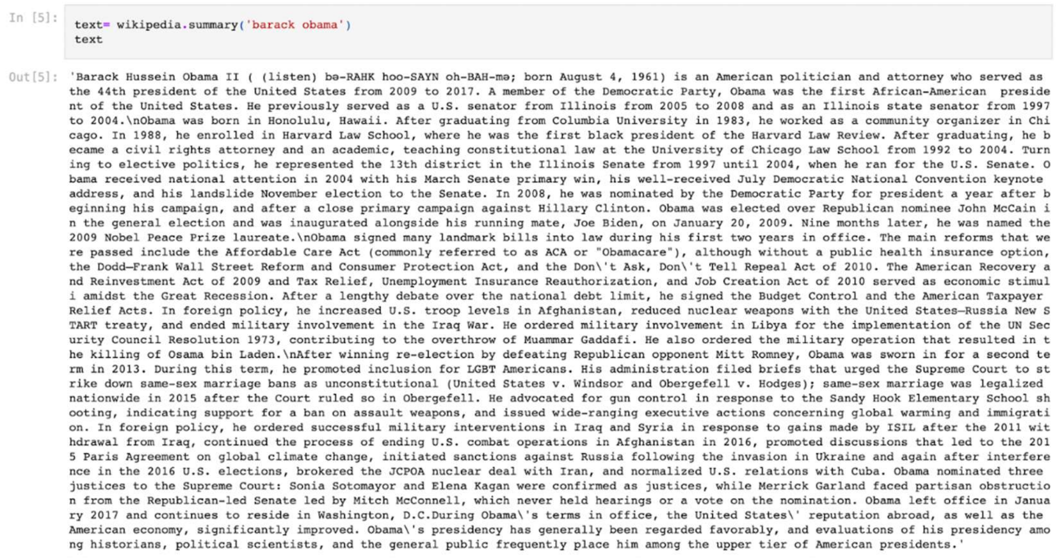
Figure 3.10: Helper functions to remove special characters and to stop words and punctuation.



```python
def create_svo_lists(doc, print_lists):

    subject_ls = []
    verb_ls = []
    object_ls = []

    for token in doc:
        if token.dep_ in SUBJECTS:
            subject_ls.append((token.lower_, token.idx))
        elif token.dep_ in VERBS:
            verb_ls.append((token.lemma_, token.idx))
        elif token.dep_ in OBJECTS:
            object_ls.append((token.lower_, token.idx))

    if print_lists:
        print('SUBJECTS: ', subject_ls)
        print('VERBS: ', verb_ls)
        print('OBJECTS: ', object_ls)

    return subject_ls, verb_ls, object_ls


def remove_duplicates(tup, tup_posn):

    check_val = set()
    result = []

    for i in tup:
        if i[tup_posn] not in check_val:
            result.append(i)
            check_val.add(i[tup_posn])

    return result
```

Figure 3.11: Helper functions to create SVO triples and to remove duplicates.

Here is the textual information on former American president Barack Obama that we extracted from Wikipedia.



```
In [5]:  text= wikipedia.summary('barack obama')
         text
```

```
Out[5]: 'Barack Hussein Obama II ( (listen) bə-RAHK hoo-SAYN oh-BAH-mə; born August 4, 1961) is an American politician and attorney who served as
the 44th president of the United States from 2009 to 2017. A member of the Democratic Party, Obama was the first African-American  preside
nt of the United States. He previously served as a U.S. senator from Illinois from 2005 to 2008 and as an Illinois state senator from 1997
to 2004.\nObama was born in Honolulu, Hawaii. After graduating from Columbia University in 1983, he worked as a community organizer in Chi
cago. In 1988, he enrolled in Harvard Law School, where he was the first black president of the Harvard Law Review. After graduating, he b
ecame a civil rights attorney and an academic, teaching constitutional law at the University of Chicago Law School from 1992 to 2004. Turn
ing to elective politics, he represented the 13th district in the Illinois Senate from 1997 until 2004, when he ran for the U.S. Senate. O
bama received national attention in 2004 with his March Senate primary win, his well-received July Democratic National Convention keynote
address, and his landslide November election to the Senate. In 2008, he was nominated by the Democratic Party for president a year after b
eginning his campaign, and after a close primary campaign against Hillary Clinton. Obama was elected over Republican nominee John McCain i
n the general election and was inaugurated alongside his running mate, Joe Biden, on January 20, 2009. Nine months later, he was named the
2009 Nobel Peace Prize laureate.\nObama signed many landmark bills into law during his first two years in office. The main reforms that we
re passed include the Affordable Care Act (commonly referred to as ACA or "Obamacare"), although without a public health insurance option,
the Dodd–Frank Wall Street Reform and Consumer Protection Act, and the Don\'t Ask, Don\'t Tell Repeal Act of 2010. The American Recovery a
nd Reinvestment Act of 2009 and Tax Relief, Unemployment Insurance Reauthorization, and Job Creation Act of 2010 served as economic stimul
i amidst the Great Recession. After a lengthy debate over the national debt limit, he signed the Budget Control and the American Taxpayer
Relief Acts. In foreign policy, he increased U.S. troop levels in Afghanistan, reduced nuclear weapons with the United States–Russia New S
TART treaty, and ended military involvement in the Iraq War. He ordered military involvement in Libya for the implementation of the UN Sec
urity Council Resolution 1973, contributing to the overthrow of Muammar Gaddafi. He also ordered the military operation that resulted in t
he killing of Osama bin Laden.\nAfter winning re-election by defeating Republican opponent Mitt Romney, Obama was sworn in for a second te
rm in 2013. During this term, he promoted inclusion for LGBT Americans. His administration filed briefs that urged the Supreme Court to st
rike down same-sex marriage bans as unconstitutional (United States v. Windsor and Obergefell v. Hodges); same-sex marriage was legalized
nationwide in 2015 after the Court ruled so in Obergefell. He advocated for gun control in response to the Sandy Hook Elementary School sh
ooting, indicating support for a ban on assault weapons, and issued wide-ranging executive actions concerning global warming and immigrati
on. In foreign policy, he ordered successful military interventions in Iraq and Syria in response to gains made by ISIL after the 2011 wit
hdrawal from Iraq, continued the process of ending U.S. combat operations in Afghanistan in 2016, promoted discussions that led to the 201
5 Paris Agreement on global climate change, initiated sanctions against Russia following the invasion in Ukraine and again after interfere
nce in the 2016 U.S. elections, brokered the JCPOA nuclear deal with Iran, and normalized U.S. relations with Cuba. Obama nominated three
justices to the Supreme Court: Sonia Sotomayor and Elena Kagan were confirmed as justices, while Merrick Garland faced partisan obstructio
n from the Republican-led Senate led by Mitch McConnell, which never held hearings or a vote on the nomination. Obama left office in Janua
ry 2017 and continues to reside in Washington, D.C.During Obama\'s terms in office, the United States\' reputation abroad, as well as the
American economy, significantly improved. Obama\'s presidency has generally been regarded favorably, and evaluations of his presidency amo
ng historians, political scientists, and the general public frequently place him among the upper tier of American presidents.'
```

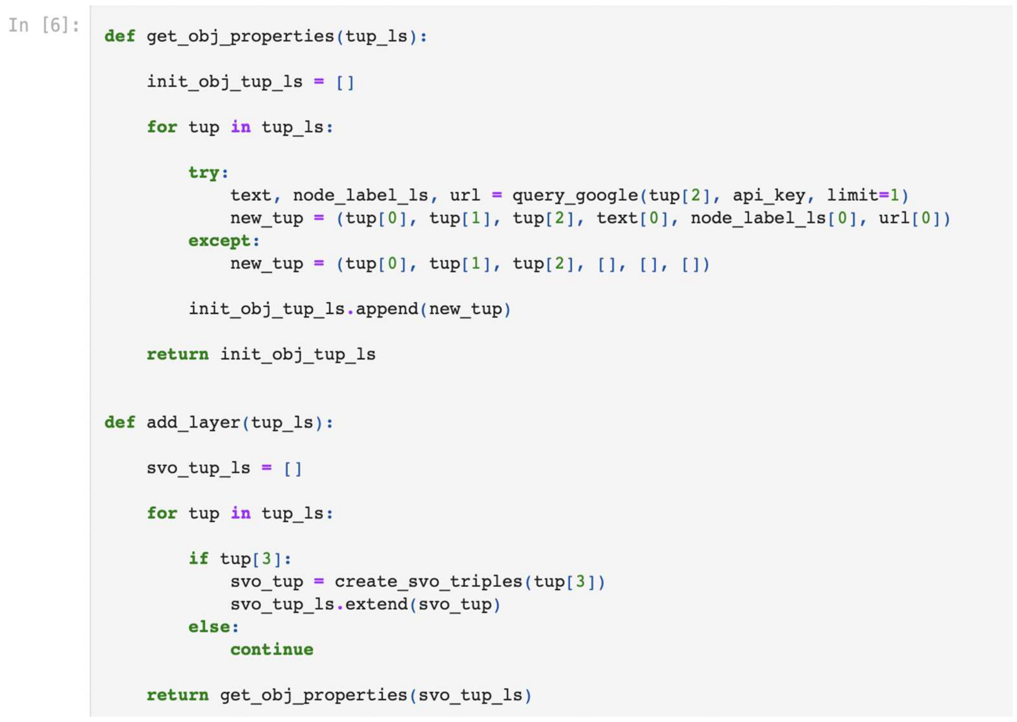
Figure 3.12: Wikipedia data of Barack Obama.

More helper functions

We are now prepared to begin retrieving data about each object in the list using the get obj properties function, such as:

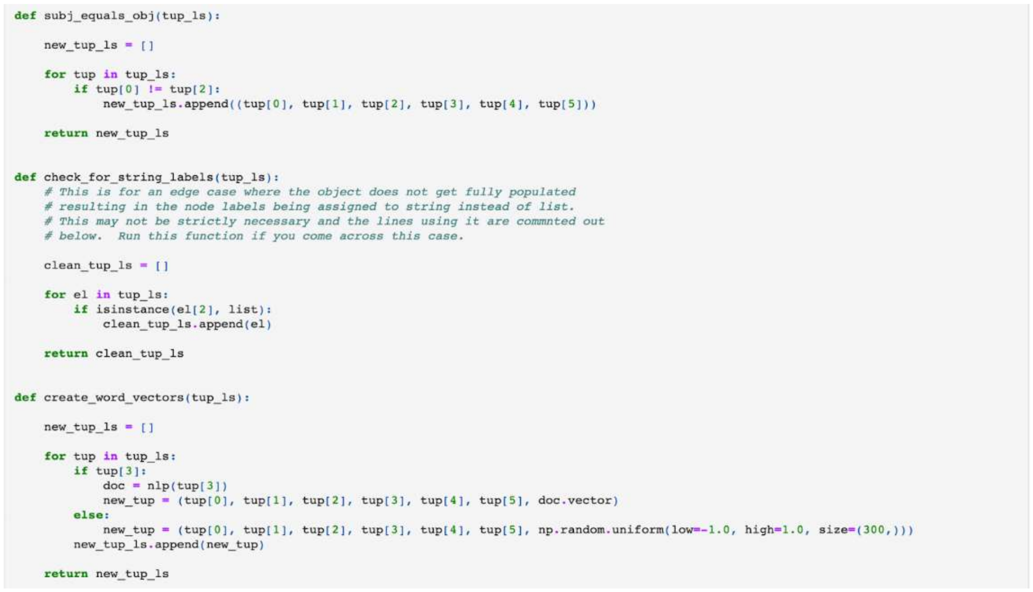
- any identified node labels
- any descriptions
- any URLs

All of the objects' node attributes will be created using these. We enable the ML models in spacy to generate word vector embeddings for each node based on the node description as we round up this part. (If no explanation is given, an array of zeros will be used.)



```
In [6]:
def get_obj_properties(tup_ls):

    init_obj_tup_ls = []

    for tup in tup_ls:

        try:
            text, node_label_ls, url = query_google(tup[2], api_key, limit=1)
            new_tup = (tup[0], tup[1], tup[2], text[0], node_label_ls[0], url[0])
        except:
            new_tup = (tup[0], tup[1], tup[2], [], [], [])

        init_obj_tup_ls.append(new_tup)

    return init_obj_tup_ls


def add_layer(tup_ls):

    svo_tup_ls = []

    for tup in tup_ls:

        if tup[3]:
            svo_tup = create_svo_triples(tup[3])
            svo_tup_ls.extend(svo_tup)
        else:
            continue

    return get_obj_properties(svo_tup_ls)
```

Figure 3.13: More helper functions used in the code.

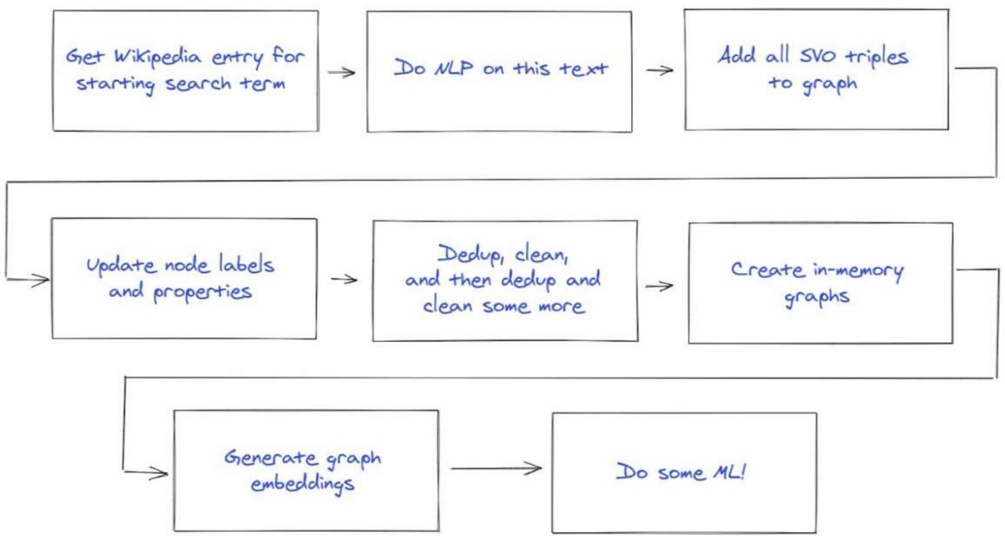


```python
def subj_equals_obj(tup_ls):

    new_tup_ls = []

    for tup in tup_ls:
        if tup[0] != tup[2]:
            new_tup_ls.append((tup[0], tup[1], tup[2], tup[3], tup[4], tup[5]))

    return new_tup_ls


def check_for_string_labels(tup_ls):
    # This is for an edge case where the object does not get fully populated
    # resulting in the node labels being assigned to string instead of list.
    # This may not be strictly necessary and the lines using it are commnted out
    # below.  Run this function if you come across this case.

    clean_tup_ls = []

    for el in tup_ls:
        if isinstance(el[2], list):
            clean_tup_ls.append(el)

    return clean_tup_ls


def create_word_vectors(tup_ls):

    new_tup_ls = []

    for tup in tup_ls:
        if tup[3]:
            doc = nlp(tup[3])
            new_tup = (tup[0], tup[1], tup[2], tup[3], tup[4], tup[5], doc.vector)
        else:
            new_tup = (tup[0], tup[1], tup[2], tup[3], tup[4], tup[5], np.random.uniform(low=-1.0, high=1.0, size=(300,)))
        new_tup_ls.append(new_tup)

    return new_tup_ls
```

Figure 3.14: Three more helper functions are used in this section.

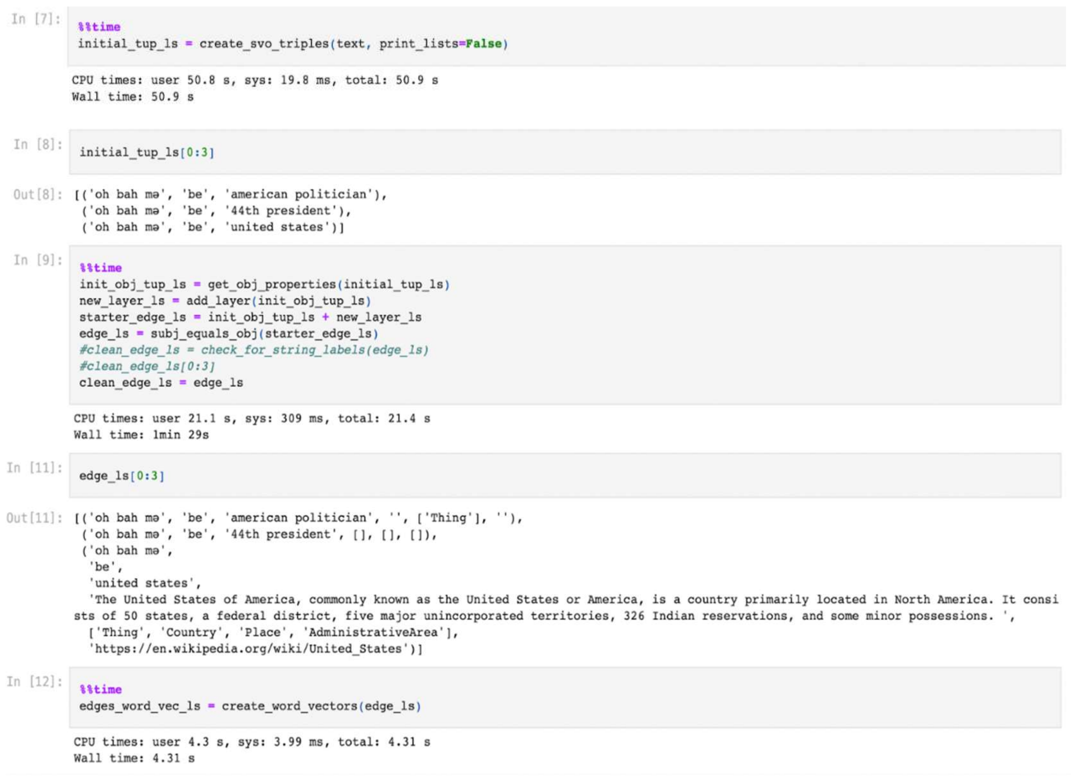
It's time to execute it step-by-step now. The first step in this method's process will be to obtain Wikipedia information for a search keyword (in our case it will be about Barack Obama). The next step is to apply Natural Language Processing (NLP) to the newly gathered textual information. Then, we are expected to create SVO triples and include every one of them on the graph.

That will be followed by the updating of the node labels and their characteristics. Then we deduplicate, clean, and repeat that operation numerous times since having clean data to deal with is really advantageous. Then, we will produce the graph embeddings and in-memory graphs. Finally, we will apply fundamental machine learning algorithms to our created knowledge graph.



Get Wikipedia entry for starting search term
Do NLP on this text
Add all SVO triples to graph
Update node labels and properties
Dedup, clean, and then dedup and clean some more
Create in-memory graphs
Generate graph embeddings
Do some ML!

Figure 3.15: Overview of the workflow in NLP only approach.



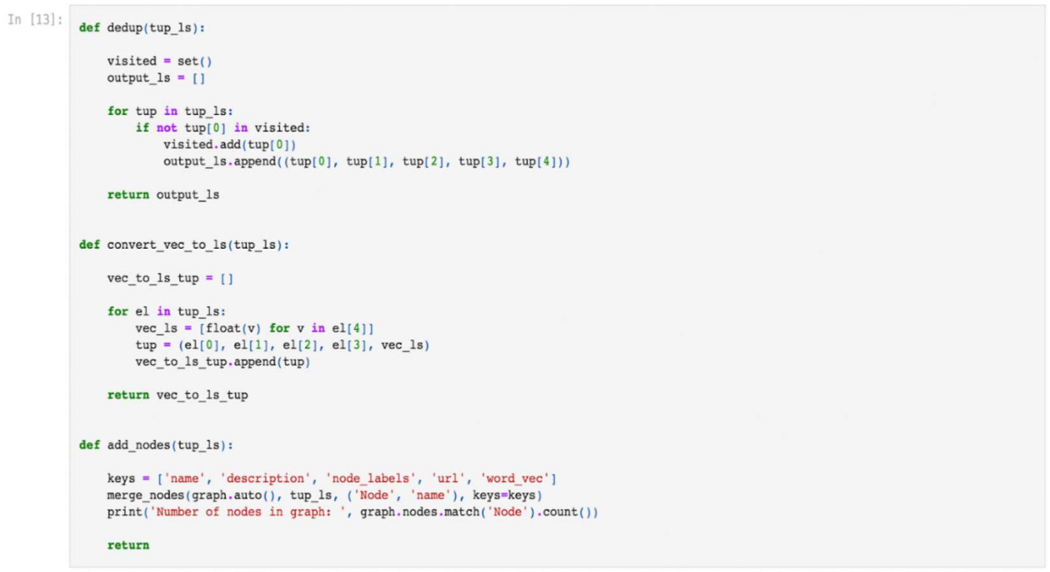
```
In [7]:  %%time
         initial_tup_ls = create_svo_triples(text, print_lists=False)

         CPU times: user 50.8 s, sys: 19.8 ms, total: 50.9 s
         Wall time: 50.9 s

In [8]:  initial_tup_ls[0:3]

Out[8]:  [('oh bah mə', 'be', 'american politician'),
          ('oh bah mə', 'be', '44th president'),
          ('oh bah mə', 'be', 'united states')]

In [9]:  %%time
         init_obj_tup_ls = get_obj_properties(initial_tup_ls)
         new_layer_ls = add_layer(init_obj_tup_ls)
         starter_edge_ls = init_obj_tup_ls + new_layer_ls
         edge_ls = subj_equals_obj(starter_edge_ls)
         #clean_edge_ls = check_for_string_labels(edge_ls)
         #clean_edge_ls[0:3]
         clean_edge_ls = edge_ls

         CPU times: user 21.1 s, sys: 309 ms, total: 21.4 s
         Wall time: 1min 29s

In [11]: edge_ls[0:3]

Out[11]: [('oh bah mə', 'be', 'american politician', '', ['Thing'], ''),
          ('oh bah mə', 'be', '44th president', [], [], []),
          ('oh bah mə',
           'be',
           'united states',
           'The United States of America, commonly known as the United States or America, is a country primarily located in North America. It consi
         sts of 50 states, a federal district, five major unincorporated territories, 326 Indian reservations, and some minor possessions. ',
           ['Thing', 'Country', 'Place', 'AdministrativeArea'],
           'https://en.wikipedia.org/wiki/United_States')]

In [12]: %%time
         edges_word_vec_ls = create_word_vectors(edge_ls)

         CPU times: user 4.3 s, sys: 3.99 ms, total: 4.31 s
         Wall time: 4.31 s
```

Figure 3.16: Fetching out phonetic spellings of target nodes.

Create the node and edge lists to populate the graph with the below helper functions

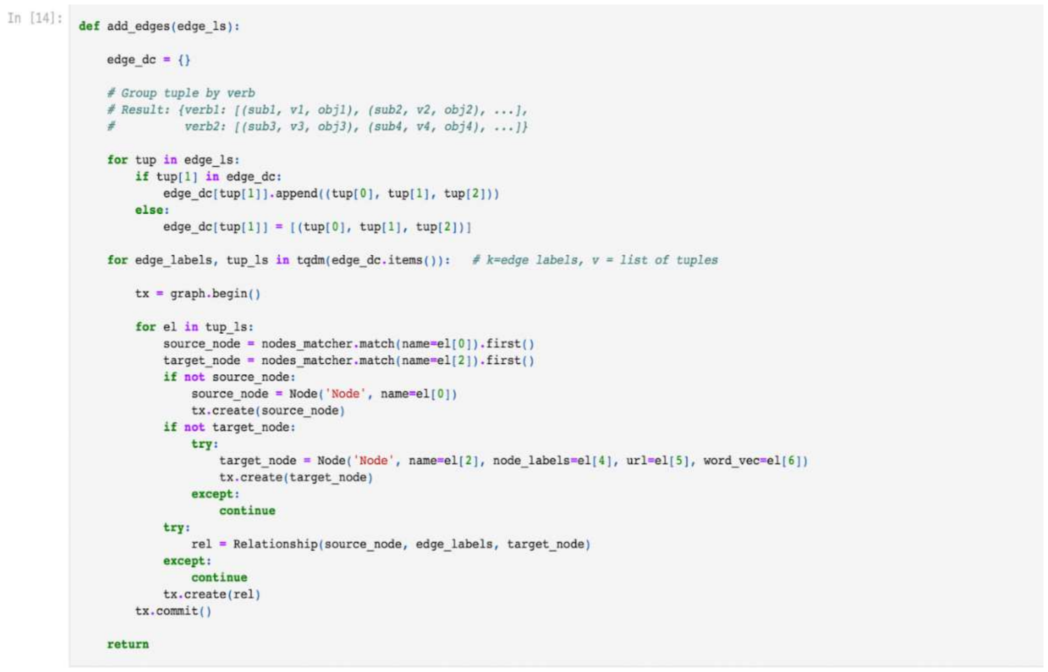
These functions achieve the following:

1. Deduping of the node list (dedup)
2. Under the idea that we might want to use word embeddings, we are pulling them in as a node property. However, Neo4j doesn't work well with numpy arrays, so we convert that array to a list of floats (convert_vec_to_ls).
3. Add nodes to the graph with the py2neo bulk importer (add_nodes)
4. Add edges (relationships) to the graph (add_edges)



```
In [13]:
def dedup(tup_ls):

    visited = set()
    output_ls = []

    for tup in tup_ls:
        if not tup[0] in visited:
            visited.add(tup[0])
            output_ls.append((tup[0], tup[1], tup[2], tup[3], tup[4]))

    return output_ls


def convert_vec_to_ls(tup_ls):

    vec_to_ls_tup = []

    for el in tup_ls:
        vec_ls = [float(v) for v in el[4]]
        tup = (el[0], el[1], el[2], el[3], vec_ls)
        vec_to_ls_tup.append(tup)

    return vec_to_ls_tup


def add_nodes(tup_ls):

    keys = ['name', 'description', 'node_labels', 'url', 'word_vec']
    merge_nodes(graph.auto(), tup_ls, ('Node', 'name'), keys=keys)
    print('Number of nodes in graph: ', graph.nodes.match('Node').count())

    return
```

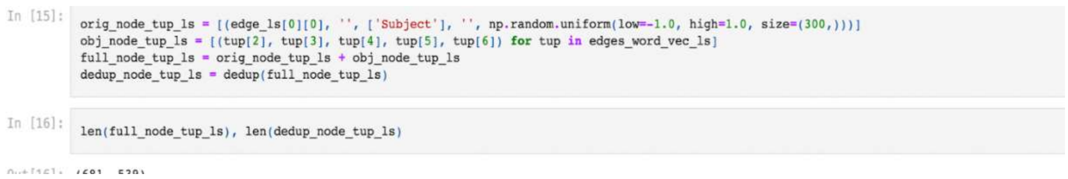
Figure 3.17: More helper functions used.



```
In [14]: def add_edges(edge_ls):

             edge_dc = {}

             # Group tuple by verb
             # Result: {verb1: [(sub1, v1, obj1), (sub2, v2, obj2), ...],
             #          verb2: [(sub3, v3, obj3), (sub4, v4, obj4), ...]}

             for tup in edge_ls:
                 if tup[1] in edge_dc:
                     edge_dc[tup[1]].append((tup[0], tup[1], tup[2]))
                 else:
                     edge_dc[tup[1]] = [(tup[0], tup[1], tup[2])]

             for edge_labels, tup_ls in tqdm(edge_dc.items()):   # k=edge labels, v = list of tuples

                 tx = graph.begin()

                 for el in tup_ls:
                     source_node = nodes_matcher.match(name=el[0]).first()
                     target_node = nodes_matcher.match(name=el[2]).first()
                     if not source_node:
                         source_node = Node('Node', name=el[0])
                         tx.create(source_node)
                     if not target_node:
                         try:
                             target_node = Node('Node', name=el[2], node_labels=el[4], url=el[5], word_vec=el[6])
                             tx.create(target_node)
                         except:
                             continue
                     try:
                         rel = Relationship(source_node, edge_labels, target_node)
                     except:
                         continue
                     tx.create(rel)
                 tx.commit()

             return
```

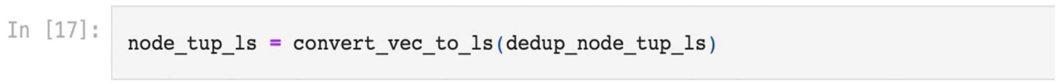
Figure 3.18: Creating some lists of tuples representing the node and edge lists

For our node list, we begin with the query subject (Barack Obama), which is in edge_ls[0][0] and put this in the variable orig_node_tup_ls. We assume a node label of Subject and no description or word vector. We then add all of the objects associated with the edges_word_vec_ls (obj_node_tup_ls) and combine that with the previous variable to create full_node_tup_ls, which we then dedupe into dedup_node_tup_ls.



```
In [15]: orig_node_tup_ls = [(edge_ls[0][0], '', ['Subject'], '', np.random.uniform(low=-1.0, high=1.0, size=(300,)))]
         obj_node_tup_ls = [(tup[2], tup[3], tup[4], tup[5], tup[6]) for tup in edges_word_vec_ls]
         full_node_tup_ls = orig_node_tup_ls + obj_node_tup_ls
         dedup_node_tup_ls = dedup(full_node_tup_ls)

In [16]: len(full_node_tup_ls), len(dedup_node_tup_ls)
```

Figure 3.19: Create the node list that will be used to populate the graph

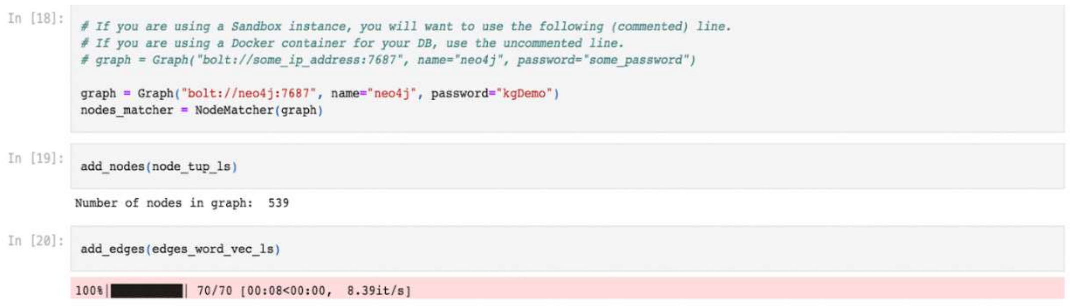


```
In [17]: node_tup_ls = convert_vec_to_ls(dedup_node_tup_ls)
```

Figure 3.20: Deduping the words for NLP.

Populate the graph

Here, we create a connection to the database, named Neo4j, which is active on the internal Docker network. In addition to supplying the login and password, we also develop a class for nodes that match (used when we establish the edges in the graph). In order to fill the database, we then add the nodes and edges.



```
In [18]:  # If you are using a Sandbox instance, you will want to use the following (commented) line.
          # If you are using a Docker container for your DB, use the uncommented line.
          # graph = Graph("bolt://some_ip_address:7687", name="neo4j", password="some_password")

          graph = Graph("bolt://neo4j:7687", name="neo4j", password="kgDemo")
          nodes_matcher = NodeMatcher(graph)

In [19]:  add_nodes(node_tup_ls)

         Number of nodes in graph:  539

In [20]:  add_edges(edges_word_vec_ls)

         100%|██████████| 70/70 [00:08<00:00,  8.39it/s]
```

Figure 3.21: Connecting Neo4j with google colab.

Entity disambiguation

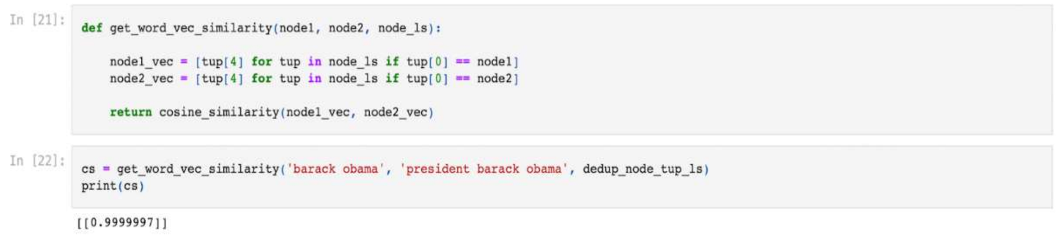
To assess the possibility that two nodes are the same thing, we will compute the cosine similarity of the word vectors of the target nodes in this notebook. Before doing anything above, keep in mind that you could accomplish this directly with spacy (which defaults to cosine similarity) by doing something like:

doc1 = nlp(text1)

doc2 = nlp(text2)

doc1.similarity(doc2)

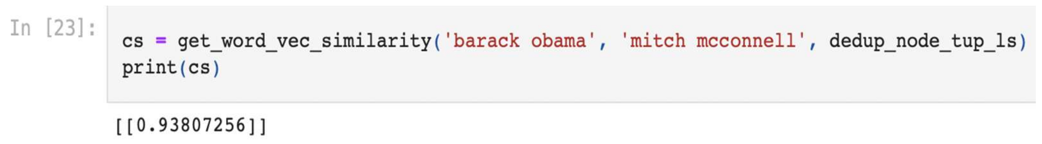
However, since the word vectors for each of the node descriptions above have already been calculated, we will just use the cosine similarity feature provided by scikit-learn. Because the nodes we are comparing might be in either the Barack or Michelle Obama node lists, we are now building a full node list.



```
In [21]:  def get_word_vec_similarity(node1, node2, node_ls):

              node1_vec = [tup[4] for tup in node_ls if tup[0] == node1]
              node2_vec = [tup[4] for tup in node_ls if tup[0] == node2]

              return cosine_similarity(node1_vec, node2_vec)

In [22]:  cs = get_word_vec_similarity('barack obama', 'president barack obama', dedup_node_tup_ls)
          print(cs)

          [[0.9999997]]
```

Figure 3.22: Defining a function to measure cosine similarity between words.

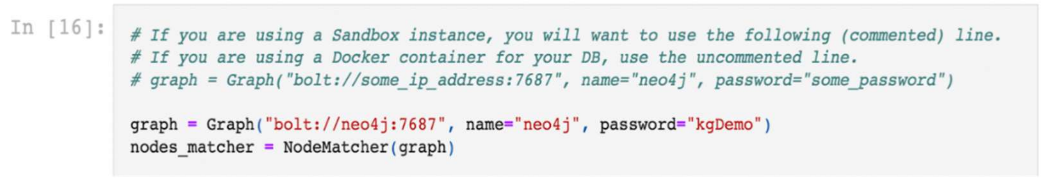
because these were identified as being the same by the Google Knowledge Graph. Although Google knew better and provided a precise description for each, the NER reported these as being distinct nodes. In knowledge graphs that are more complex, this may not always be the case.



```
In [23]: cs = get_word_vec_similarity('barack obama', 'mitch mcconnell', dedup_node_tup_ls)
         print(cs)

         [[0.93807256]]
```

Figure 3.23: Finding out the cosine similarity values.

It should be noted that you may start using this notebook right away if your database has already been pre-populated with the files in json files.

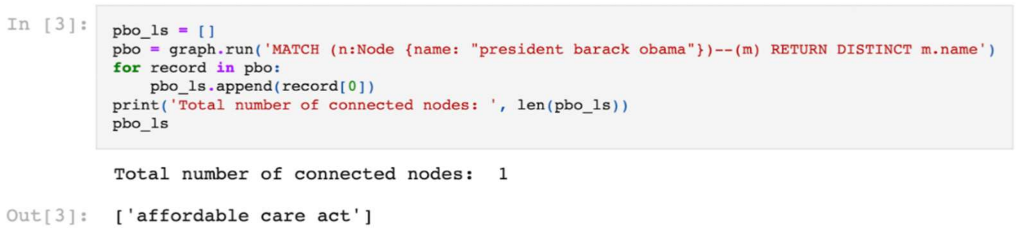


```
In [16]: # If you are using a Sandbox instance, you will want to use the following (commented) line.
         # If you are using a Docker container for your DB, use the uncommented line.
         # graph = Graph("bolt://some_ip_address:7687", name="neo4j", password="some_password")

         graph = Graph("bolt://neo4j:7687", name="neo4j", password="kgDemo")
         nodes_matcher = NodeMatcher(graph)
```

Figure 3.24: Setting up the Neo4j environment to populate the graphs.

Entity disambiguation

Remember how we examined the cosine similarity of word vectors in the last notebook? Let’s instead focus on the relationships that exist between our initial node, “oh bah m,” and “Barack Hussein Obama ii.” If there was a lot of resemblance, we may anticipate that there would be a lot of relationship overlap.



```
In [3]:  pbo_ls = []
         pbo = graph.run('MATCH (n:Node {name: "president barack obama"})--(m) RETURN DISTINCT m.name')
         for record in pbo:
             pbo_ls.append(record[0])
         print('Total number of connected nodes: ', len(pbo_ls))
         pbo_ls

         Total number of connected nodes:  1

Out[3]:  ['affordable care act']
```

Figure 3.25: Getting the total number of nodes.

Using machine learning with knowledge graphs

• Traditional ML uses a relational database-type model

◦ All data points are independent of each other

• Example: churn prediction based on user behaviour
• Graphs (and graph databases) treat relationships as a "first class citizen"
◦ Models can include homophily
• Example: churn prediction includes the churn of neighbours within the graph
◦ Models can also include the same data as the traditional, independent data point models

Algorithms used:

1. word2vec

Typically, word2vec produces one vector per word. The spacy implementation of vectorization averages the word vectors inside a text (sentence).

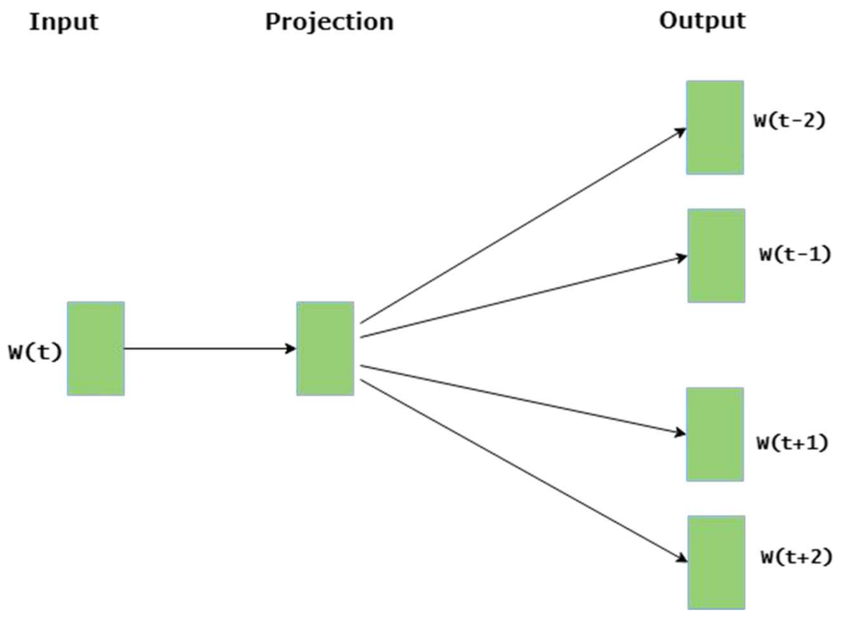


Figure 3.26: Figure explaining word2vec algorithm

2. node2vec

node2vec is an algorithm to generate vector representations of nodes on a graph. The node2vec framework learns low-dimensional representations for nodes in a graph through the use of random walks through a graph starting at a target node.

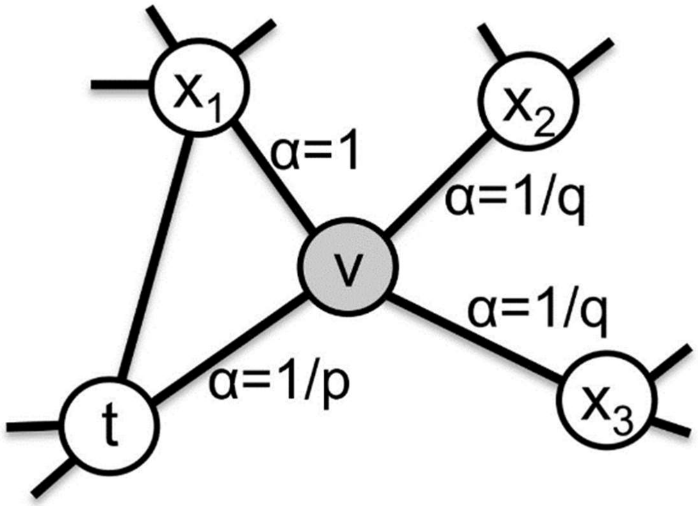
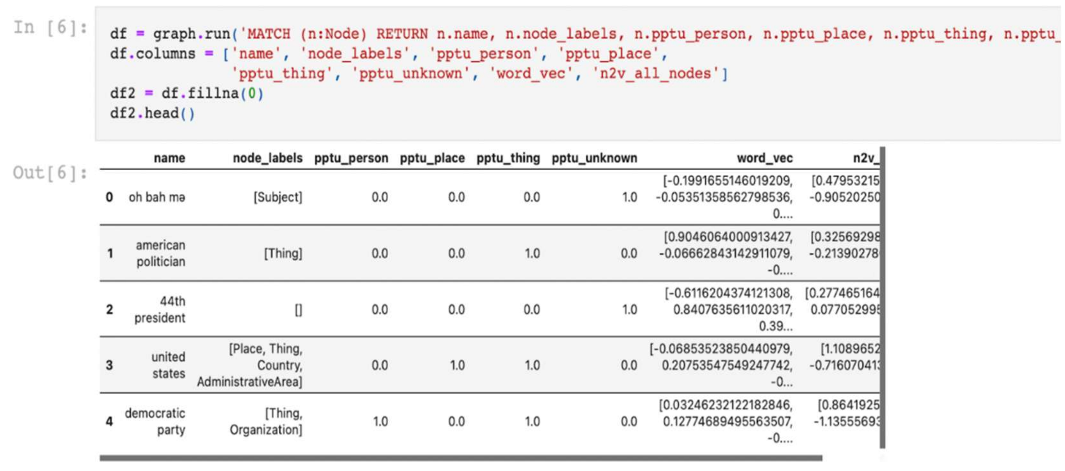


Figure 3.27: Figure explaining node2vec algorithm

Chapter-4
EXPERIMENTS & RESULT ANALYSIS

We can observe that the list of connected nodes for "oh bah m" contains 100% of the nodes connected to "Barack Hussein Obama ii". This is a good indication that the former and the latter could be the same thing. Connect here to graph now and do some ML. Here, we're going to make use of Py2neo's support for Cypher searches, and the ability to output the result to a Pandas Data Frame.

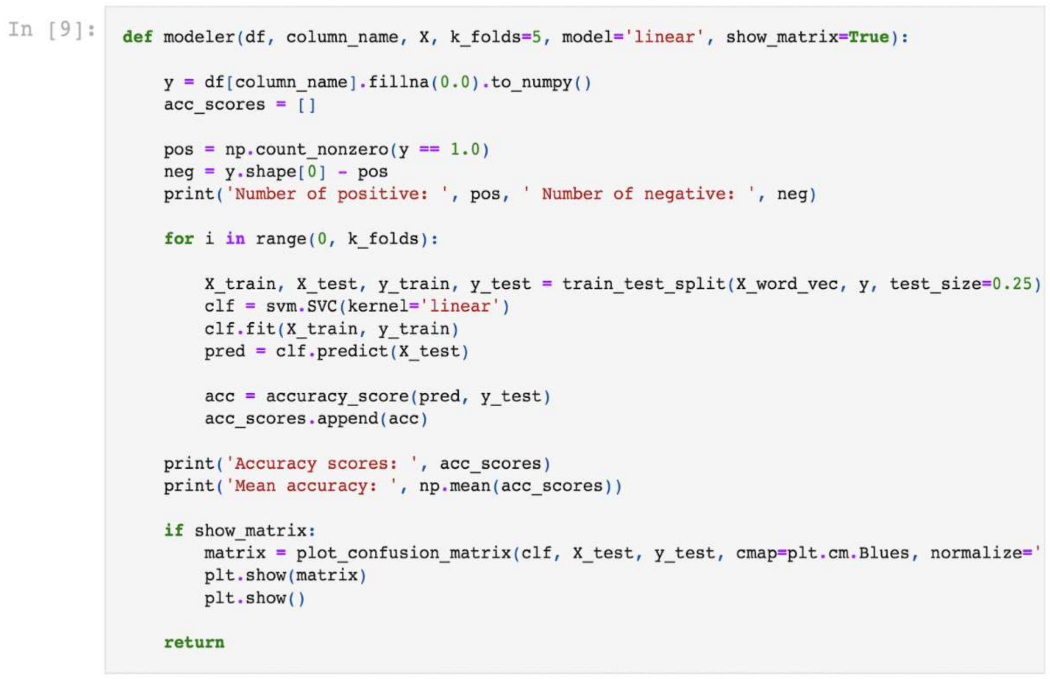


```
In [6]:  df = graph.run('MATCH (n:Node) RETURN n.name, n.node_labels, n.pptu_person, n.pptu_place, n.pptu_thing, n.pptu_
         df.columns = ['name', 'node_labels', 'pptu_person', 'pptu_place',
                       'pptu_thing', 'pptu_unknown', 'word_vec', 'n2v_all_nodes']
         df2 = df.fillna(0)
         df2.head()
```

```
Out[6]:
           name          node_labels  pptu_person  pptu_place  pptu_thing  pptu_unknown                word_vec          n2v_
   0  oh bah mə            [Subject]          0.0         0.0         0.0           1.0   [-0.1991655146019209,   [0.47953215
                                                                                          -0.05351358562798536,   -0.90520250
                                                                                                            0....
   1   american              [Thing]          0.0         0.0         1.0           0.0    [0.9046064000913427,   [0.32569298
      politician                                                                          -0.06662843142911079,   -0.21390278
                                                                                                           -0....
   2       44th                   []          0.0         0.0         0.0           1.0   [-0.6116204374121308,   [0.277465164
      president                                                                             0.8407635611020317,   0.07705299
                                                                                                         0.39...
   3     united        [Place, Thing,         0.0         1.0         1.0           0.0  [-0.06853523850440979,   [1.1089652
         states              Country,                                                      0.20753547549247742,   -0.71607041
                  AdministrativeArea]                                                                      -0...
   4  democratic              [Thing,         1.0         0.0         1.0           0.0   [0.03246232122182846,   [0.8641925
          party         Organization]                                                      0.12774689495563507,   -1.13555693
                                                                                                           -0....
```

Figure 4.1: Creating a variety of X variables for the node labels.

We are developing this function to execute a classifier using support vector machines in order to evaluate how well the various embeddings predict various labels. We are aware that this is an issue with several labels (Person, Place, Thing, Unknown), but in the interest of simplicity, we will just compare the prediction to single labels from that dataset. The interested reader is urged to experiment with more complicated models that can better manage the multi-label challenge. Additionally, we demonstrate below how substantially unbalanced each of the classes is. The interested party is urged to experiment with class balance to determine how this influences the perception as a whole.



```
In [9]: def modeler(df, column_name, X, k_folds=5, model='linear', show_matrix=True):

            y = df[column_name].fillna(0.0).to_numpy()
            acc_scores = []

            pos = np.count_nonzero(y == 1.0)
            neg = y.shape[0] - pos
            print('Number of positive: ', pos, ' Number of negative: ', neg)

            for i in range(0, k_folds):

                X_train, X_test, y_train, y_test = train_test_split(X_word_vec, y, test_size=0.25)
                clf = svm.SVC(kernel='linear')
                clf.fit(X_train, y_train)
                pred = clf.predict(X_test)

                acc = accuracy_score(pred, y_test)
                acc_scores.append(acc)

            print('Accuracy scores: ', acc_scores)
            print('Mean accuracy: ', np.mean(acc_scores))

            if show_matrix:
                matrix = plot_confusion_matrix(clf, X_test, y_test, cmap=plt.cm.Blues, normalize='
                plt.show(matrix)
                plt.show()

            return
```

Figure 4.2: Helper function to run SVM.

The confusion matrix, which informs us of the effectiveness of our graph model,

```
In [10]: modeler(df2, 'pptu_person', X_word_vec)

         Number of positive:  119  Number of negative:  538
         Accuracy scores:  [0.8545454545454545, 0.8666666666666667, 0.8606060606060
         606, 0.8, 0.8424242424242424]
         Mean accuracy:  0.8448484848484848485
```

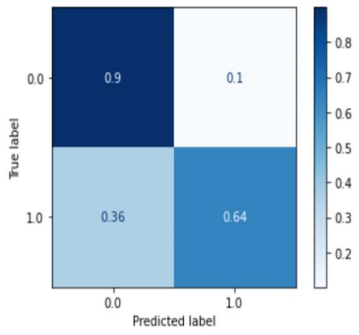
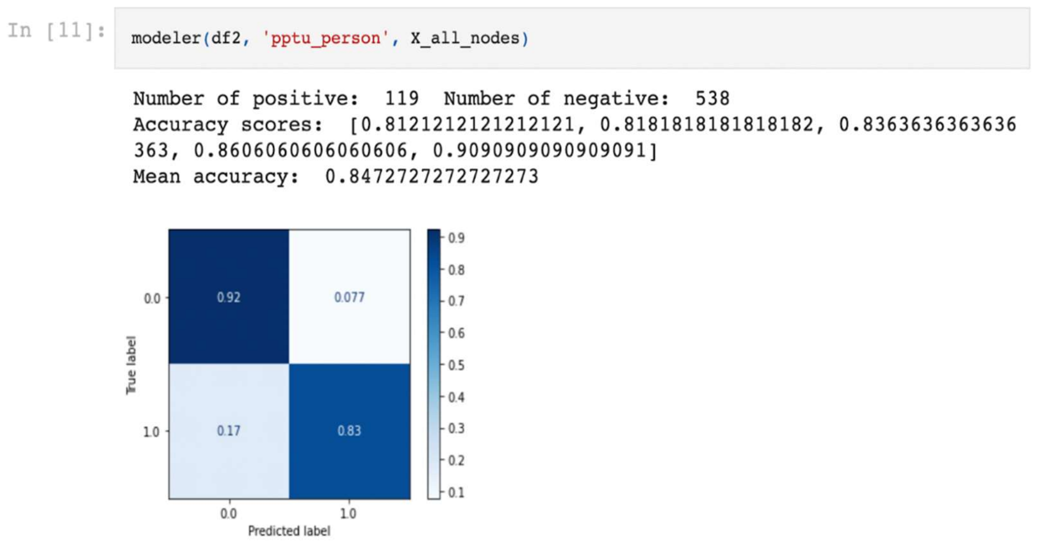


Figure 4.3: Confusion matrix showing us model performance.

In a similar vein, we will produce another confusion matrix to display the performance indicator of the alternative model. Differences between the predicted and observed values are represented in confusion matrices. The number of severely classified cases that were correctly identified is shown by the result "TN," which stands for True Negative. Similar to this, "TP" denotes the quantity of correctly identified positive cases and stands for True Positive. The abbreviation "FP" stands for "real negative instances that were incorrectly classified as positive," while "FN" stands for "genuine positive cases that were wrongly classified as negative." One of the metrics most frequently employed in classification is accuracy. The efficiency of a model can be determined using the formula below (through a confusion matrix).



```
In [11]: modeler(df2, 'pptu_person', X_all_nodes)

Number of positive:  119  Number of negative:  538
Accuracy scores:  [0.8121212121212121, 0.8181818181818182, 0.8363636363636
363, 0.8606060606060606, 0.9090909090909091]
Mean accuracy:  0.8472727272727273
```

Figure 4.4: Confusion matrix showing us the performance of our model.

```
In [12]: modeler(df2, 'pptu_place', X_word_vec)
```

```
Number of positive:  77  Number of negative:  580
Accuracy scores:  [0.9515151515151515, 0.9636363636363636, 0.9393939393939
394, 0.9272727272727272, 0.9030303030303031]
Mean accuracy:  0.9369696969696969
```

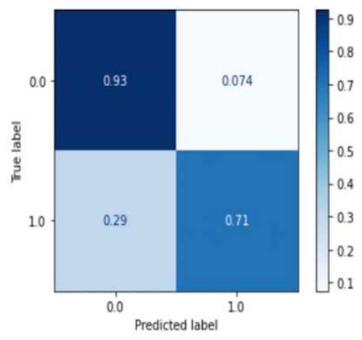


Figure 4.5: Confusion matrix showing us the performance of our model.

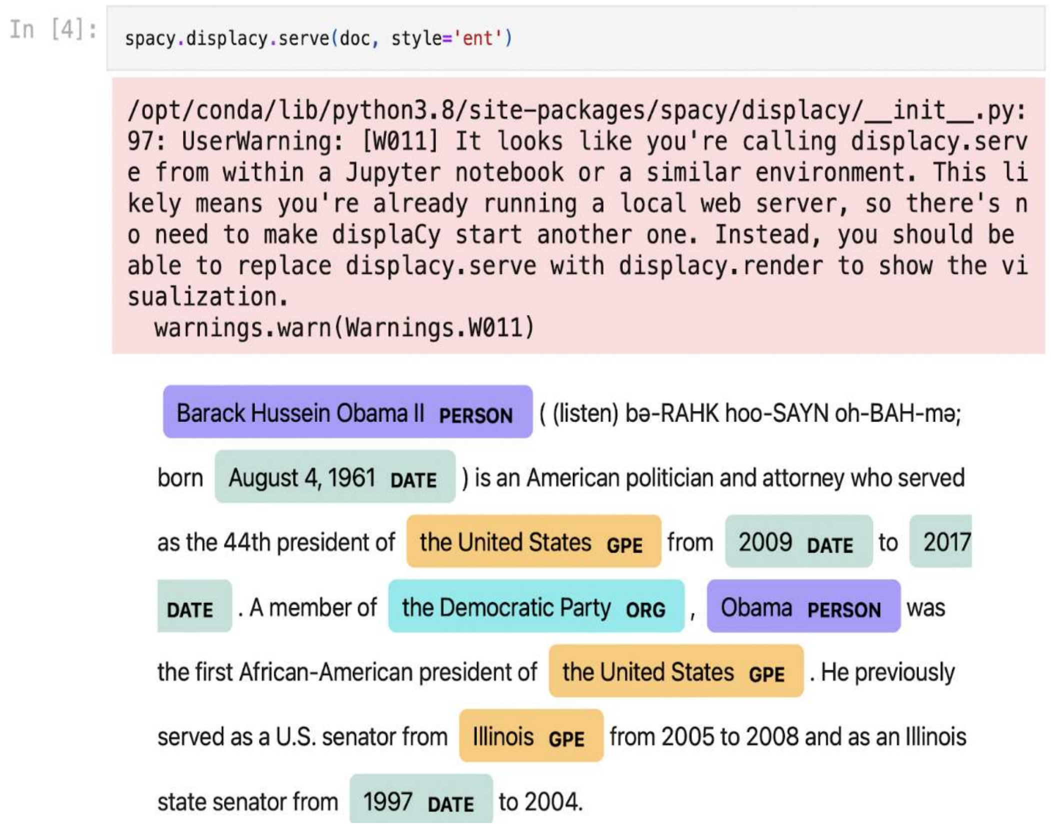
Next steps

You may try so many separate stuffs right here! I might take into account many factors, such as:

- Take time to tune the hyperparameters. This can be done for:
  - The spacy word embeddings
  - The graph embeddings
  - The ML models
- Attempting more complicated embedding methodologies, such GraphSAGE, which takes node aspects into consideration.
- Investigate various embeddings. There is a tonne more ways to build vectors that might be deployed for training the ML models than what we did here, where we used the spacy word vectors to create autoencoder for the nodes. Be imaginative!
- Work the class imbalance problem.
- In actuality, this graph is fairly modest. Work on expanding the graph by including more layers into it using either the Google Knowledge Graph or Wikipedia. We may anticipate that the graph embedding techniques will

start to truly shine beyond the word representations as the graph becomes bigger.

First, we acquire the Wikipedia summary for the search phrase we want to use, Barack Obama. The specified things in the text that will serve as the initial nodes for our graph may be seen using displacy. Despite a number of evident mistakes in the example below, the named entity recognition (NER) algorithm in spacy is still very effective at doing this task.



```
In [4]: spacy.displacy.serve(doc, style='ent')
```

```
/opt/conda/lib/python3.8/site-packages/spacy/displacy/__init__.py:
97: UserWarning: [W011] It looks like you're calling displacy.serv
e from within a Jupyter notebook or a similar environment. This li
kely means you're already running a local web server, so there's n
o need to make displaCy start another one. Instead, you should be
able to replace displacy.serve with displacy.render to show the vi
sualization.
  warnings.warn(Warnings.W011)
```

Barack Hussein Obama II PERSON ((listen) bə-RAHK hoo-SAYN oh-BAH-mə; born August 4, 1961 DATE) is an American politician and attorney who served as the 44th president of the United States GPE from 2009 DATE to 2017 DATE . A member of the Democratic Party ORG , Obama PERSON was the first African-American president of the United States GPE . He previously served as a U.S. senator from Illinois GPE from 2005 to 2008 and as an Illinois state senator from 1997 DATE to 2004.

Figure 4.6: Displacy to get Wikipedia summary paragraph.

Let's review some of the detected entities

1. Text cleaning

Some of the critters themselves will be filthy text. Consequently, we still wish to remove stop words and special characters, among other things. You can see our remaining list of lexical items by the time we reach the final two cells of the next row. This will serve as our initial scrape list for Wikidata. These are some

Wikidata interfaces assistance functions. Here, we are also going to set up the bot's connection to the website. Verification that we can link the bot to Wikidata.

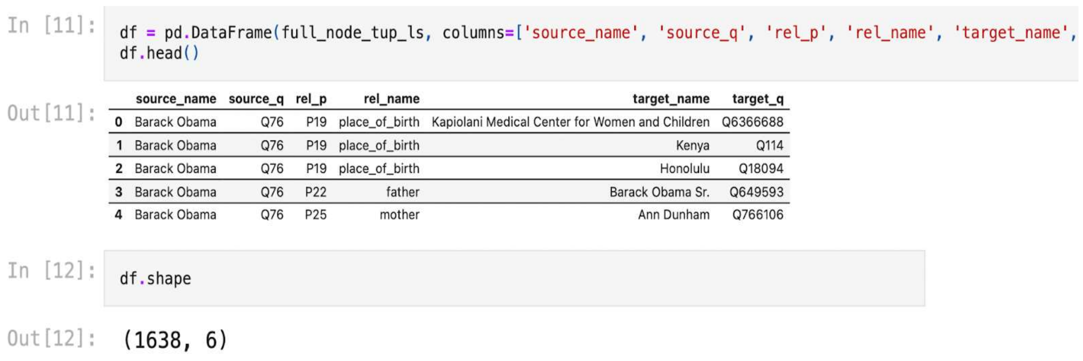
2. Scraping wikidata with our bot

The first step will be to locate each of our named things in Wikidata. This is accomplished by connecting each unique entity to a Wikidata Q-code, which is the indexing system that Wikidata employs for all entities. As you can see, not all of the entities are listed in Wikidata, perhaps because the text before the actual thing contains modifiers (ex: Republican nominee John McCain). Will we still be alright?

3. Get the verbs

These are indexed using the P-value and are known as “claims” or “statements” in Wikidata. P values vary by literally tens of thousands. We looked over the information and chose a series that I believed could be particularly intriguing. Without a doubt, this list has to be adjusted for the application/graph.

Depending on the size of your beginning list and the volume of traffic Wikidata is experiencing at any one time, this procedure might take several minutes. Even experience timeout problems. They will ultimately work themselves out. Grab a coffee cup. This takes about 10 to 12 minutes for Barack Obama’s entity list.

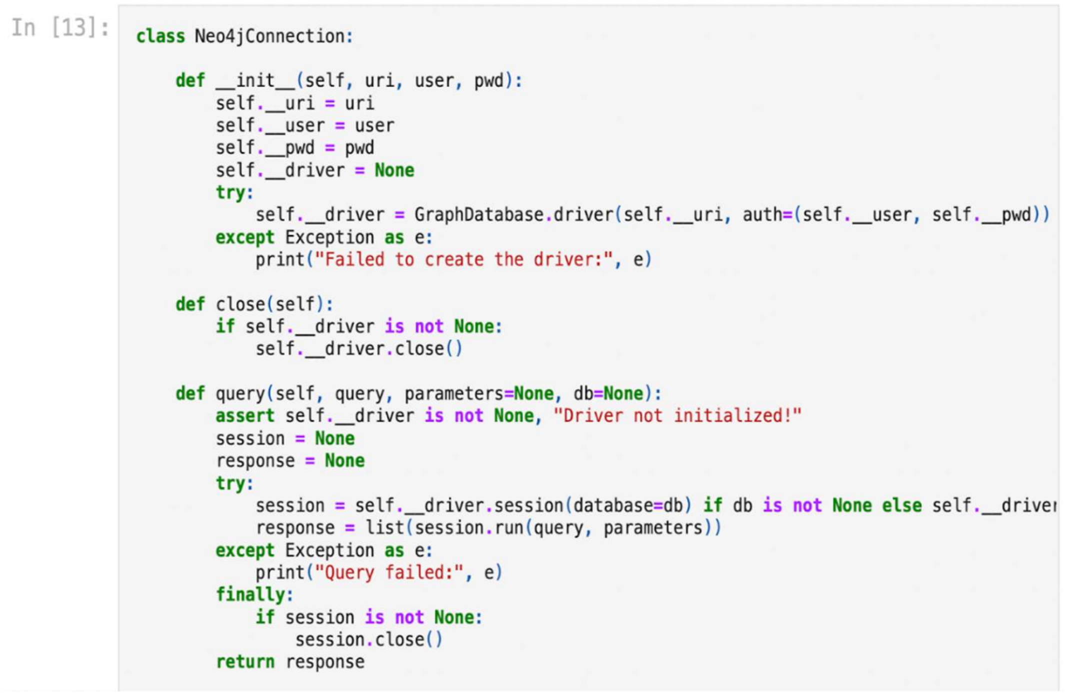


```
In [11]:  df = pd.DataFrame(full_node_tup_ls, columns=['source_name', 'source_q', 'rel_p', 'rel_name', 'target_name',
          df.head()

Out[11]:
     source_name  source_q  rel_p       rel_name                                  target_name   target_q
  0  Barack Obama      Q76    P19  place_of_birth  Kapiolani Medical Center for Women and Children  Q6366688
  1  Barack Obama      Q76    P19  place_of_birth                                        Kenya       Q114
  2  Barack Obama      Q76    P19  place_of_birth                                     Honolulu     Q18094
  3  Barack Obama      Q76    P22          father                            Barack Obama Sr.    Q649593
  4  Barack Obama      Q76    P25          mother                                  Ann Dunham    Q766106

In [12]:  df.shape

Out[12]:  (1638, 6)
```

Figure 4.7: Getting relation between source name and target name.

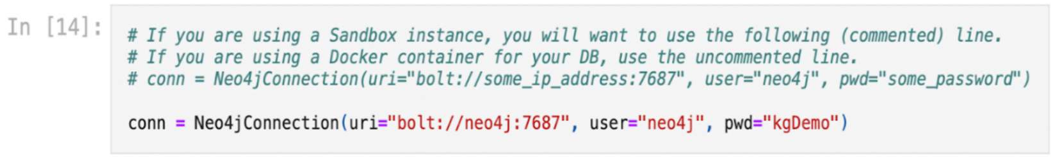
Connecting to Neo4j

We will use the same class to connect to Neo4j as before. We also implement a restriction on unique P-values, which has numerous potential advantages, especially as the graph grows.



```
In [13]: class Neo4jConnection:

             def __init__(self, uri, user, pwd):
                 self.__uri = uri
                 self.__user = user
                 self.__pwd = pwd
                 self.__driver = None
                 try:
                     self.__driver = GraphDatabase.driver(self.__uri, auth=(self.__user, self.__pwd))
                 except Exception as e:
                     print("Failed to create the driver:", e)

             def close(self):
                 if self.__driver is not None:
                     self.__driver.close()

             def query(self, query, parameters=None, db=None):
                 assert self.__driver is not None, "Driver not initialized!"
                 session = None
                 response = None
                 try:
                     session = self.__driver.session(database=db) if db is not None else self.__driver
                     response = list(session.run(query, parameters))
                 except Exception as e:
                     print("Query failed:", e)
                 finally:
                     if session is not None:
                         session.close()
                 return response
```

Figure 4.8: Connecting to Neo4.



```
In [14]: # If you are using a Sandbox instance, you will want to use the following (commented) line.
         # If you are using a Docker container for your DB, use the uncommented line.
         # conn = Neo4jConnection(uri="bolt://some_ip_address:7687", user="neo4j", pwd="some_password")

         conn = Neo4jConnection(uri="bolt://neo4j:7687", user="neo4j", pwd="kgDemo")
```

Figure 4.9: Setting up Neo4j environment.

Some helper functions

The graph is filled with data using the functions listed below. We do want to be able to assign a node label that is illustrative in order to enrich the graph. For this, we'll make use of the Wikidata claim "instance of" (P31). Barack Obama, for instance, is an example of a human whereas the United States is an example of a "sovereign state."

Conclusion of performance analysis

We have now completed populating our database. 1312 nodes and 1622 relationships must be obtained (once deduping in Cypher is completed and all nodes are attributed to the proper labels determined by P31). Once those are finished, we may go on to the third notebook, where we will demonstrate how to do some fundamental ML on the graph.

Embedding visualization

We will start by determining if our embeddings cluster at all or in a logical manner. However, as we employed a very high-dimensional space to generate those embeddings, we will do dimensionality reduction using t-SNE in order to facilitate visualisation. The two categories in this case are 1 for "is place" and 0 for "NOT IS Place." To ensure that we are heading in the correct direction, we would want for our embeddings to cluster in a distinct way.

We haven't actually tuned our embeddings at all. Tuning embedded systems is a unique art form that is outside the purview of this presentation. As you shall see, the embeddings don't cluster very well and the resulting ML models don't generate particularly impressive results as a result. The following is mostly meant as a demonstration. Making embeddings that work for you in a real-world, deployable application will clearly require considerably more work.

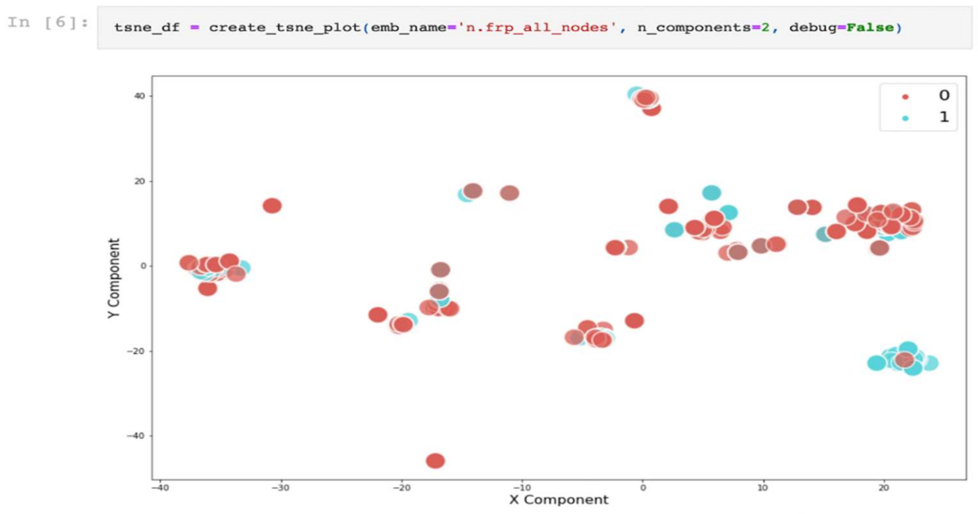

```
In [6]: tsne_df = create_tsne_plot(emb_name='n.frp_all_nodes', n_components=2, debug=False)
```

Figure 4.10: Dimensionality reduction for the sake of visualization

Problems we could solve:

1. Community/cluster detection: Community/cluster detection is a common problem in social network analysis where we want to identify groups of nodes that share common characteristics or interests. This problem has numerous applications, such as identifying groups of users with similar behavior in a social network or detecting groups of genes that are co-regulated in a biological network. Machine learning algorithms can be used to automatically identify these communities or clusters based on the patterns of interactions between nodes.

2. Node classification, link prediction: Node classification and link prediction are two important problems in graph analysis. Node classification involves assigning labels to nodes in a graph based on their characteristics or attributes. For example, we may want to classify nodes in a social network as "students" or "professors" based on their profiles. Link prediction involves predicting the likelihood of a link forming between two nodes in a graph. Machine learning algorithms can be used to learn patterns from the graph structure and node attributes to perform node classification and link prediction tasks.

3. Graph-to-graph classification: Graph-to-graph classification involves assigning labels to entire graphs based on their structural properties. For example, we may want to classify graphs as "social networks" or "protein interaction networks" based on their connectivity patterns. Machine learning algorithms can be trained to learn features from the graph structure and use them to classify entire graphs.

4. Unstructured text, NLP: Unstructured text refers to text data that is not organized in a structured way, such as tweets, news articles, and customer reviews. Natural Language Processing (NLP) is a field of study that focuses on the processing of unstructured text data. Machine learning algorithms can be used to extract information from unstructured text data, such as sentiment analysis, named entity recognition, and text classification.

Chapter-5
CONCLUSIONS

5.1 Conclusions

We've provided a brief introduction of knowledge graphs, which have grown in recent years. Knowledge graphs have drawn a lot of interest from different organisations and sectors as well as various research communities. The integration and extraction of value from various data sources at a large scale, whether in the context of a specific organisation, community, or more general collections of human knowledge, are problems that really are universal in nature and thus are handled by knowledge graphs. The most substantial process is that, for representing and integrating many kinds of large-scale data, graphs offer a simple, adaptable, intuitive, and powerful abstraction. Graphs have been used to represent data and information in a variety of domains, including Graph Algorithms and Theory, Graph Databases, Information Extraction, Information Representation, Machine Learning, the Semantic Web, and others. Now that these advances have been combined, knowledge graphs can benefit from them. Additionally, there are a number of fully complete graph query languages available now that offer navigational searches that can match paths of any length. There are now several graph databases and user interfaces that support these query languages. Neo4j has indeed been used by us to create graph databases using cypher query language. A fundamental concept in discrete mathematics that has use in all branches of computer technology is the directed labelled graph. Data graphs, taxonomies, and ontologies have been the most prominent uses of directed labelled graphs in artificial intelligence and databases. Traditionally, top-down design and manual knowledge engineering were used to construct tiny, specialised versions of these systems. Scale, bottom-up development, and a variety of construction methods set current knowledge graphs apart from traditional knowledge graphs. The extent and scope of the knowledge graphs we see now were never attained by the early semantic networks in AI. Because machine learning is largely data-driven and top-down schema design for data integration are difficult to construct, knowledge graph creation must be done from the bottom up. Last but just not least, we are considerably automating and using crowdsourced to augment traditional or manual knowledge engineering methodologies. The aforementioned events come together to provide traditional knowledge graph theory and algorithms a new degree of significance. The architecture of a knowledge graph's schema and its semantic definition are still crucial, even when we build it from the bottom up. While some steps in the development of a knowledge graph may be automated, manual validation and human monitoring are still needed. The possibility for combining knowledge graph methodologies with cutting-edge machine learning, crowdsourcing, and scale computing technologies is opened up by this synergy.

5.2 Future Scope

Knowledge graphs may be used in so many different aspects of daily life and goods, knowledge graphs are gradually becoming more popular. A typical example is all of the voice assistant products we use and every online recommendation. A significant number of merchandises now contain some form of intelligence, and almost all firms and enterprises are looking for ways to incorporate more intelligence into their goods and services in order to gain market share and improve decision-making. Global awareness of these challenges in the field of AI is currently rising. Recent developments in knowledge representation have produced outstanding outcomes. The basis of advancements is the use of graph representation for conceptually capturing interpretation and making the results available as machine-readable contextual data. Graphs have the additional benefit of being crawlable. They may also be used to automatically and elegantly combine knowledge from many sources as a foundation for reasoning if they have the right level of semantic representation. Information graphs are therefore an effective technique to express holistic understanding for usage throughout the organisation and are well suited for publishing reference material in a solution-independent and future-proof manner. As we said in the preamble, labelled directed graphs have been used for knowledge representation from the early days of AI. Applications with intelligent behaviour and billions of users include search engines, personal assistants, and recommendation systems. It is now generally acknowledged that these applications perform appropriately when knowledge graphs are used. Using a knowledge graph, a personal assistant may complete more tasks more accurately and efficiently. A recommender system with a graph database provides the user with better recommendations. Similar to this, when a search engine has access to a knowledge graph, it may produce superior results. As a consequence, these applications give a compelling background and a list of prerequisites for knowledge graphs to affect product portfolio offers.

As a result, knowledge graphs energize AI systems by giving them incentive and a set of needs. The capacity to efficiently and at scale generate the knowledge graph for application is also improving thanks to AI techniques.

REFERENCES

[1] Yan, J., Wang, C., Cheng, W., Gao, M. and Zhou, A., 2018. A retrospective of knowledge graphs. Frontiers of Computer Science, 12(1), pp.55-74.

[2] Ayinuer, N., Ruxianguli, A. and Yasen, A., 2022, May. Design and Research of Unstructured Data Knowledge Graph Toolbased on Neo4j Graph Database. In 2022 11th International Conference on Communications, Circuits and Systems (ICCCAS) (pp. 296-300). IEEE.

[3] Ive, Julia. "Natural Language Processing: A Machine Learning Perspective by Yue Zhang and Zhiyang Teng." (2022): 233-235.

[4] Achichi, Manel, Zohra Bellahsene, and Konstantin Todorov. "Legato: Results for oaei 2017." In OM: Ontology Matching, no. 2032, pp. 146-152. 2017.

[6] Hogan A, Blomqvist E, Cochez M, d'Amato C, Melo GD, Gutierrez C, Kirrane S, Gayo JE, Navigli R, Neumaier S, Ngomo AC. Knowledge graphs. ACM Computing Surveys (CSUR). 2021 Jul 2;54(4):1-37.

[7] Ji, Shaoxiong, Shirui Pan, Erik Cambria, Pekka Marttinen, and S. Yu Philip. "A survey on knowledge graphs: Representation, acquisition, and applications." IEEE Transactions on Neural Networks and Learning Systems 33, no. 2 (2021): 494-514.

[8] Noy, Natasha, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. "Industry-scale Knowledge Graphs: Lessons and Challenges: Five diverse technology companies show how it's done." Queue 17, no. 2 (2019): 48-75.

[9] Liu, Haibo, Guoyi Jiang, Linhua Su, Yang Cao, Fengxin Diao, and Lipeng Mi. "Construction of power projects knowledge graph based on graph database Neo4j." In 2020 International Conference on Computer, Information and Telecommunication Systems (CITS), pp. 1-4. IEEE, 2020.

[10] Yang, Xu, Ziyi Huan, Yisong Zhai, and Ting Lin. "Research of personalized recommendation technology based on knowledge graphs." Applied Sciences 11, no. 15 (2021): 7104.

[11] Dessì, Danilo, Francesco Osborne, Diego Reforgiato Recupero, Davide Buscaldi, and Enrico Motta. "Generating knowledge graphs by employing natural language processing and machine learning techniques within the scholarly domain." Future Generation Computer Systems 116 (2021): 253-264.

[12] Dou, Jinhua, Jingyan Qin, Zanxia Jin, and Zhuang Li. "Knowledge graph based on domain ontology and natural language processing technology for Chinese intangible cultural heritage." Journal of Visual Languages & Computing 48 (2018): 19-28.

[13] Miller, Justin J. "Graph database applications and concepts with Neo4j." In Proceedings of the southern association for information systems conference, Atlanta, GA, USA, vol. 2324, no. 36. 2013.

[14] Guu, Kelvin, John Miller, and Percy Liang. "Traversing knowledge graphs in vector space." arXiv preprint arXiv:1506.01094 (2015).

[15] Fensel, Dieter, U. Simsek, Kevin Angele, Elwin Huaman, Elias Kärle, Oleksandra Panasiuk, Ioan Toma, Jürgen Umbrich, and Alexander Wahler. Knowledge graphs. Springer International Publishing, 2020.