A Framework for Building Knowledge Graph from Unstructured Data:

Pradhan Mantri Jan Arogya Yojana (PMJAY)

Project report submitted in partial fulfillment of
the conditions for the award of the degree of

Master of Technology
(Computer Science)

Suvam Gurung
(Regd.No.: 20559)

SRI SATHYA SAI INSTITUTE OF HIGHER LEARNING
(Deemed to be University)

Department of Mathematics and Computer Science
Prasanthi Nilayam Campus
March 2023

Acknowledgements

"The purpose of education is to foster virtues. This is the essence of all education."

-Sri Sathya Sai Baba

This entire work would never have been completed without the guidance, support, kindness, love and understanding of so many, to whom I will ever be grateful and in debt. I will always, to my extent of capability, be ever ready to do what I can for them.

I offer my foremost gratitude along with this humble endeavour of mine at the lotus feet of the one who has the most beautiful countenance of all time, Bhagawan Sri Sathya Sai Baba. I will ever be grateful to Him for his blessings upon me during the moments of joy, ignorance, pain, loss, hardships, the steep ups and downs of life till the moment I shall exist.

So proud was my mother, Late Smt. Chitrakala Gurung, upon the fact that I am a Sai Student, so much that she would give this fact the topmost priority above everything else. And hence this motivation to complete this work, to fulfil her dream to see me graduate. I dedicate this work and thank her for every innocent love and care she showered upon me. It will always be engraved in my heart. Thank You Mom!

I am grateful to my father, Sri. Nabin Gurung, for his respectful gestures of love and care. And always being there to pat me and say," Don't worry, you focus on your thing" even at the hardest of all time. I thank my sister, Sushri Suviyam Gurung, for being a constant source of inspiration and strength, inspiring me to be mentally strong and mindful at trying times. I also thank Miss. Upashna Gurung, for being there to love and support me emotionally and reinstalling faith in Swami in the darkest of times.

I shall be eternally grateful to Dr. Pallav Kumar Baruah, Dr. A S Vishwanathan, Dr. R Raghunatha Sharma, Research Scholars Sri Sai Sanwid Pradhan, Sri Manohar Bhujel, Sri Prajjwal Chettri and Sri Nitesh Tamang for helping me when I was incapable of any action during my most difficult times. I will always be there for them in any possible manner.

I thank Sri. Basanta Sharma, Sri. Nishal Pradhan, Sri. Deep Narayan Pradhan, Sushri Norkey Wangmu Yolmo, who are all alumni of SSSIHL, Sri Yowan Thapa, Sushri Sabina Pradhan, Sri. Amit Sood, Life-Coach and Sri. Ashok Thakuri, Bal Vikas Guru, for their timely kind support, counselling, guidance and motivation. I shall always be in

debt to their kindness.

I express my heartfelt gratitude to my supervisors, Dr. R Raghunath Sharma, DMACS, SSSIHL, Prasanthi Campus and Sri. P Sunil Kumar, DMACS, SSSIHL, Muddenahalli Campus for their advice, patience, guidance, motivation, dedication and all the more their kindness and understanding during the entire process and till now.

I thank Dr. Prabhakar Akella, DMACS, SSSIHL, PSN Campus and Dr. Avadesh Kumar, DMACS, SSSIHL, PSN Campus for their expertise and guidance. I am also very grateful to Dr. Satya Sai Mudigonda, SSSIHL, PSN Campus, for providing me with accommodations having a conducive environment to complete my project as well as his expertise, knowledge, guidance and care.

I am thankful to the acting Vice-Chancellor Prof. B Sai Giridhar, acting Registrar Dr. Pallav Kumar Baruah, Controller of Examinations Sri Sanjay Sahani and the Director of PSN Campus Dr. G Raghavender Raju for giving me this opportunity to undertake this project.

I express my deepest sense of gratitude to all those who have directly or indirectly helped me in this project. I beg pardon if I have forgotten to mention their names here, but they shall be remembered forever with deepest sense of gratitude.

- Suvam Gurung

Abstract

Ayushman Bharat–Pradhan Mantri Jan Arogya Yojana" (PM-JAY), launched in India in September 2018, is one of the largest social security schemes of the world in the health sector. The scheme intends to move the nation closer to 'Universal Health Coverage' (UHC) to achieve Sustainable Development Goals (SDG) by providing a health coverage of Rupees 5 lakh per family per year with a minimal premium, for secondary and tertiary care hospitalisation. This scheme like many other government or private schemes has unstructured data in the form of PDFs as guidelines and other documents.

Knowledge Graph is a graph of data that represents real world entities and relationships between them. It will be discussed in the later chapters. The integration of Knowledge Graphs has become a new norm in the IT world for efficient searches, recommendation systems, smart assistants, connecting data with ML models and so on.

Natural Language Processing fills the gap between human's expression and computer's understanding which aids in increased accuracy and performance of NLP Models that still shows a promising future.

In this thesis, we have attempted to build a framework to fill the gap between unstructured source of data and creation of Knowledge Graph with domain specific NLP Model as the bridge. The Knowledge Graph thus built is visualised using Neo4j platform. This entire project is a step-by-step procedure to build the above mentioned framework accomplished using Google Colaboratory. Python libraries like Apache Tika, SpaCy, OpenAI, Pandas and Regular Expressions along with Cypher Query Language and Neo4j Software have been utilised to realise the proposed framework.

The main highlight of this project is the development of a domain-specific Named Entity Recognition (NER) component trained using spaCy which aids the generation of Knowledge Graph for a social security scheme such as AB-PMJAY. Such a framework for a social security scheme is quite unique. We have focused on the creation of Knowledge Graph for the general process flow of AB-PMJAY scheme with respect to different stakeholders.The knowledge graph thus created will show the comprehensiveness of process flow of PMJAY as a visual composition of nodes and relationships infused with properties that can be queried by different stakeholders as per their interest.

Keywords: Domain-Specific NER Component, Knowledge Graph, AB-PMJAY, Neo4j, Cypher Query Language, SpaCy, Regular Expressions

Contents

List of Figures

Chapter 1

Introduction

"The purpose of education is not merely to sustain the body; it should broaden man's mind and make him an ideal and virtuous person"

-Sri Sathya Sai Baba

With the availability of enormous amounts of data, the scope of every IT Field has found a new ignition with rising advancement in technology. We have a vast repository of data available in myriad forms and structures. It would be extensively useful if they are made available in a comprehensive form, easy to access, analyse and query as per our requirement. Being able to analyse data like we analyse real-world entities and decipher relations amongst them can open up valuable opportunities for new discovery, experimentation and analysis. But a huge portion of this data exists in unstructured format in forms PDFs and texts. This necessitates the availability of a framework that extracts structure out of unstructured data in the form of a Knowledge Graph which can be queried as per one's interest.

This framework is targeted to address the issues of limitations of default NLP Model, data refining and representations of the extracted data that are domain-specific thereby filling the gap between human's expression in the form of unstructured data and the computer's ability to interpret and present them. And also to provide a platform for visualisation, experimentation and analysis of the data, more specifically, the data related to process flow of Ayushman Bharat - Pradhan Mantri Jan Arogya Yojana (PMJAY)

national health scheme.

1.1 Motivation

The extraction of data from unstructured domain-specific text faces problems such as default NLP Models not being able to recognize and label the domain-specific terms accurately, data refining is not sufficiently productive and representation of the data happens to be in forms of tabulated forms/csvs. Thus resulting in misinterpreted cumbersome analysis and outcome. This extraction process mainly depends on the power of Natural Language Processing Models and Regular Expressions. Every domain has their domain-specific jargon that does not fall under the umbrella of default language models so a domain-specific NLP model as well as Regular Expressions for data refinement is necessary [18]. As per the problem of data representation, Knowledge Graph is proving to be taking up the market above traditional forms of tabulated representations [4].

Pradhan Mantri Jan Aarogya Yojana (PMJAY) is a massive scheme that has the potential to improve the health outcomes of millions of people in India by providing them with access to quality healthcare services. It covers over 10.74 crore poor and vulnerable families (approximately 50 crore beneficiaries), making it one of the largest health insurance schemes in the world. The scheme provides a benefit cover of up to Rs. 5 lakh per family per year for secondary and tertiary hospitalisation. PMJAY has a network of over 24,000 empaneled hospitals across India, including private and public hospitals [2]. The government has allocated a budget of Rs. 6,400 crore for the implementation of PMJAY in the financial year 2021-22 [14].

In the state of Jharkhand, the government conducted a door-to-door survey to identify eligible households and enrolled them in the scheme. The process involved verifying the identity and eligibility of the beneficiaries and issuing a PMJAY card, which is used to avail of the benefits of the scheme [2]. A woman in Uttar Pradesh who was suffering from a heart condition was able to undergo a successful surgery at a private hospital under the PMJAY scheme. The hospital provided the treatment on a cashless basis, and the entire cost of the treatment was covered by PMJAY [14]. A hospital in Bihar successfully

claimed reimbursement for the treatment of a patient under the PMJAY scheme. The hospital received the reimbursement amount within a few days of submitting the claim [12]. The government of Madhya Pradesh has set up a real-time monitoring system to track the utilisation of PMJAY services in the state. The system provides information on the number of beneficiaries enrolled, the number of claims submitted and settled, and the quality of services provided by the empaneled hospitals [12].

But there seem to be a lack of visualised and interactable depiction/representation of the process flow involving the stakeholders and juncture of their interaction with the process. A beneficiary being able to see all their points of interactions in the scheme and procedures/types of treatment would support ISA in spreading knowledge of the scheme. And same applies to the stakeholders, the stakeholders would have a comprehensive idea of the scheme at their fingertips and could use the data/representation as per their innovation and analysis.

The Pradhan Mantri Jan Aarogya Yojana (PMJAY) scheme contains a large set of unstructured data in the form of guideline PDFs [2]. NLP models can be trained as per the domain, catering to the need of understanding domain-specific words/jargons. Entities and relations can then be extracted from sentences generated through ER diagrams of the domain's process flow of the scheme. And for visualised representation and queries, Knowledge Graph can be used though Neo4j platform.

With these incentives this framework is built so that the domain can make use of it to visualise their own repository of unstructured data.

1.2 Contributions

This project makes the following contributions:

1. A Framework for building Knowledge Graph from Unstructured Data: This framework takes unstructured data as input and generates a cypher script which can be imported to Neo4j for visualisation and querying the data.
2. Domain Specific NLP Model for AB-PMJAY: The Named Entity Recognition (NER)

component of the spacy model is trained to the domain-specific words and saved as an NLP Model.

3. Corpus to train spacy's NER component crafted for PMJAY: A corpus containing around 700 sentences with annotations as per PMJAY scheme is created to train the domain specific NLP model.
4. Regular Expressions for PMJAY's Data Refinement: A set of regular expressions to refine unstructured data from the domain of PMJAY has been created. Mainly addressing the problem of ambiguous variations of words.
5. Python Code for Training NLP model: Python code for training Named Entity Recognizer (NER) component of spacy model.
6. Data Refining Code: Data refining python code using the domain-specific regular expressions to refine the unstructured input data.
7. Performance assessment of NLP Models: The NER component could be trained either from existing NLP models or from scratch. A performance evaluation is done of both the models.
8. Cypher Script for creating PMJAY Process Flow Knowledge Graph: The output of the framework is this cypher script which will create a Knowledge Graph for PMJAY Process Flow when imported to Neo4j software.
9. Manual for Neo4j: Installation manual and guide to import cypher script.

1.3 Thesis Outline

The rest of the report is organised as follows.

- Chapter 2 gives a summary of literature review done in this project.
- Chapter 3 gives the overview of the framework.
- Chapter 4 describes the implementation of the framework.
- Chapter 5 describes the performance analysis of default and domain-specific NLP Models.
- Chapter 6 describes a use case with a dummy record of PMJAY Process Flow.
- Chapter 7 describes the conclusion and future work that can be done.

Chapter 2

Literature Survey

In this chapter we will go through a concise survey of the resources and technologies used in this project.

2.1 Ayushman Bharat - Pradhan Mantri Jan Arogya Yojana (AB-PMJAY)

Ayushman Bharat - Pradhan Mantri Jan Arogya Yojana (AB-PMJAY) is a flagship healthcare program launched by the Government of India in September 2018. The program aims to provide health coverage to vulnerable and economically disadvantaged families in India, including rural households and urban poor. It is a part of the larger Ayushman Bharat scheme, which includes both health and wellness components. The health component, AB-PMJAY, provides cashless and paperless health coverage up to Rs. 5 lakhs per family per year for secondary and tertiary hospitalisation. The wellness component, Health and Wellness Centres (HWCs), aims to provide comprehensive primary healthcare services to the underserved and underprivileged populations across the country. [2]

Its scheme covers more than 10.74 crore poor and vulnerable families, or approximately 50 crore individuals in India. The scheme provides coverage for more than 1,500 medical procedures, including heart surgeries, knee replacements, and cancer treatments.

It also covers pre- and post-hospitalization expenses, as well as transportation costs for patients who need to travel to the hospital. [3]

One of the key features of AB-PMJAY is its technology-driven approach. The scheme uses a digital platform to manage the health coverage of beneficiaries, including eligibility verification, claims processing, and payments. The platform, called the Ayushman Bharat Pradhan Mantri Jan Arogya Yojana Health Benefit Portal, also provides real-time access to health data, allowing policymakers to monitor and evaluate the scheme's performance.[9]

Another important feature of AB-PMJAY is its focus on quality and accountability. The scheme has a robust monitoring and evaluation system in place, which includes regular audits and assessments of hospitals and other healthcare providers. It also has a grievance redressal mechanism to ensure that beneficiaries can provide feedback and file complaints if they face any issues with the scheme. [9] AB-PMJAY has been widely praised for its potential to transform the healthcare landscape in India. By providing health coverage to vulnerable and economically disadvantaged families, the scheme has the potential to improve health outcomes, reduce financial distress, and promote inclusive growth in the country. However, the scheme also faces several challenges, including the need for better implementation, coordination, and sustainability, as well as the need to address the underlying structural issues in the healthcare system in India.[3]

Overall, AB-PMJAY is a promising initiative that has the potential to improve the lives of millions of people in India. With its focus on quality, accountability, and technology, the scheme represents an important step towards building a more equitable and accessible healthcare system in the country.[2]

The processes of PMJAY that are essential to this project are discussed in subsequent sections.

2.1.1 Process Flow of PMJAY

Here we explain the process involved in PMJAY scheme that show how a general patient can avail the benefits of PMJAY and what happens once recognized as PMJAY Beneficiary

under the scheme.

AB PMJAY Overview

A patient goes through the following process when s/he want to avail the benefits under PMJAY scheme as depicted in 2.1

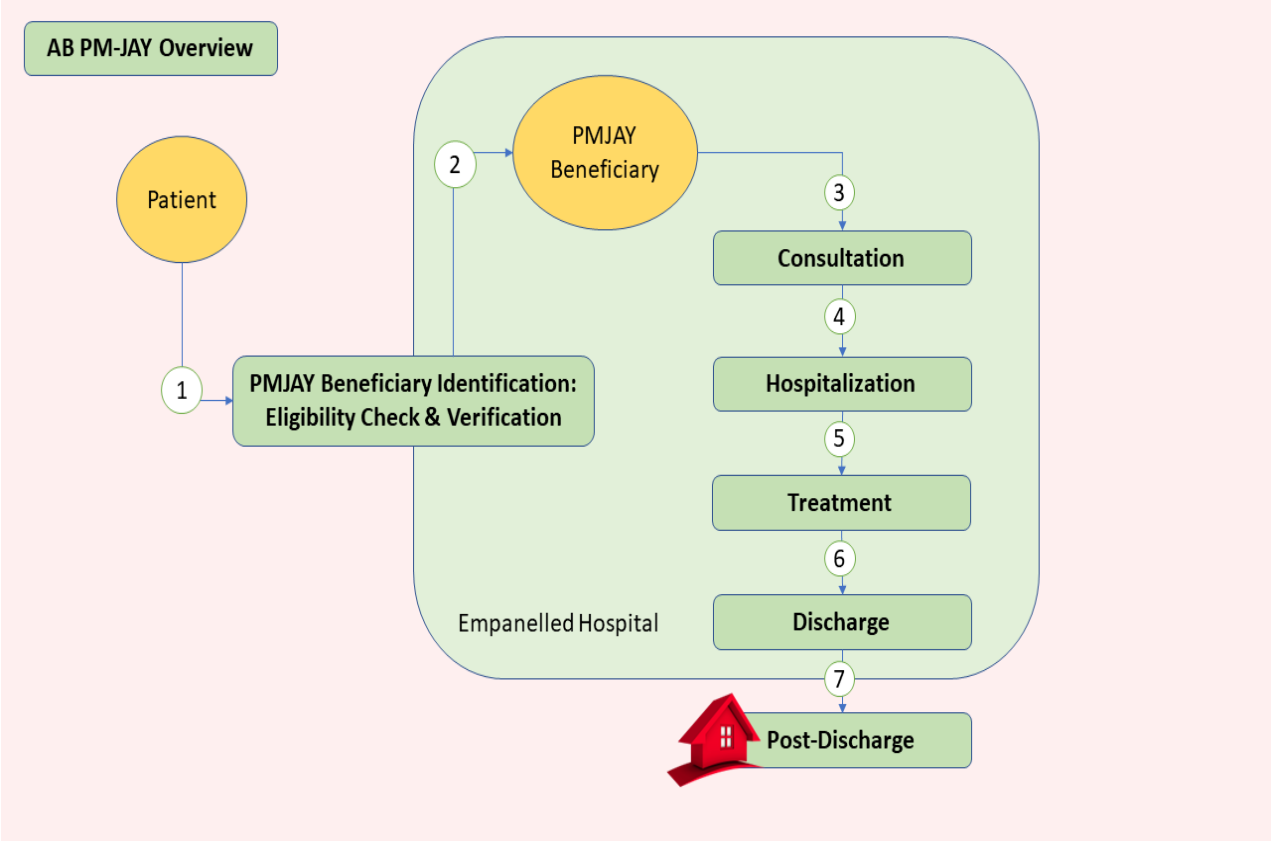


Figure 2.1: High-Level Overview of AB-PMJAY

. The patient is first checked for eligibility and verified as PMJAY beneficiary. Once the patient is verified as a beneficiary, s/he is given consultations from doctors/ specialist who will decide upon further course of action whether hospitalisation, diagnostics or medication is necessary. Once the beneficiary is hospitalised, the treatment is selected and carried out. On completion of the treatment, the beneficiary is discharged with necessary reports and medication is provided for a certain period of time. Post-discharge feedbacks are requested from the beneficiary.

PMJAY Beneficiary Identification

If a patient wants to avail the benefits under PMJAY scheme, s/he will have to be eligible and verified as a beneficiary. The patient should check his/her eligibility through either "Am I Eligible" App/Website, Call-Centre, PMJAY Kiosk or EHCP Registration desk at the empanelled hospital or CSC as depicted in 2.2 Once the patient is found

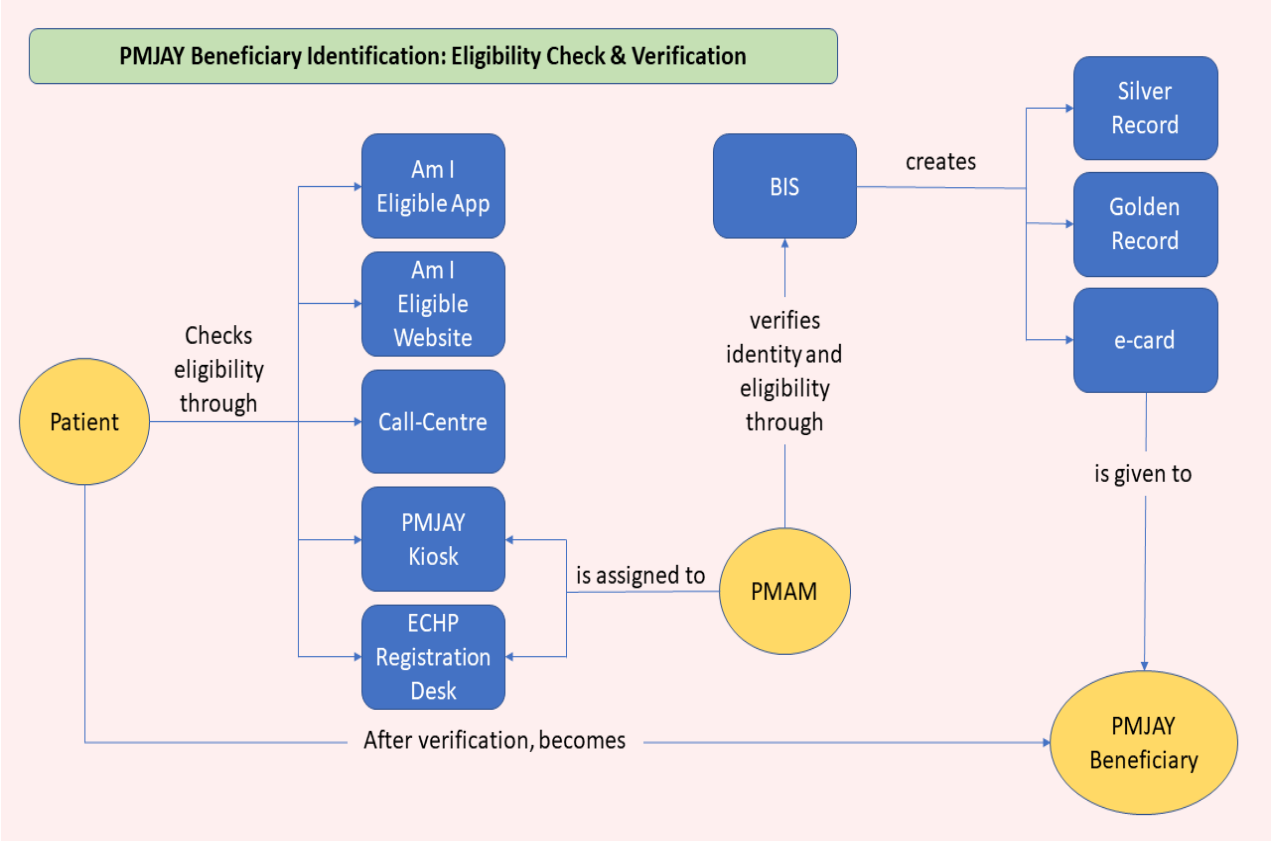


Figure 2.2: PMJAY Beneficiary Identification

eligible, Pradhan Mantri Arogya Mitra (PMAM) sends the data to Beneficiary Identification System which verifies the patient and creates Silver Record for every first-time entry, Golden Record for verified patients and generates an e-card which is given to the patient who is now a beneficiary.

If the patient is not eligible under PMJAY scheme, the patient will have to undergo the normal procedure of treatment and pay all the relevant charges like a regular patient.

Consultation

Once the patient is recognized as a PMJAY Beneficiary, s/he is given a consultation regarding their condition from a general doctor/ specialist who will decide if the beneficiary will require hospitalisation and/or diagnostics and prescribe medication as needed. If the beneficiary requires hospitalisation, a pre-authorization form is filled by the consulting doctor/specialist after recommending the medical package or treatment needed for the case. This form is given to a Medical coordinator (MEDCO) who in turn hands it to the PMAM. PMAM will forward the request to higher authorities for approval. If the beneficiary does not require hospitalisation, s/he needs to pay the relevant charges. It is depicted in 2.3

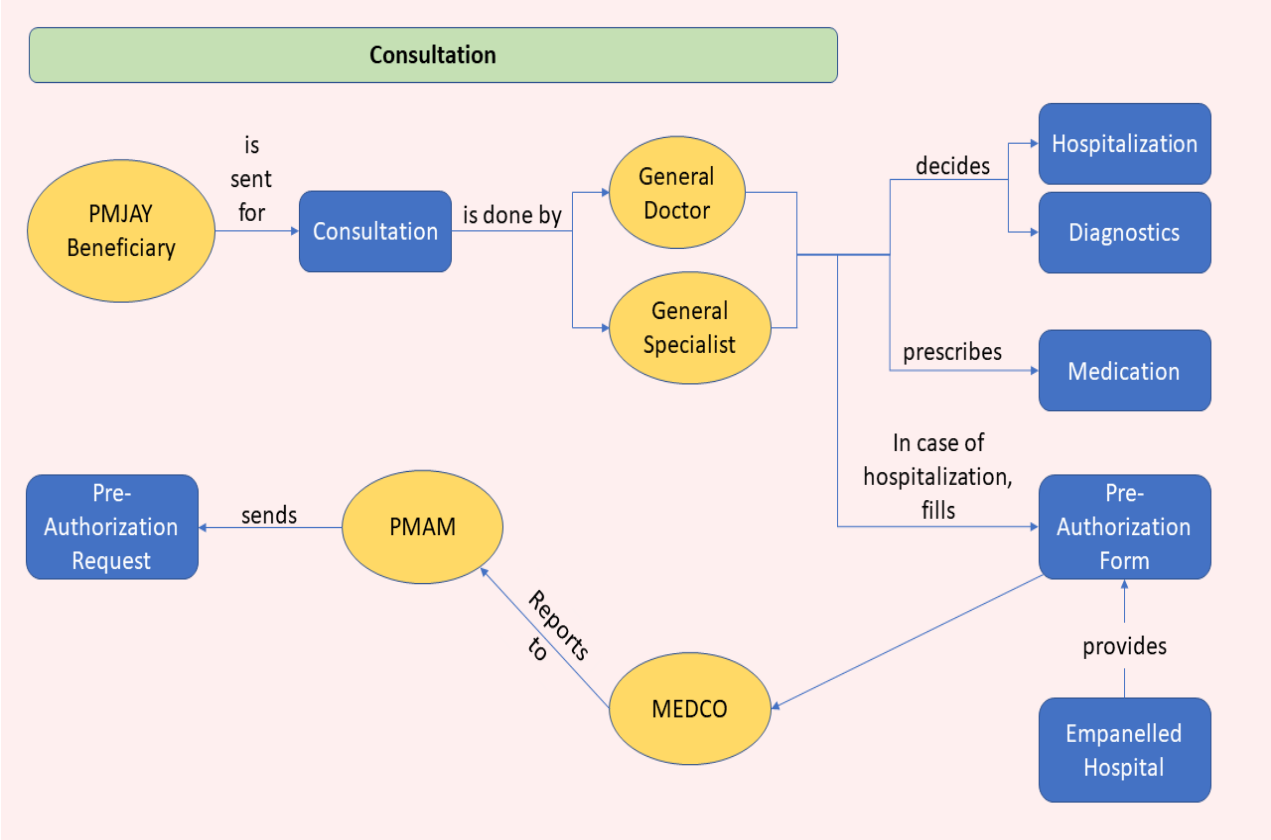


Figure 2.3: Consultation

Hospitalisation

For those beneficiaries who require hospitalisation, a pre-authorization is requested to higher authorities like State Health Agency (SHA), Implementation Support Agency

(ISA) and Insurance Company. This request is made by PMAM. If the request is rejected, the beneficiary will have to pay relevant charges, otherwise, the beneficiary is admitted to the empanelled hospital. On admission, a picture of the beneficiary is taken by PMAM and uploaded to the Transaction Management System (TMS) as depicted in 2.4

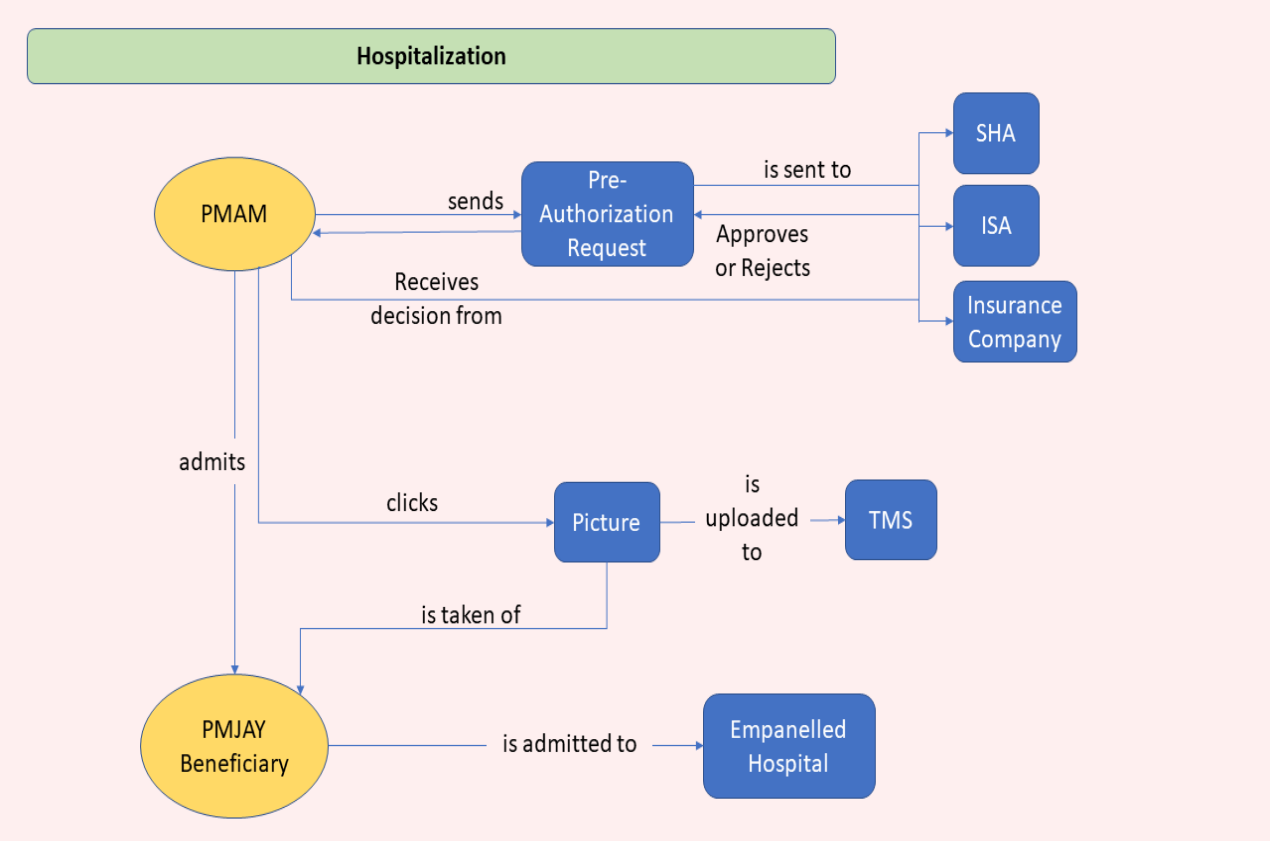


Figure 2.4: Hospitalization

Treatment

After admission, the treatment begins. Medical Packages are provided by the empanelled hospital and Treatment Procedure is provided by the PMJAY scheme which is implemented by the empanelled hospital. Treating Doctor/Specialist is appointed to the beneficiary who does the treatment as per the treatment guidelines provided by PMJAY. They also prepare the Treatment report which is forwarded by MEDCO to PMAM, who in turn uploads it to the TMS portal. A picture of the beneficiary is taken during the treatment or post-treatment and uploaded to the TMS portal by PMAM. After the treatment is complete, the beneficiary is discharged. It is depicted in 6.9

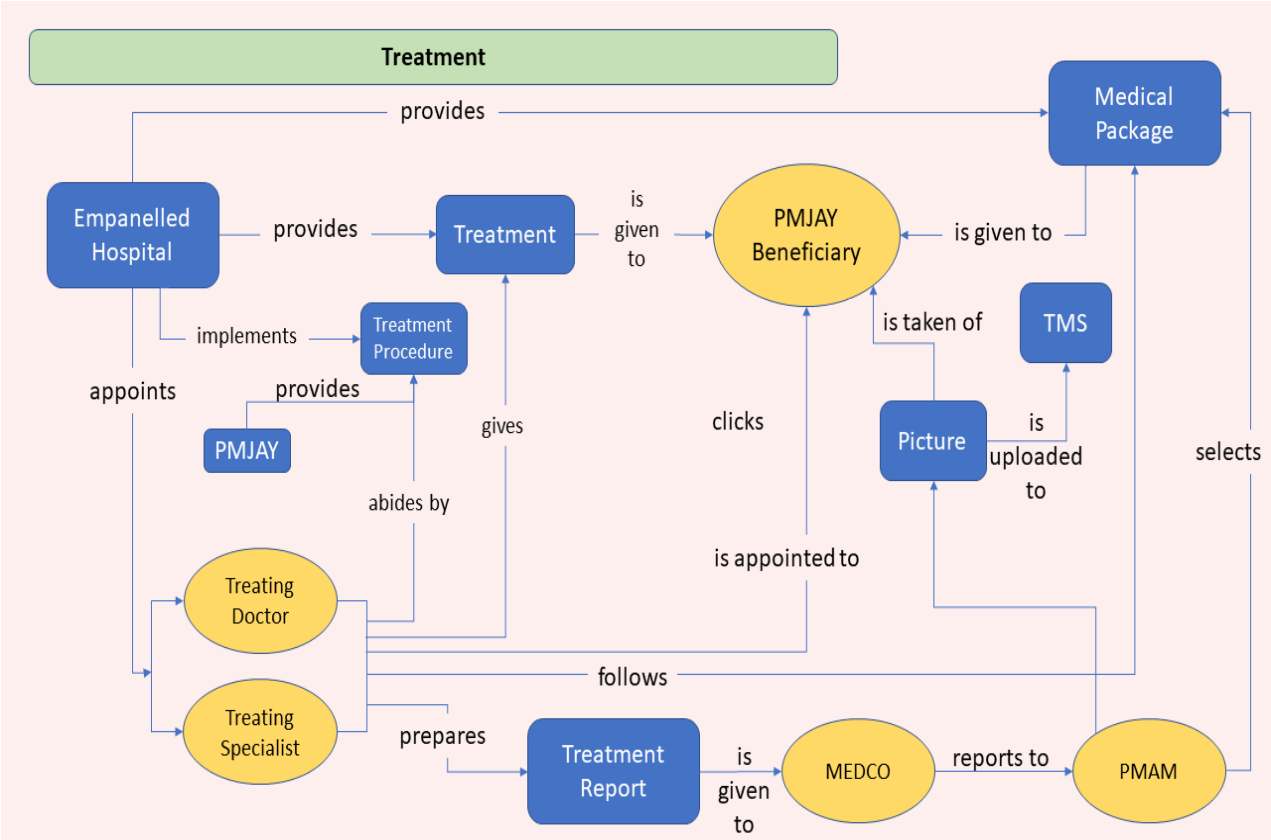


Figure 2.5: Treatment

Discharge

On discharge, all the necessary documents (medical reports, clinical notes, medical scans, etc.) are collected by PMAM along with a picture of the beneficiary at the time of discharge, which are uploaded to the TMS portal. The physical copies of the report are handed to the beneficiary. PMAM files the claim and forwards it to the Insurance Company, SHA and ISA who will settle the claims for the treatment accordingly as depicted in 2.6

Post Discharge

Post-Discharge, feedbacks are taken from the beneficiary through feedback forms which are mediated by PMAM. Call-centre also requests for feedback. These feedbacks are forwarded to the Insurance Company, SHA, NHA and ISA. Empanelled Hospital and Care Providers (EHCP) provides medications and diagnostics as per the package to the beneficiary for up to 15 days. It is depicted in 2.7

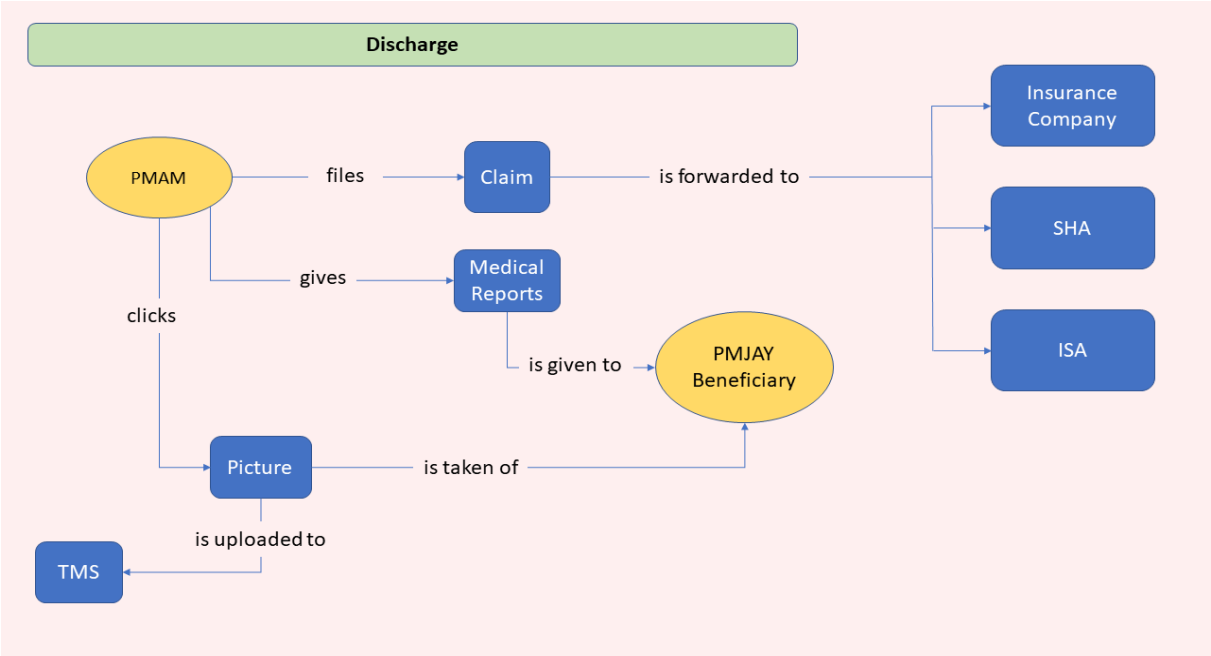


Figure 2.6: Discharge

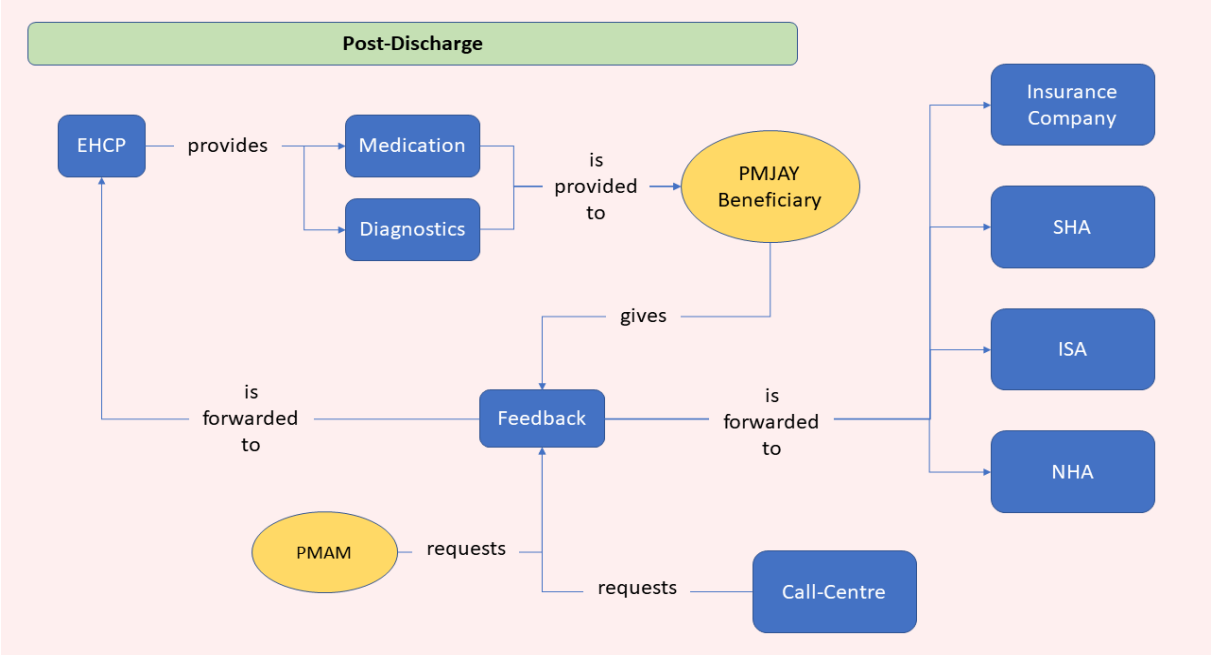


Figure 2.7: Post Discharge

2.2 Natural Language Processing - SpaCy

Natural Language Processing (NLP) is a field of artificial intelligence that focuses on the interaction between computers and human language. It involves using machine learning and computational linguistics to enable machines to understand, interpret, and generate

human language. NLP has a wide range of applications, including chatbots, sentiment analysis, text classification, and machine translation. [15]

There are several NLP tools to choose from like NLTK, SpaCy, TextBlob, Gensim, Polyglot, and CoreNLP. According to a comparative study [8], based on several criteria such as ease of installation and use, pre-processing capabilities, language support, and performance on tasks like part-of-speech tagging, named entity recognition, and sentiment analysis. The author concludes that while each library has its own strengths and weaknesses, SpaCy is the most well-rounded library with excellent performance on various NLP tasks and ease of use. So Spacy has been chosen for this work.

SpaCy is a popular Python library for NLP that provides a simple and efficient way to process natural language text. It is designed to be fast, accurate, and easy to use, and offers a wide range of features for tasks like named entity recognition (NER), part-of-speech tagging, and dependency parsing. SpaCy also includes pre-trained models for a variety of languages, which can be used out-of-the-box or fine-tuned for specific use cases. [15]

One of the key features of SpaCy is its ability to handle large amounts of text quickly and efficiently. It is built using Cython, a superset of Python that compiles to machine code, which allows it to process text much faster than pure Python. SpaCy also includes a range of optimization techniques, such as stream processing and batching, which allow it to handle large volumes of text with minimal memory usage.

Another important feature of SpaCy is its accuracy and flexibility. It includes a range of pre-trained models that have been trained on large amounts of text, which allows it to achieve high levels of accuracy on many NLP tasks. SpaCy also provides a range of customization options, such as the ability to train new models on specific domains or languages, or to add new components to the existing processing pipeline.

SpaCy's processing pipeline is modular, which allows users to customise and extend it as needed. For example, users can add custom components to perform additional processing steps, such as sentiment analysis or topic modelling, or can modify the existing components to better suit their needs. This makes SpaCy a flexible and powerful tool for

a wide range of NLP applications.

SpaCy is a powerful and flexible Python library for natural language processing that offers a wide range of features and customization options. Its speed, accuracy, and ease of use make it a popular choice for NLP tasks of all kinds, and its modular architecture allows for easy customization and extension. [16]

We have made extensive use of NLP through spaCy, for tokenization, parsing, analysing dependencies and training our own domain specific NER component creating an NLP Model for PMJAY specifically. We have trained the new NLP model on top of *en_core_web_sm* which is a pre-trained natural language processing (NLP) model developed by the spaCy library, which is an open-source library for NLP in Python.

The *en* in *en_core_web_sm* stands for English, which means that the model is specifically designed to process text in the English language. The *core* refers to the fact that the model includes the basic NLP components necessary for most NLP tasks, such as tokenization, part-of-speech tagging, and dependency parsing. The *web* indicates that the model is optimized for web data, which is typically messy and unstructured.

The *sm* stands for *small*, which means that the model is a smaller version of spaCy's English language models, with a smaller footprint and faster processing time. Despite its small size, *en_core_web_sm* is still capable of performing many common NLP tasks with high accuracy, including named entity recognition, sentence boundary detection, and lemmatization.

Overall, *en_core_web_sm* is a useful and efficient tool for processing English language text data in a variety of NLP applications, including information extraction, sentiment analysis, and text classification.

2.3 Knowledge Graph

A Knowledge Graph is a powerful technology that is used to organise and represent knowledge in a structured and machine-readable way. It is a type of database that captures and represents the relationships between different entities and concepts, such as people, places, events, and things. Knowledge Graphs are often used to support artificial

intelligence applications like chatbots, virtual assistants, and recommendation systems, as well as to power search engines like Google. [7, 22] One of the key features of a Knowledge Graph is that it allows for the representation of not only individual entities, but also the relationships between them. For example, in a Knowledge Graph of a university, a student entity might be connected to a course entity through a relationship that represents enrollment in that course. Similarly, a course entity might be connected to a professor entity through a relationship that represents the teaching of that course. These relationships can be represented in a structured and machine-readable format, allowing for more advanced querying and analysis of the data.

Another important feature of a Knowledge Graph is its ability to incorporate data from multiple sources. This means that a Knowledge Graph can be used to integrate and unify data from a wide range of different domains and sources, such as academic publications, news articles, and social media. This can be particularly useful for applications like recommendation systems, which can use the Knowledge Graph to identify related content and make personalised recommendations to users.

One of the most well-known examples of a Knowledge Graph is the Google Knowledge Graph, which powers many of the search results and recommendations displayed by Google. The Google Knowledge Graph incorporates data from a wide range of sources, including structured data markup on websites, public data sources like Wikidata, and user behaviour data from Google's own services. This allows Google to provide more informative and relevant search results to users, as well as to power features like Google Assistant and Google Maps.

Overall, Knowledge Graphs are a powerful technology that can be used to organise and represent complex knowledge in a structured and machine-readable way. They are becoming increasingly important for a wide range of applications in artificial intelligence, natural language processing, and recommendation systems, and are likely to continue to play a central role in the development of intelligent systems in the years to come. [21] Knowledge Graph has been used to show the end result of this framework by giving a visualisation of data in the form of a graph with nodes and relations incorporated with

properties pertaining to the domain, PMJAY scheme.

2.4 Regular Expressions

Regular expressions, also known as regex or regexp, are a powerful tool for matching and manipulating text patterns. They are a formal language that can be used to describe patterns in text, such as email addresses, phone numbers, URLs, and more. Regular expressions are supported in many programming languages, text editors, and other software tools, and can be used to automate tasks like data cleaning, text extraction, and search and replace. [11]

The basic syntax of a regular expression consists of a sequence of characters and meta-characters that describe a pattern to be matched in a string. For example, the regular expression $\backslash d\{3\} - \backslash d\{2\} - \backslash d\{4\}$ matches a string that consists of a three-digit number, followed by a hyphen, then a two-digit number, another hyphen, and finally a four-digit number, which is a typical format for a US Social Security Number.

Regular expressions can also be used to define character classes, which describe sets of characters that can be matched. For example, the character class $[a - z]$ matches any lowercase letter from a to z, while the character class [0-9] matches any digit from 0 to 9. Character classes can also be negated by using the $\wedge$ symbol, so $[\wedge a - z]$ matches any character that is not a lowercase letter.

In addition to basic pattern matching, regular expressions can also be used to perform more advanced operations like grouping, alternation, and quantification. Grouping allows parts of a pattern to be captured and reused in subsequent expressions, while alternation allows for matching of multiple possible patterns. Quantifiers, such as +, *, and ?, allow for specifying how many times a pattern should be matched, or whether it is optional.

While regular expressions are a powerful tool, they can also be complex and difficult to understand, particularly for beginners. They can also be computationally expensive, particularly when used with large datasets or complex patterns. However, with practice and experience, regular expressions can be an invaluable tool for working with text data

in a wide range of applications, from data cleaning and parsing to text mining and natural language processing. [11]

Regular Expressions are used in this framework to refine and cleanse input data along with elimination of variations in specific words pertaining to PMJAY scheme to remove ambiguity of the input data.

2.5 OpenAI

OpenAI is a leading artificial intelligence research organisation that is focused on creating safe and beneficial AI. The company was founded in 2015 by a group of high-profile technology leaders, including Elon Musk, Sam Altman, and Greg Brockman. Since then, OpenAI has made significant contributions to the field of AI research, including breakthroughs in natural language processing, reinforcement learning, and computer vision. [13]

One of the most significant areas of focus for OpenAI is natural language processing (NLP), which is the subfield of AI concerned with the interaction between computers and human language. OpenAI has developed several state-of-the-art NLP models, including the GPT series of models. GPT-3, the most recent version of the model, is capable of performing a wide range of language tasks, including language translation, question-answering, and language generation. In a paper published in the Journal of Open Source Software, the authors describe the architecture and capabilities of GPT-3, as well as its potential applications in the fields of language processing and artificial intelligence.

OpenAI has also made significant contributions to the field of reinforcement learning, which is a type of machine learning that involves an agent learning to interact with an environment in order to achieve a goal. OpenAI's breakthrough work in this area includes the development of algorithms like Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC), which have been shown to be highly effective in a wide range of reinforcement learning tasks. In a paper published in the journal Nature, OpenAI researchers describe the development and performance of PPO and SAC, and highlight their potential applications in fields like robotics, gaming, and autonomous vehicles.

In addition to its research contributions, OpenAI has also developed a number of tools and resources to support the development of AI. One such resource is Gym, an open-source toolkit for developing and comparing reinforcement learning algorithms. Another is Codex, a language model capable of generating code in a wide range of programming languages. In a paper published in the Journal of Machine Learning Research, the authors describe the architecture and capabilities of Codex, and demonstrate its potential applications in areas like code autocompletion and error correction.

Overall, OpenAI's contributions to the field of AI research have been significant and wide-ranging, and the organisation is likely to continue to play a leading role in the development of safe and beneficial AI in the years to come. [13] We have integrated openAI in this framework to generate domain specific unstructured data as sentences that are used for training NER component.

2.6 Neo4j

Neo4j is a graph database management system that allows users to store, manage, and query data as nodes and relationships, rather than in traditional tabular format. It is a popular choice for data-driven applications and is used in a wide range of industries, including finance, healthcare, and social media [10, 19]. Some of the key features and limitations of Neo4j are:

Features:

1. Native graph storage: One of the main features of Neo4j is that it provides native graph storage. This means that data is stored in a way that is optimised for graph-based queries, which can significantly improve performance compared to traditional relational databases.
2. Flexible data model: Neo4j provides a flexible data model that allows users to model complex relationships between entities. This makes it ideal for applications that require a high level of data complexity and inter-connectivity, such as social networks and recommendation engines.

3. High performance: Neo4j is designed to be highly performant, even with large datasets. It provides a number of features that allow users to optimise their queries, including indexing and caching.
4. Cypher query language: Neo4j provides a query language called Cypher, which is designed specifically for graph-based queries. Cypher is easy to learn and provides a powerful set of tools for querying and analysing graph data.
5. Integrations: Neo4j provides a number of integrations with popular programming languages and frameworks, including Java, Python, and Spring. This makes it easy to use Neo4j in a variety of different applications and environments.

Limitations:

1. Steep learning curve: Neo4j can have a steep learning curve, particularly for users who are not familiar with graph databases. It can take some time to get used to the graph-based data model and the Cypher query language.
2. Cost: While there is a free version of Neo4j available, the enterprise version can be expensive, particularly for larger installations. This may be a limiting factor for smaller organisations or projects with limited budgets.
3. Limited tooling: While Neo4j provides a number of integrations with popular programming languages, it can be limited in terms of tooling. This means that users may need to build their own tools for monitoring and managing their Neo4j installation.
4. No support for transactions: Neo4j does not support transactions across multiple nodes, which can be a limiting factor for some use cases. This means that users may need to implement their own mechanisms for ensuring consistency and data integrity.
5. Limited scalability: While Neo4j is designed to be highly performant, it can be limited in terms of scalability. This means that it may not be the best choice for applications with extremely large datasets.

In conclusion, Neo4j is a powerful and flexible graph database management system that provides a number of features for storing, managing, and querying graph data. It

has a steep learning curve, cost, limited tooling, lack of support for transactions, and limited scalability that may be limiting factors for some use cases. The free version has no hard limit on the number of nodes but may face practical limits that affect performance as the number of nodes grows. The performance of Neo4j depends on the dataset size, complexity of relationships, hardware resources, and type of queries being performed. The free version is suitable for smaller projects and prototypes, but for larger-scale production deployments, users may need to consider the enterprise version.

We have utilised the free version of Neo4j in this project to visualise our data in the form of a Knowledge Graph.

2.7 Google Colaboratory

Google Colab is an online platform that provides a free environment for researchers, developers, and data scientists to perform their computations using Python. It is based on the Jupyter Notebook platform and allows users to write and run code, as well as create and share documents that contain live code, equations, visualisations, and narrative text. [5]

Some of the key features of Google Colab include:

1. Free access: One of the main features of Google Colab is that it is entirely free. Users do not have to pay for any computational resources, making it an excellent option for those who want to experiment with machine learning and data science without investing in expensive hardware.
2. Pre-installed packages: Google Colab comes with a wide range of pre-installed packages, including popular libraries like NumPy, Pandas, and Matplotlib, making it easy to get started with data analysis and machine learning.
3. Collaborative platform: Google Colab is designed to be a collaborative platform, allowing multiple users to work on the same notebook simultaneously. This makes it easy to collaborate with colleagues and share your work with others.
4. GPU support: Google Colab also provides access to powerful GPUs (Graphics Processing Units) for training deep learning models, which can significantly reduce

training time. This feature is particularly useful for those working on computationally intensive projects.

5. Cloud-based storage: Google Colab provides users with access to cloud-based storage, allowing them to save their work and datasets in the cloud. This means that users can access their work from anywhere with an internet connection, and they do not have to worry about losing their work if their local machine crashes.
6. Easy integration with Google services: Google Colab is part of the Google ecosystem, and it integrates seamlessly with other Google services like Google Drive, making it easy to save and share notebooks and datasets.
7. Easy to use: Google Colab is easy to use, even for beginners. It provides an intuitive interface that allows users to create and run code, add text and visuals, and collaborate with others.

Google Colab is a great platform for users who want to experiment with machine learning and data science. However, there are some limitations for free users. Some of the limitations are:

1. Limited computational resources: Free users of Google Colab have access to limited computational resources, which means that the amount of data they can work with is also limited. This can be a significant limitation for users who are working on computationally intensive projects.
2. Limited session time: Google Colab imposes a time limit on each session, which is typically around 12 hours. If a user's session exceeds this time limit, they will have to start a new session, which can be inconvenient for long-running computations.
3. No guaranteed uptime: Google Colab is a cloud-based platform, and it relies on Google's infrastructure to provide computational resources. While Google provides a high level of uptime, there is no guarantee that the platform will be available 100
4. Limited storage space: Free users of Google Colab have limited storage space available to them, which can be a problem if they are working with large datasets. Users may need to find alternative storage solutions or pay for additional storage space.

5. Limited support: Google Colab does not provide dedicated support for free users. While there is a user community that can provide help and support, users may have to rely on their own resources to troubleshoot problems.

In conclusion, while Google Colab is an excellent platform for free users, researchers, developers, and data scientists who want to experiment with machine learning and data science, it does have some limitations. Free users are limited in terms of computational resources, session time, storage space, uptime, and support. These limitations may not be a problem for small-scale projects, but they can be significant for larger, more complex projects.

2.8 Python Programming Language

Python is a high-level, interpreted programming language that is widely used for general-purpose programming, scientific computing, data analysis, web development, artificial intelligence, and more. It was first released in 1991 by Guido van Rossum and has since become one of the most popular programming languages in the world. [6, 20]

One of the main features of Python is its simplicity and readability. Python's syntax is straightforward and easy to understand, making it an ideal language for beginners to learn. The language also has a large and active community of developers who contribute to a wide range of libraries and frameworks, making it easier to get started with complex projects.

Python is an interpreted language, which means that it does not need to be compiled before it can be executed. This makes it easy to write and test code in real-time, as well as to run programs on multiple platforms without needing to recompile the code. Python's dynamic nature also means that it is easy to add or remove functionality without changing the code.

Python is also known for its extensive library support, which includes a wide range of pre-built functions and modules for a variety of tasks. The standard library includes modules for file I/O, regular expressions, web development, networking, and more. There

are also many third-party libraries available for scientific computing, data analysis, machine learning, and other specialised areas.

Python is often used in scientific computing and data analysis due to its ability to handle large datasets and its extensive library support. Libraries such as NumPy, Pandas, and Matplotlib provide advanced tools for numerical computation, data analysis, and data visualisation. Python is also commonly used in machine learning and artificial intelligence due to its ease of use and the availability of powerful libraries like TensorFlow and PyTorch.

Python is supported by a wide range of platforms and operating systems, including Windows, macOS, and Linux. It also has a large and active community of developers, making it easy to find resources, tutorials, and support online.

Overall, Python is a versatile, powerful, and easy-to-learn programming language that is widely used across many different fields. Its simplicity, readability, and extensive library support make it an ideal language for both beginners and experienced developers alike.

Chapter 3

Framework Overview

This chapter will describe the framework the project proposes for Pradhan Mantri Jan Arogya Yojana (PMJAY) scheme particularly focusing on the process flow of the scheme from verification to post-discharge of a beneficiary.

A general understanding of this framework for PMJAY scheme is that the PMJAY related documents are given as input for this framework and a Knowledge Graph is built as the output as shown in the fig.3.1.

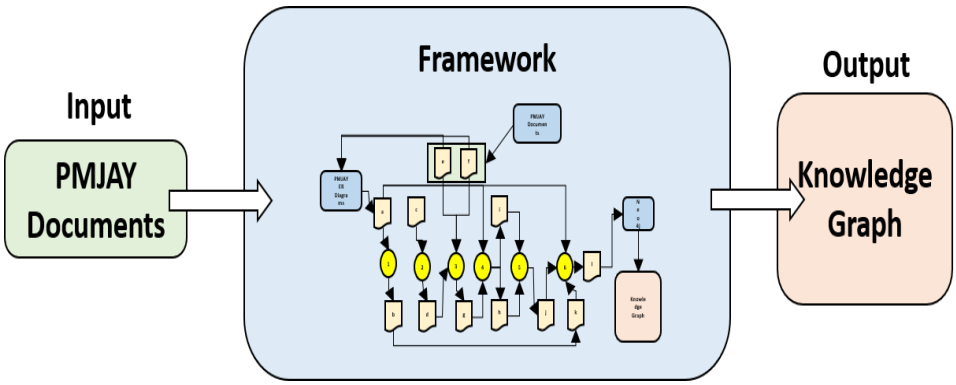


Figure 3.1: Framework Overview

Now, dwelling into the general flow of the framework in more detail, the project work flow is depicted in fig.3.2.

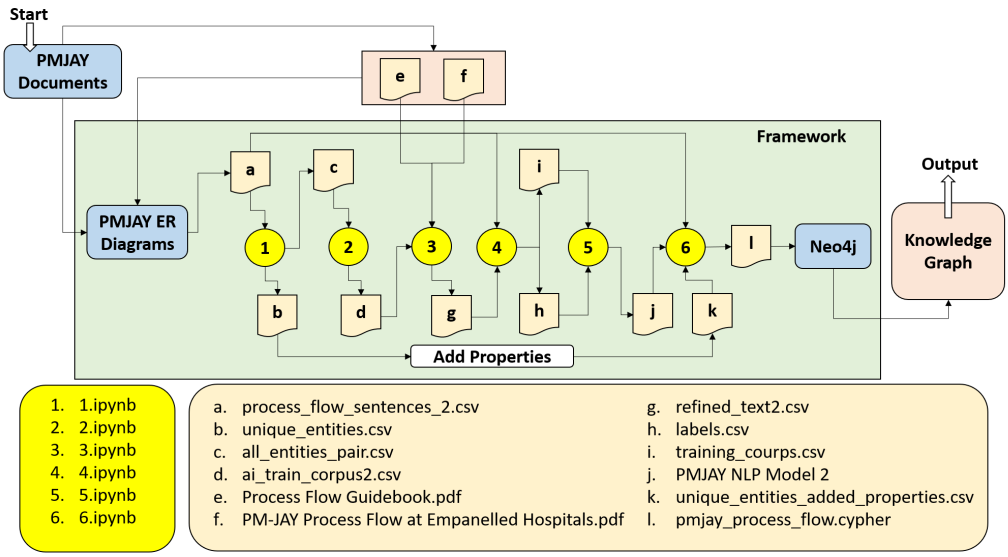


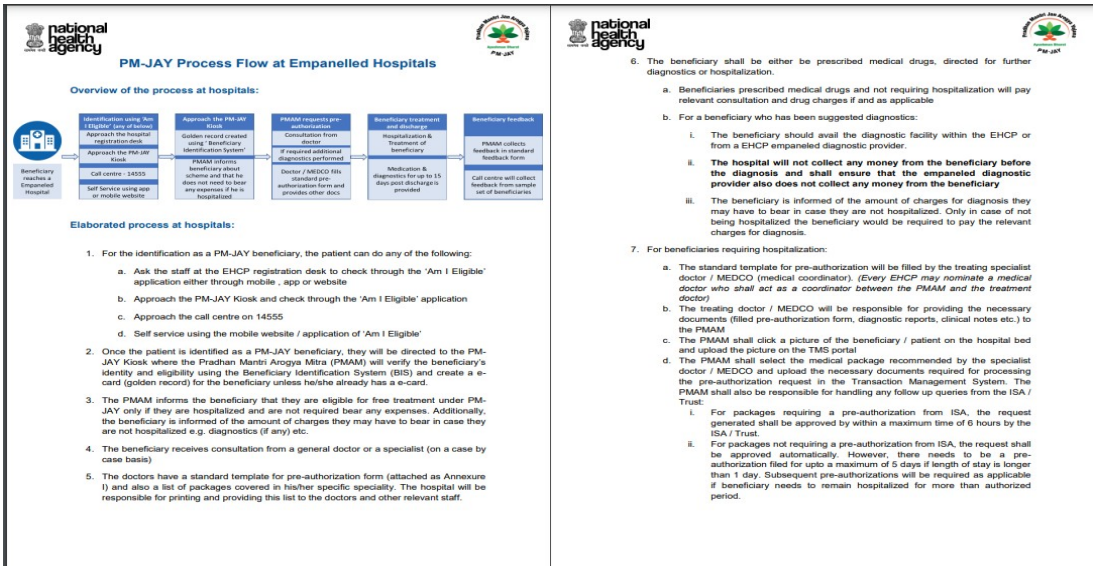
Figure 3.2: Framework

3.1 PMJAY Document Analysis

PMJAY Documents that are unstructured (e,f) [fig.3.3 and 3.4] are first analyzed to create an ER Diagram for the PMJAY process flow. Using these ER Diagrams (shown in Chapter 2: PMJAY), sentences depicting the process flow are created as process_flow_sentences_2.csv (a) [fig.3.5].

3.2 Knowing Entities

Process_flow_sentences.csv (a) [fig.3.5] containing the sentences that describes the process flow of PMJAY is given as input to 1.ipynb (1) which will generate a csv file containing a list of unique entities as unique_entities.csv (b) [fig.3.6] and another csv file that will contain all possible pairs of entities as all_entities_pair.csv (c) [fig.3.6] both of which are domain-specific as outputs.



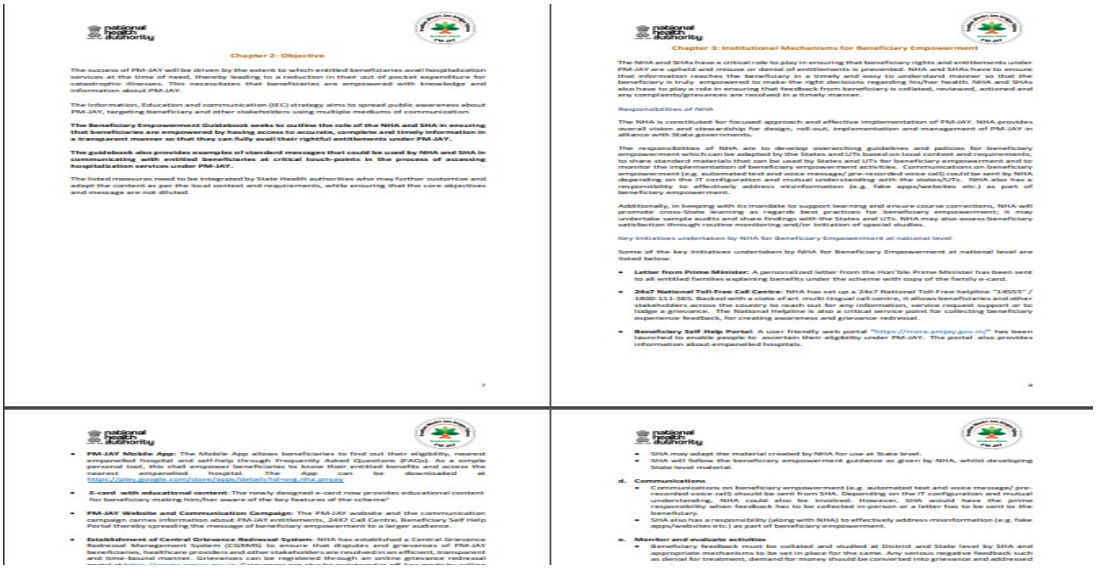
PM-JAY Process Flow at Empanelled Hospitals

Overview of the process at hospitals:

Elaborated process at hospitals:

1. For the identification as a PM-JAY beneficiary, the patient can do any of the following:
   a. Ask the staff at the EHCP registration desk to check through the 'Am I Eligible' application either through mobile , app or website
   b. Approach the PM-JAY Kiosk and check through the 'Am I Eligible' application
   c. Approach the call centre on 14555
   d. Self service using the mobile website / application of 'Am I Eligible'
2. Once the patient is identified as a PM-JAY beneficiary, they will be directed to the PM-JAY Kiosk where the Pradhan Mantri Arogya Mitra (PMAM) will verify the beneficiary's identity and eligibility using the Beneficiary Identification System (BIS) and create a e-card (golden record) for the beneficiary unless he/she already has a e-card.
3. The PMAM informs the beneficiary that they are eligible for free treatment under PM-JAY only if they are hospitalized and are not required bear any expenses. Additionally, the beneficiary is informed of the amount of charges they may have to bear in case they are not hospitalized e.g. diagnostics (if any) etc.
4. The beneficiary receives consultation from a general doctor or a specialist (on a case by case basis)
5. The doctors have a standard template for pre-authorization form (attached as Annexure I) and also a list of packages covered in his/her specific speciality. The hospital will be responsible for printing and providing this list to the doctors and other relevant staff.
6. The beneficiary shall be either be prescribed medical drugs, directed for further diagnostics or hospitalization.
   a. Beneficiaries prescribed medical drugs and not requiring hospitalization will pay relevant consultation and drug charges if and as applicable
   b. For a beneficiary who has been suggested diagnostics:
      i. The beneficiary should avail the diagnostic facility within the EHCP or from a EHCP empaneled diagnostic provider.
      ii. **The hospital will not collect any money from the beneficiary before the diagnosis and shall ensure that the empaneled diagnostic provider also does not collect any money from the beneficiary**
      iii. The beneficiary is informed of the amount of charges for diagnosis they may have to bear in case they are not hospitalized. Only in case of not being hospitalized the beneficiary would be required to pay the relevant charges for diagnosis.
7. For beneficiaries requiring hospitalization:
   a. The standard template for pre-authorization will be filled by the treating specialist doctor / MEDCO (medical coordinator). *(Every EHCP may nominate a medical doctor who shall act as a coordinator between the PMAM and the treatment doctor)*
   b. The treating doctor / MEDCO will be responsible for providing the necessary doceints (filled pre-authorization form, diagnostic reports, clinical notes etc.) to the PMAM
   c. The PMAM shall click a picture of the beneficiary / patient on the hospital bed and upload the picture on the TMS portal
   d. The PMAM shall select the medical package recommended by the specialist doctor / MEDCO and upload the necessary documents required for processing the pre-authorization request in the Transaction Management System. The PMAM shall also be responsible for handling any follow up queries from the ISA / Trust:
      i. For packages requiring a pre-authorization from ISA, the request generated shall be approved by within a maximum time of 6 hours by the ISA / Trust.
      ii. For packages not requiring a pre-authorization from ISA, the request shall be approved automatically. However, there needs to be a pre-authorization filed for upto a maximum of 5 days if length of stay is longer than 1 day. Subsequent pre-authorizations will be required as applicable if beneficiary needs to remain hospitalized for more than authorized period.

Figure 3.3: (e) PMJAY Document



Chapter 2: Objective

Chapter 3: Institutional Mechanisms for Beneficiary Empowerment

Figure 3.4: (f) PMJAY Document

index	sentences
0	The Patient checks eligibility through Am I Eligible App
1	The Patient checks eligibility through Am I Eligible Website
2	The Patient checks eligibility through Call-Centre
3	The Patient checks eligibility through PMJAY Kiosk
4	The Patient checks eligibility through EHCP Registration Desk
5	PMJAY Kiosk is assigned a PMAM
6	EHCP Registration Desk is assigned a PMAM
7	PMAM verifies identity and eligibility through BIS
8	BIS creates Silver Record
9	BIS creates Golden Record
10	BIS creates e-card
11	e-card is given to the PMJAY Beneficiary
12	PMAM informs about entitlements to the PMJAY Beneficiary
13	Patient becomes PMJAY Beneficiary
14	PMJAY Beneficiary is sent for Consultation
15	Consultation is done by General Doctor
16	Consultation is done by General Specialist
17	PMJAY Beneficiary consults a General Doctor
18	PMJAY Beneficiary consults a General Specialist

Figure 3.5: (a) process_flow_sentences_2.csv

	unique entities
0	Patient
1	PMJAY Kiosk
2	EHCP Registration Desk
3	PMAM
4	BIS
...	...
65	Medical Report
66	Feedback
67	EHCP
68	Doctor
69	Patient

70 rows × 1 columns

(a) . (b)unique_entities.csv

	ent1	ent2
0	Patient	Am I Eligible App
1	Patient	Am I Eligible Website
2	Patient	Call-Centre
3	Patient	PMJAY Kiosk
4	Patient	EHCP Registration Desk
...	...	...
97	General Specialist	MEDCO
98	Treating Specialist	MEDCO
99	Empanelled Hospital	MEDCO
100	PMJAY Beneficiary	Patient
101	Empanelled Hospital	Doctor

102 rows × 2 columns

(b) . (c)all_entities_pair.csv

Figure 3.6: (b) unique_entities.csv and (c) all_entities_pair.csv

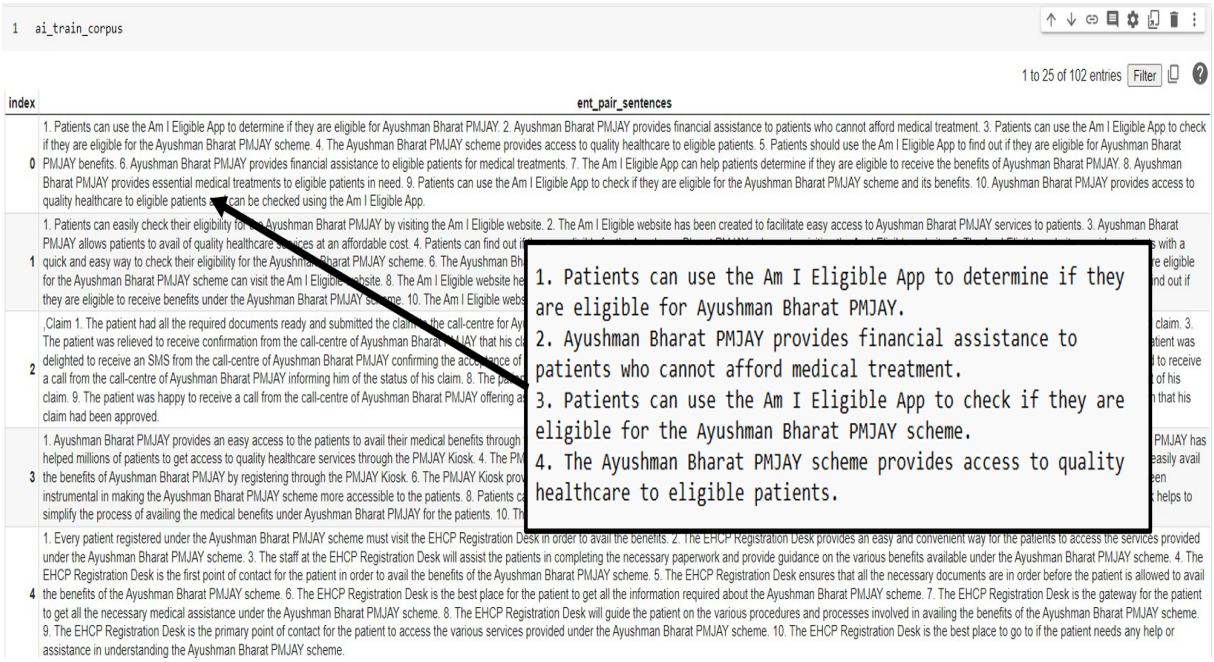


Figure 3.7: (d) ai_train_corpus.csv

3.3 Training Corpus Collection

(c) [fig.3.6] is given as input to 2.ipynb (2) which will use openAI and produce 1020 sentences with the entity pairs with ai_train_corpus.csv (d) [fig.3.7] as output. 3.ipynb (3) takes (d) [fig.3.7] as input along with (e) and (f) [fig.3.3 and 3.4], which are PMJAY Documents in the form of PDFs and produces refined_text.csv (g) [fig.3.8] as its output. (3) combines unstructured texts extracted from PDFs and ai_train_corpus.csv (d) [fig.3.7] and performs multiple levels of refinement to produce refined_text.csv (g) [fig.3.8]. (g) is given as input along with process_flow_sentences_2.csv (a) [fig.3.5] to 4.ipynb (4) which will annotate the sentences of refined_text (g) and produce labels.csv (h) [fig.3.9], containing unique labels for entities, and training_corpus.csv (i) [fig.3.10] containing annotated sentences that is used for training NER component of SpaCy.

3.4 Training NLP Models

These two outputs, labels.csv(h) [fig.3.9] and training_corpus(i) [fig.3.10], are given as input to 5.ipynb (5) which will train the NER component of SpaCy and produce PMJAY

	sentences
0	Microsoft Word - PMJAY Beneficiary Empowerment...
1	PMJAY is an ambitious government scheme which ...
2	The NHA is committed to ensuring that PMJAY Be...
3	With this spirit , NHA is sharing the PMJAY Be...
4	We sincerely hope that the SHA and other stake...
...	...
1338	PMJAY helps PMJAY Beneficiary to access qualit...
1339	PMJAY empowers PMJAY Beneficiary to make infor...
1340	PMJAY ensures that all Empanelled Hospital adh...
1341	PMJAY provides a sense of security to PMJAY Be...
1342	PMJAY helps to bridge the gap between the rich...

1343 rows × 1 columns

Figure 3.8: (g) refined_text.csv

	labels
0	DOCUMENT
1	PROCESS
2	PERSON
3	PORTAL
4	MOBILE_APP
5	CSC
6	PERMISSION
7	ORG

Figure 3.9: (h) labels.csv

	sentences	annotations
0	Microsoft Word - PMJAY Beneficiary Empowerment...	{'entities': [(17, 34, 'PERSON'), (67, 72, 'OR...
1	PMJAY is an ambitious government scheme which ...	{'entities': [(0, 5, 'ORG'), (153, 172, 'ORG')...
2	The NHA is committed to ensuring that PMJAY Be...	{'entities': [(4, 7, 'ORG'), (38, 55, 'PERSON')]}
3	With this spirit , NHA is sharing the PMJAY Be...	{'entities': [(19, 22, 'ORG'), (38, 55, 'PERSO...
4	We sincerely hope that the SHA and other stake...	{'entities': [(27, 30, 'ORG'), (71, 76, 'ORG')...
...	...	...
1285	PMJAY helps PMJAY Beneficiary to access qualit...	{'entities': [(0, 5, 'ORG'), (12, 29, 'PERSON'...
1286	PMJAY empowers PMJAY Beneficiary to make infor...	{'entities': [(0, 5, 'ORG'), (15, 32, 'PERSON'...
1287	PMJAY ensures that all Empanelled Hospital adh...	{'entities': [(0, 5, 'ORG'), (23, 42, 'ORG'), ...
1288	PMJAY provides a sense of security to PMJAY Be...	{'entities': [(0, 5, 'ORG'), (38, 55, 'PERSON'...
1289	PMJAY helps to bridge the gap between the rich...	{'entities': [(0, 5, 'ORG'), (100, 117, 'PERSO...

1290 rows × 2 columns

Figure 3.10: (i) training_corpus.csv

NLP Model (j) as output [fig.3.11].

3.5 Entity Relation Extraction and Cypher Query Generation

This PMJAY NLP Model (j) [fig.3.11] is given as input to 6.ipynb (6) along with previously created process_flow_sentences.csv (a) and unique_entities_added_properties.csv (k) [fig.3.5 and 3.12] which is created using unique_entities.csv (b) [fig.**??**] and adding properties to each entity by analysing the PMJAY Documents.

Entities and Relations are extracted in 6.ipynb (6) and cypher script, pmjay_process_flow.cypher (l) [fig.3.13], is generated to represent the extracted entities and relations with properties in the form of Knowledge Graph.

3.6 Visualization

The generated pmjay_process_flow.cypher (l) [fig.3.13] is imported into Neo4j for visualisation of the Knowledge Graph representing the process flow of PMJAY which can be

```
import spacy
from spacy import displacy
from IPython.core.display import display, HTML

nlp= spacy.load('/content/drive/MyDrive/Final (Completed)/nlp_model_2')
t="The Patient checks eligibility throung PMJAY Kiosk or EHCP Registration Desk at any Empanelle
The BIS verifies the eligibility of the Patient and decides if he or she is a PMJAY Beneficiary.
The Doctor decides if the Patient needs Hospitalization, Medication or further Diagnostics."
doc=nlp(t)
html = displacy.render(doc, style="ent")
display(HTML(html))

```

The Patient PERSON checks eligibility throung PMJAY Kiosk CSC or EHCP Registration Desk CSC at any Empanelled Hospital ORG . The BIS PORTAL verifies the eligibility of the Patient PERSON and decides if he or she is a PMJAY Beneficiary PERSON . The Doctor PERSON decides if the Patient PERSON needs Hospitalization,

Figure 3.11: (j)PMJAY NLP Model

0	Treatment Procedure	name:"Treatment Procedure"	description: "The standart treament guidelines...	url: "https://www.pmjay.gov.in/resources/docum...
1	Insurance Company	name:"Insurance Company"	description:"The colaborated Insurance Company...	NaN
2	Medication	name:"Medication"	description: "The medications are provided to ...	NaN
3	Am I Eligible App	name:"Am I Eligible App"	description:"The Am I Eligible App is a moblie...	NaN
4	TMS	name:"TMS"	full_form:"Transaction Management System"	description:"It is an online platform develope...
5	Picture	name:"Picture"	description:"The Picture of the PMJAY Benefici...	NaN
6	Claim	name:"Claim"	description:"When the beneficiary seeks treatm...	NaN
7	EHCP	name:"EHCP"	full_form:"Empanelled Health Care Provider"	description:"Empanelled Health Care Providers ...

Figure 3.12: (k) unique_entities_added_properties.csv

```
CREATE (ab:DOCUMENT{name:"Silver Record", description:"A beneficiary is considered a Silver Record if their name, gender, age
CREATE (ac:ORG{name:"Empanelled Hospital", description:"Empanelled hospitals are healthcare facilities that have been enliste
CREATE (ad:ORG{name:"Call-Centre", description:"A call centre to provide information and assistance to beneficiaries regardin
CREATE (ae:ORG{name:"PMJAY", full_form:"Pradhan Mantri Jan Arogya Yojana", description:"PMJAY stands for Pradhan Mantri Jan A
CREATE (af:PROCESS{name:"Hospitalization", description:"The medical treatment that requires an individual to stay in a hospit
CREATE (ag:PERSON{name:"General Specialist", description:"A Specialist appointed to a PMJAY Beneficiary for initials consulta
CREATE (ah:DOCUMENT{name:"e-card", description:"The e-card serves as a proof of identity and eligibility for availing cashles
CREATE (ai:CSC{name:"EHCP Registration Desk", description:"The EHCP registration desk is an online portal where a person can
CREATE (aj:PERSON{name:"MEDCO", full_form:"Medical Coordinator", description:"A medical coordinator is a healthcare professio
CREATE (ak:DOCUMENT{name:"Medical Report", description:"A medical report is a document that contains details of the medical c
CREATE (al:CSC{name:"PMJAY Kiosk", description:"A physical facility that is set up to provide information and assistance to b
CREATE (am:PERSON{name:"PMAM", full_form:"Pradhan Mantri Aarogya Mitra", description:"They are trained professionals who are
CREATE (an:PERSON{name:"General Doctor", description:"A doctor appointed to a PMJAY Beneficiary for intital consultation."})
CREATE (z)-[:checks_eligibility_through]->(ad)
CREATE (z)-[:checks_eligibility_through]->(al)
CREATE (z)-[:checks_eligibility_through]->(ai)
CREATE (al)-[:is_assigned_a]->(am)
CREATE (ai)-[:is_assigned_a]->(am)
CREATE (am)-[:verifies_identity_and_eligibility_through]->(w)
CREATE (w)-[:creates]->(ab)
```

Figure 3.13: (l)pmjay_process_flow.cypher

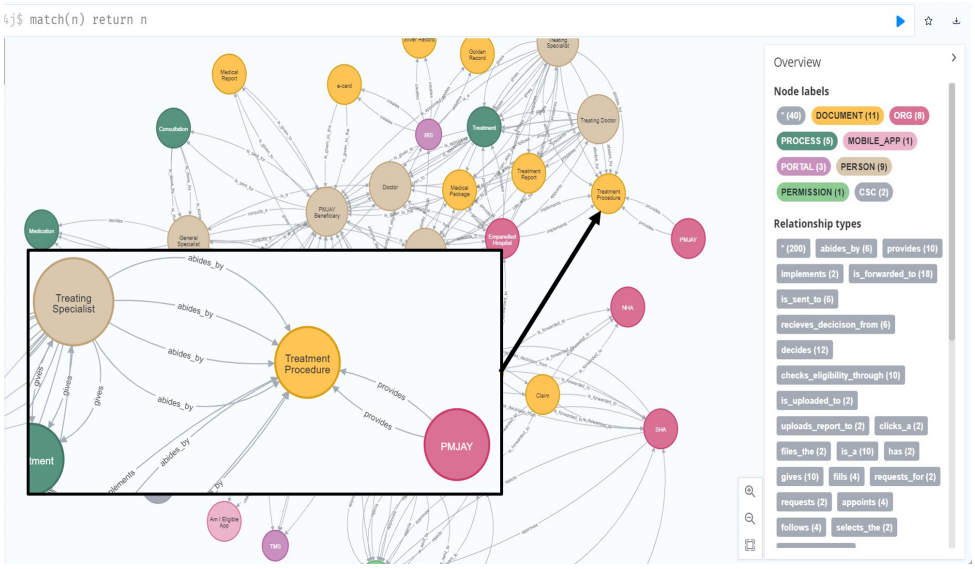


Figure 3.14: (l)Neo4j: PMJAY Process Flow Knowledge Graph

queried as per stakeholders interest. The Knowledge Graph is depicted in fig.3.14

In the next chapter 4, we will see the implementation and results in more detail followed by Performance Evaluation and Use Cases in Chapter 5 and 6 respectively.

Chapter 4

Implementation and Results

In this chapter, the detailed workflow of the project will be described with explanation of each and every input and output files, related code and libraries used therein with snapshots of the code and outputs.

From the previous chapter, the objectives of the project can be broken down listed as such:

1. Analysis of PMJAY Documents
2. Data Refining
3. Training Corpus Creation
4. Training of NER Component
5. Generation of Cypher Script
6. Visualisation and Query in Neo4j

4.1 Analysis of PMJAY Documents

Domain-specific Named Entity Recognizition (NER) demands a thorough knowledge of the domain so that entities and labels which are necessary to train the model can be understood. PMJAY websites and related PDFs have been analysed to recognize those entities and labels. Specifically the process flow of PMJAY scheme has been studied and ER Diagrams depicting the PMJAY Process Flow has been shown in Chapter 2.

sent_sl_no	sentences	ent_1	lab_1	ent_2	lab_2
1	The Patient checks eligibility through Am I Eligible App	Patient	PERSON	Am I Eligible Ap	MOBILE_APP
2	The Patient checks eligibility through Am I Eligible Website	Patient	PERSON	Am I Eligible We	PORTAL
3	The Patient checks eligibility through Call-Centre	Patient	PERSON	Call-Centre	ORG
4	The Patient checks eligibility through PMJAY Kiosk	Patient	PERSON	PMJAY Kiosk	CSC
5	The Patient checks eligibility through EHCP Registration Desk	Patient	PERSON	EHCP Registrati	CSC
6	PMJAY Kiosk is assigned a PMAM	PMJAY Kiosk	CSC	PMAM	PERSON
7	EHCP Registration Desk is assigned a PMAM	EHCP Registrati	CSC	PMAM	PERSON
8	PMAM verifies identity and eligibility through BIS	PMAM	PERSON	BIS	PORTAL
9	BIS creates Silver Record	BIS	PORTAL	Silver Record	DOCUMENT
10	BIS creates Golden Record	BIS	PORTAL	Golden Record	DOCUMENT

Figure 4.1: Process Flow Sentences

PMJAY Process Flow - Entities and Labels

Entity	Label	Entity	Label	Entity	Label
Golden Record	DOCUMENT	Medical Report	DOCUMENT	Consultation	PROCESS
Treatment Procedure	DOCUMENT	Treatment Report	DOCUMENT	Doctor	PERSON
Pre-Authorization Request	PERMISSION	PMJAY Kiosk	CSC	Treating Specialist	PERSON
PMJAY	ORG	ISA	ORG	Empanelled Hospital	ORG
MEDCO	PERSON	BIS	PORTAL	Hospitalization	PROCESS
EHCP Registration Desk	CSC	TMS	PORTAL	Diagnostics	PROCESS
Treatment	PROCESS	Call-Centre	ORG	Picture	DOCUMENT
EHCP	ORG	General Doctor	PERSON	NHA	ORG
PMAM	PERSON	Treating Doctor	PERSON	Silver Record	DOCUMENT
SHA	ORG	e-card	DOCUMENT	Am I Eligible App	MOBILE_APP
Insurance Company	ORG	Medical Package	DOCUMENT	PMJAY Beneficiary	PERSON
Patient	PERSON	Pre-Authorization Form	DOCUMENT	Am I Eligible Website	PORTAL
General Specialist	PERSON	Claim	DOCUMENT		
Feedback	DOCUMENT	Medication	PROCESS		

Figure 4.2: Entities and Labels

From these ER diagrams, process flow sentences along with entities and labels have been recorded/written in process_flow_sentences_2.csv file, a sample of which is fig.4.1.

After analysis of the PMJAY Documents, a total of 40 Entities and 8 Labels have been recognized. Those entities and labels are depicted in fig.4.14:-

The descriptions of the 8 labels are as follows:-

1. CSC: Common Service Centers like PMJAY Kiosk and EHCP Registration Desk.
2. PROCESS: Activities/events that involve a process such as hospitalisation, medication, treatment, diagnostics etc.
3. PERSON: Name or Designated domain-specific name of a person such as PMJAY Beneficiary, Doctor, PMAM etc.
4. MOBILE_APP: A mobile application that is related to PMJAY Scheme.
5. ORG: Organisation, Agency, Institution etc.
6. PERMISSION: Requests made by a person or organisation
7. PORTAL: Websites related to PMJAY scheme.

8. DOCUMENT: Documents like medical reports, forms, claims etc.

After the analysis of the PMJAY documents, the next step is extraction of text from these documents and refinement as per entities recognized.

4.2 Data Refining

Text has been extracted from the unstructured PDF documents on PMJAY using Apache Tika library which is a powerful and flexible tool for working with unstructured data used to parse and extract text from a wide range of file formats without having to implement custom parsers or extractors themselves. The code snippet is shown below, where raw text has been extracted from two PDFs using the tika library. Please refer to 3.ipynb for full code of Data Refinement.

Now we have the raw_text as intermediary output. We also have AI generated sentences created from analysed entities. These AI generated sentences are acquired by using openai library which is a Python library that provides a simple interface for accessing OpenAI's state-of-the-art artificial intelligence models. These models can be used for a wide range of tasks, such as natural language processing, machine translation, and image recognition. We have used it to generate PMJAY related sentences. Please refer to "1.ipynb" for full code and explanation. The above raw_text and ai_generated_sentence [fig.3.7] are combined into one and will go through 3 levels of refinement.

Level 1 Refinement: The first level refinement will remove "noise" from the raw text that is not relevant to the analysis or processing of the text, specifically, the extra spaces, newlines and special characters that are not required. RegEx Patterns are used for this objective as described in fig.4.3. This level 1 refine done using the "refine_1(text)" function shown in fig.4.4.

Level 2 Refinement: The second level refinement will sort-out all possible variations of entities. For example, the possible variations of the entity PMJAY are PM-JAY, pm-jay, Pm-Jay, Pradhan Mantri Jan Arogya Yojana and so on. In this level of refinement, these variations are replaced with the entity i.e., PMJAY. Regular Expressions are

RegEx Patterns:
Substitute these patterns with required character.
A. For Ordering Alphabets and Numbering Numerals and Roman: Substitute with '<empty string>'
`pattern: ((\b(\d|\d\d)\.)|(\b(i|v|x)+\.)|(\b(I|V|X)+\.)|(\b([a-z]|[A-Z])\.($|\ )+))`

1. `(\b(\d|\d\d)\.)` :- For Numbers like '1.','2.',...,'9.','10.','11.',...,'99.'
2. `((\b(i|v|x)+\.)|(\b(I|V|X)+\.))` :- For Roman Numbers like 'i.','ii.',...,'I.','II','III',...
3. `(r\b([a-z]|[A-Z])\.($|\ )+)` :- For Ordering Alphabets like 'a.','b.',...'z.','A.','B.',...'Z.'

B. For extra spaces, characters and newlines: Substitute with single space.
`pattern:((\n|\s|!|#|%|\^|&|\*|\(|\))+)`

1. `(\s+)` :-For one or more spaces: Substitute with one space.
2. `(\n+)` :-For one or more new lines: Substitute with one space
3. `(!|#|%|\^|&|\*|\(|\))+` :-For unnecessary characters '!','$','%','^','&','*','(' and ')'.

C. For multiple subsequent dots "....": Substitute with single dot "."

1. `(\.+)` :- For one or more dot/s "...": substitute with single dot ".".

Figure 4.3: Regex Pattern 1

```
[ ]  1  #Regular Expression Library
     2  import re
```

```
[ ]  1  #Level 1 Refinement
     2  def refine_1(text):
     3      #Ordering Alpha
     4      a= re.compile(r
     5      b= re.compile(r
     6      c= re.compile(r'(\.+) ') #Multiple dots (...)
     7      d= re.compile(r'(\d*\ Section\ \d\ *:\ *)') #Section
     8      e= re.compile(r'(\d*\ Chapter\ \d\ *:\ *)') #Chapter
     9      f= re.compile(r'(\d*\ Annexe\ \d\ *:\ *)') #Annexe
    10      g= re.compile(r'(^\ *\d*\ )') #Starting Numbers in a Sentence
    11      h= re.compile(r'•') # Marking Dot (•)
    12      i= re.compile(r'(^[aA-zZ]*\.*$)') #Single-Word Sentences
    13      text= i.sub('',(h.sub('',(g.sub('',(f.sub('',(e.sub('',(d.sub('',(c.sub('.',(b.sub(' ',(a.sub('',text))))))))))))))))
    14      return(text)
```

```
g= re.compile(r'(^\ *\d*\ )') #Starting Numbers in a Sentence
h= re.compile(r'•') # Marking Dot (•)
i= re.compile(r'(^[aA-zZ]*\.*$)') #Single-Word Sentences
```

Figure 4.4: Level 1 Refinement

used to carry out this task. A set of PMJAY specific regular expressions has been made for this purpose, a sample of which are shown in fig.4.5. This level 2 refinement is done using the "refine_2(text)" function shown in fig.4.6.

```
1  #Regex Patterns for possible variations of unique entities
2  regex_pattern=[]
3  #am i eligible, aam - i-eligible
4  regex_pattern.append(('Am I Eligible App',(re.compile(r'(\'*[aA]+?m[\ -]*?[iI]+?[\ -]*?[eE]+?ligible[\ -]*?[aA]+?pp(lication)*s*([\ \']*?)*)'))))
5  regex_pattern.append(('Am I Eligible Website',(re.compile(r'(\'*[aA]+?m[\ -]*?[iI]+?[\ -]*?[eE]+?ligible[\ -]*?[wW]+?ebsites*([\ \']*?)*)'))))
6  regex_pattern.append(('BIS',(re.compile(r'((\bbis\b)|([bB]+?eneficiary\ *[iI]+?dentification\ *[sS]+?ystems*))'))))
7  regex_pattern.append(('Call-Centre',(re.compile(r'[cC]+?all[\ -]*?[cC]+?ent[re]?[er]?s*'))))
8  regex_pattern.append(('Claim',(re.compile(r'\b[cC]+?laims*'))))
9  regex_pattern.append(('Consultation',(re.compile(r'\b[cC]+?onsultations*'))))
10 regex_pattern.append(('Diagnostics',(re.compile(r'[dD]+?iagnostics*'))))
11 regex_pattern.append(('Doctor',(re.compile(r'\b[dD]+?octors*'))))
12 regex_pattern.append(('e-card',(re.compile(r'[eE]+?[-\ ]*?[cC]+?ards*'))))
13 regex_pattern.append(('EHCP',(re.compile(r'(\b[eE]+?[hH]+?[cC]+?[pP]+?\b)|[eE]+?mpanelled\ *[hH]+?ealth\ *[cC]+?are\ *[pP]+?roviders*'))))
14 regex_pattern.append(('EHCP Registration Desk',(re.compile(r'([eE]+?[hH]+?[cC]+?[pP]+?)+[-\ ]*?[rR]+?egistration[-\ ]*?([dD]+?esks*)+'))))
15 regex_pattern.append(('Empanelled Hospital',(re.compile(r'([eE]+?[hH]+?[cC]+?[pP]+?\ )*([eE]+?mpanelled\ *)*[hH]+?ospitals*'))))
16 regex_pattern.append(('Feedback',(re.compile(r'[fF]+?eedbacks*'))))
17 regex_pattern.append(('General Doctor',(re.compile(r'[gG]+?eneral\ *[dD]+?octors*'))))
18 regex_pattern.append(('General Doctor or General Specialist',(re.compile(r'[gG]+?eneral\ *[dD]+?octors*((\ *|/)|(/|\ *))+([gG]+?eneral\ *)*[sS]+?pecialists*'))))
19 regex_pattern.append(('General Specialist',(re.compile(r'[gG]+?eneral\ *[sS]+?pecialists*'))))
20 regex_pattern.append(('Golden Record',(re.compile(r'[gG]+?olden[\ -]*?[rR]+?ecords*'))))
21 regex_pattern.append(('Hospitalization',(re.compile(r'[hH]+?ispitali[zs]?ations*'))))
22 regex_pattern.append(('Insurance Company',(re.compile(r'(([iI]+?nsurers*)|([iI]+?nsurance\ *([cC]+?ompan(y|ies))*))'))))
23 regex_pattern.append(('ISA',(re.compile(r'(\bisa\b)|([iI]+?mplementations*\ *[sS]+?upport\ *[aA]+?genc(y|ies))'))))
24 regex_pattern.append(('MEDCO',(re.compile(r'([mM]+?edco)|([mM]+?edical\ *[cC]+?o(-)*ordinators*)'))))
25 regex_pattern.append(('Medical Package',(re.compile(r'(([mM]+?edical\ *)*([pP]+?ackage*))'))))
26 regex_pattern.append(('Medical Report',(re.compile(r'(([mM]+?edical\ *)*([dD]+?iagnostics*\ *)*[rR]+eports*)|([cC]+?linical\ *[nN]+?otes*)|([dD]+?ischarge\ *[sS]+?ummar(y|ies))'))))
27 regex_pattern.append(('NHA',(re.compile(r'(\bnha\b)|([nN]+?ational\ *[hH]+?ealth\ *[aA]+?uthorit(y|ies))'))))
```

```
regex_pattern.append(('Claim',(re.compile(r'\b[cC]+?laims*'))))
regex_pattern.append(('Consultation',(re.compile(r'\b[cC]+?onsultations*'))))
regex_pattern.append(('Diagnostics',(re.compile(r'[dD]+?iagnostics*'))))
regex_pattern.append(('Doctor',(re.compile(r'\b[dD]+?octors*'))))
regex_pattern.append(('e-card',(re.compile(r'[eE]+?[-\ ]*?[cC]+?ards*'))))
```

Figure 4.5: Regex Pattern 2

Level 3 Refinement: Once level 2 refinement is done on combined raw_text, the

```python
def refine_2(sent):
    for p in regex_pattern:
        sent=p[1].sub(p[0],sent)
    return(sent)
```

(a) Level 2 Refinement

```python
def refine_3(sent):
    p= re.compile(r'\b(\w+)\s+\1\b')
    sent= p.sub(r'\1',sent)
    return(sent)
```

(b) Level 3 Refinement

Figure 4.6: Level 2 and 3 Refinements

	sentences
0	Microsoft Word - Beneficiary Empowerment Guide...
1	PM - JAY is an ambitious government scheme whi...
2	The National Health Authority NHA is committed...
3	With this spirit , NHA is sharing the Benefici...
4	We sincerely hope that the SHAs and other stak...
...	...
1338	Ayushman Bharat PMJAY helps beneficiaries to a...
1339	Ayushman Bharat PMJAY empowers beneficiaries t...
1340	Ayushman Bharat PMJAY ensures that all empanel...
1341	Ayushman Bharat PMJAY provides a sense of secu...
1342	Ayushman Bharat PMJAY helps to bridge the gap ...

1343 rows × 1 columns

(a) Unrefined

	sentences
0	Microsoft Word - PMJAY Beneficiary Empowerment...
1	PMJAY is an ambitious government scheme which ...
2	The NHA is committed to ensuring that PMJAY Be...
3	With this spirit , NHA is sharing the PMJAY Be...
4	We sincerely hope that the SHA and other stake...
...	...
1338	PMJAY helps PMJAY Beneficiary to access qualit...
1339	PMJAY empowers PMJAY Beneficiary to make infor...
1340	PMJAY ensures that all Empanelled Hospital adh...
1341	PMJAY provides a sense of security to PMJAY Be...
1342	PMJAY helps to bridge the gap between the rich...

1343 rows × 1 columns

(b) Final Refined

Figure 4.7: Unrefined and Final Refined Text

third level of refinement is performed in which the adjacent duplicates formed dues to level 2 refinement are removed. For example, “PM-JAY (Pradhan Mantri Jan Arogya Yojana)” after level-2 refinement becomes “PMJAY (PMJAY)”, thus performing level 3 refinement will give us “PMJAY”. This task is accomplished using the “refine_3(text)” function shown in fig.4.6.

Once all the raw text goes through these 3 levels of refinement, we get a fully refined text which is saved as “refined_text.csv”. the output is shown in fig.4.7. Next we create the training corpus to train the NER component of SpaCy for PMJAY.

4.3 Training Corpus Creation

Now the refined_text is ready, we use it to create a training corpus that will be used to train the NER component of spacy. The training corpus will have annotations of entities with their relevant labels. The format of a training set is as follows:

```
t_data=[(sentence,{'entities':[(start_ind,end_ind,label),...]}),...]
```

where "t_data" is a list of tuples where each tuple contains a sentence and a dictionary. This dictionary has a key:value pair, where key is 'entities' and value is a list of tuples, each containing the start index, the end index and the label for the entity.

The annotation is done by finding start and end indices for entities in a sentence and labelling them accordingly. In the code, "4.ipynb", which automates this process, refined_text2.csv [fig/3.8] is given as input which needs to be annotated. Also, process_flow_sentences_2.csv [fig.3.5] is imported to get the list of entities and labels. The snippet in fig. 4.8 shows code for annotating the entities where "t" will contain the list of annotated sentences, a sample of which would look like

```
('PMJAY has made healthcare more accessible for people by providing
 them with free Empanelled Hospitalization through a network of
 General Doctor.', [(0, 5, 'ORG'), (81, 100, 'ORG'),
 (92, 107, 'PROCESS'), (129, 143, 'PERSON'), (137, 143, 'PERSON')])
```

```
#Find entities in the sentences and annotate them with relevant labels
print("Finding entities in the sentences and annotating them with relevant labels")
import re
t=[] #List of Annotated sentences
for s in corpus['sentences']:
    index=[] #Stores start and end indices of entities in a sentence along with their labels
    for e in entities:
        #get start and end index of tok.text
        for i in re.finditer(e, s): #Finds indices of entity e in sentence s
            #index.append((e,i.start(),i.end(),el_dict[e])) #For Analysis Purpose Only
            index.append((i.start(),i.end(),el_dict[e]))
    index=sorted(index)
    t.append((s,index))
```

Figure 4.8: Annotation Code

However, the annotated sentences will have overlapping annotations for entities which causes a problem while training the NER component, these overlapping entities are filtered by first detecting them and then keeping only the annotations which have a larger span of entity length. For example, in "EHCP Registration Desk", "EHCP" is an entity and "EHCP Registration Desk" is also an entity, the annotations will be (0,4, ORG) and (0,22, CSC) so the annotation with lesser length will be removed.

Once the overlapping entities are removed, we now can have a perfect training set containing 1290 annotated sentences as training_corpus.csv, a sample of which is shown in fig.4.9

	sentences	annotations
0	Microsoft Word - PMJAY Beneficiary Empowerment...	{'entities': [(17, 34, 'PERSON'), (67, 72, 'OR...
1	PMJAY is an ambitious government scheme which ...	{'entities': [(0, 5, 'ORG'), (153, 172, 'ORG')...
2	The NHA is committed to ensuring that PMJAY Be...	{'entities': [(4, 7, 'ORG'), (38, 55, 'PERSON')]}
3	With this spirit , NHA is sharing the PMJAY Be...	{'entities': [(19, 22, 'ORG'), (38, 55, 'PERSO...
4	We sincerely hope that the SHA and other stake...	{'entities': [(27, 30, 'ORG'), (71, 76, 'ORG')...
...	...	...
1285	PMJAY helps PMJAY Beneficiary to access qualit...	{'entities': [(0, 5, 'ORG'), (12, 29, 'PERSON'...
1286	PMJAY empowers PMJAY Beneficiary to make infor...	{'entities': [(0, 5, 'ORG'), (15, 32, 'PERSON'...
1287	PMJAY ensures that all Empanelled Hospital adh...	{'entities': [(0, 5, 'ORG'), (23, 42, 'ORG'), ...
1288	PMJAY provides a sense of security to PMJAY Be...	{'entities': [(0, 5, 'ORG'), (38, 55, 'PERSON'...
1289	PMJAY helps to bridge the gap between the rich...	{'entities': [(0, 5, 'ORG'), (100, 117, 'PERSO...

1290 rows × 2 columns

Figure 4.9: Training Corpus for domain specific NLP Model

In the next section, we train the NER component using the training_corpus.

4.4 Training of NLP Model

Named Entity Recognition (NER) is a component of Natural Language Processing (NLP) that helps in identifying entities in a given text. This project makes use of spaCy which is a library for NLP. We will use the “en_core_web_sm” english model of spaCy.

For a domain-specific model, we will train PMJAY specific NER component which will be used to create cypher script to build a knowledge graph. spaCy has various built-in pipes viz., tokenizer, parser, tagger, ner and so on. We focus on ner pipe, train it as per the domain PMJAY, and save it as a NLP model for use.

To train and build a domain specific NER component, we use training_coprus.csv generated in the previous section. We will train NLP model in two ways i.e., from a blank model and from a pre-existing trained model. During the performance evaluation we will

select the one which give a better performance. Please refer to the full code “5.ipynb”.

The steps involved in training a domain specific NER are as follows:

1. Load spacy english models, here “en_core_web_sm” (pre-trained) and ”en” (blank)
2. Add custom labels into the models
3. Disable the pipes that need not be altered in pre-trained model
4. Split training data into train and test set.
5. Train the models
6. Test the models

```
import spacy
# A Blank Model
nlp1= spacy.blank('en')
nlp1.add_pipe('ner')
nlp1.begin_training()

# An Existing NER Model
nlp2= spacy.load('en_core_web_sm')

# Getting pipeline component
ner1= nlp1.get_pipe("ner")
ner2= nlp2.get_pipe("ner")

# Adding labels to the 'ner1' and 'ner2'
for label in labels:
    ner1.add_label(label)
    ner2.add_label(label)

#Disable pipeline components that doesn't need to change in nlp2
pipe_exceptions=["ner","trf_wordpiecer","trf_tok2vec"]
unaffected_pipes=[pipe for pipe in nlp2.pipe_names if pipe not in pipe_exceptions]
```

Figure 4.10: Training: Steps 1, 2 and 3

The code in fig.4.10 is used for steps 1,2 and 3 mentioned above. The training begin by importing spacy library. A new blank model for the English language is then created and stored in a variable called nlp1. A named entity recognition (NER) pipeline component is added to nlp1 and training for the model is initiated. An existing NER model called ’en_core_web_sm’ is loaded into nlp2 using the spacy.load() function. The NER pipeline components for both models are retrieved and stored in variables called ner1 and ner2, respectively. Labels are added to both ner1 and ner2 by iterating through a list of labels and using the add_label() function.

```python
#Splitting the Training Corpus into Train and Test Sets
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(all_train_data, test_size=0.2, random_state=42)
print("Train Data:",len(train_data),"\nTest Data:",len(test_data))
```

```
Train Data: 1032
Test Data: 258
```

(a) Step 4: Train-Test Split

```python
# Train the NER component
import random
from spacy.util import minibatch, compounding
from spacy.training.example import Example
n_iter = 10
other_pipes = [pipe for pipe in nlp2.pipe_names if pipe != 'ner']
with nlp2.disable_pipes(*other_pipes):
    optimizer1 = nlp1.create_optimizer()
    optimizer2 = nlp2.create_optimizer()
    for i in range(n_iter):
        random.shuffle(train_data)
        losses1 = {}
        losses2 = {}
        batches = minibatch(train_data, size=compounding(4.0, 32.0, 1.001))
        for batch in batches:
            texts, annotations = zip(*batch)
            for j in range(0,len(texts)):
                doc1 = nlp1.make_doc(texts[j])
                doc2 = nlp2.make_doc(texts[j])
                example1 = Example.from_dict(doc1, annotations[j])
                example2 = Example.from_dict(doc2, annotations[j])
                nlp1.update([example1], sgd=optimizer1, drop=0.35, losses=losses1)
                nlp2.update([example2], sgd=optimizer2, drop=0.35, losses=losses2)
        print(f'nlp1_Epoch {i}: {losses1}')
        print(f'nlp2_Epoch {i}: {losses2}')
# Save the trained model to disk
nlp1.to_disk('/content/drive/MyDrive/Final (Completed)/nlp_model_1')
nlp2.to_disk('/content/drive/MyDrive/Final (Completed)/nlp_model_2')
```

(b) Step 5: Training Models

Figure 4.11: Training: Steps 4 and 5

Finally, certain pipeline components in nlp2 are disabled by creating two lists. The first list, pipe_exceptions, contains the names of the pipeline components that should be kept, while the second list, unaffected_pipes, contains the names of all the pipeline components in nlp2 except for those in pipe_exceptions. This allows for selective enabling or disabling of pipeline components as needed.

Now for training the NER component, the snippet in fig.4.11 first splits the train_data [fig.4.9] in train_data and test_data and will train two spaCy models using the train_data.

In step 4, the train_test_split() function from the sklearn.model_selection module is

used to split the all_train_data corpus into two separate sets - train_data and test_data. The test_size parameter is set to 0.2, which means that the test set will contain 20% of the total data, while the training set will contain the remaining 80%. The random_state parameter is set to 42, which is a seed value used to ensure that the same random train-test split is generated each time the code is run. The sizes of the resulting training and test sets 1032 and 258 sentences respectively.

In step 5, two NER models (nlp1 and nlp2) are trained using the train_data. The training is done for 10 iterations using the Stochastic Gradient Descent (sgd) optimizer. First, a list of other pipeline components is created which doesn't include the NER component. Then, these other components are disabled for the nlp2 model.

After that, the train_data is randomly shuffled and split into batches of size compounding(4.0, 32.0, 1.001). Then, for each batch, texts and annotations are extracted and used to create Example objects for both nlp1 and nlp2. These Example objects are used to update the models using the update() method. The dropout rate of 0.35 is used to prevent overfitting. The losses are computed for each epoch for both nlp1 and nlp2 and printed.

Finally, the trained models are saved to disk using the to_disk() method as nlp_model_1 and nlp_model_2.

In step 6 [fig.4.12], we test the two models along with default "en_core_web_sm" model. We test the performance of three different NLP models, i.e., Blank, Existing, and Default models, on the test data.

First, the default NLP model "en_core_web_sm" is loaded using the Spacy library. Then, labels are added to the default model using the "add_label" method of the named entity recognition (NER) component.

Next, the test data is split into texts and annotations, and three examples lists (example1, example2, and example3) are created to store the processed data by the three models. For each text in the test data, the corresponding document is created using the "make_doc" method of each model. Then, using the "from_dict" method of the Spacy "Example" class, the document and its annotations are converted into an example and

appended to the corresponding examples list.

Finally, the performance of each model is evaluated on the test data using the "evaluate" method of the NLP object and stored in scores1, scores2, and scores3, respectively. The Performance Evaluation will be covered in Chapter 5. Now we have our domain-specific NLP Models and based on the performance evaluation of the two models that will be discussed in Chapter 5, we select nlp_model_2 to carry on the implementation of the framework.

```
#Test the NLP Models
#Default NLP Model
nlp3= spacy.load('en_core_web_sm')
#Adding labels to default NLP Model
ner3= nlp3.get_pipe("ner")
for label in labels:
    ner3.add_label(label)
#Testing on test_data
texts, annotations = zip(*test_data)
example1=[]
example2=[]
example3=[]
for i in range(0,len(texts)):
    doc1 = nlp1.make_doc(texts[i]) #Blank
    doc2 = nlp2.make_doc(texts[i]) #Existing
    doc3 = nlp3.make_doc(texts[i]) #Default
    example1.append(Example.from_dict(doc1, annotations[i]))
    example2.append(Example.from_dict(doc2, annotations[i]))
    example3.append(Example.from_dict(doc3, annotations[i]))
scores1=nlp1.evaluate(example1) #Blank
scores2=nlp2.evaluate(example2) #Existing
scores3=nlp3.evaluate(example3) #Default
```

Figure 4.12: Training: Steps 6

4.5 Generation of Cypher Script

Now that we have an NLP Model trained with domain-specific NER pipe that will identify PMJAY's entities from text, the entities and relationship between entities will be extracted. To build a Knowledge graph, triplets are necessary. These triplets will comprise a source entity, relationship and a target entity. The entities will form nodes

and relationships will form edges in the knowledge graph. Please refer to "6.ipynb" for full code. The steps involved are as follows:

1. Extract Entities
2. Extract Relations
3. Make Triplets
4. Add properties to the nodes/entities
5. Create Cypher Script

In step 1 shown by snippet in fig.4.13 will process a list of sentences and extract pmjay's process flow specific named entities in the text. The input given is process_flow_sentences_2.csv [fig.3.5] which contains the sentences describing process flow of the PMJAY scheme.

```
import spacy
pmjay_nlp= spacy.load('/content/drive/MyDrive/Final (Completed)/nlp_model_2')
```

```
#Extration of Entities from pfs sentences also create ent-label dictionary
#v2

ent_pair=[] # [(sentence,[ent_1, ent_2]),...]
el_dict={} #Entity-Label dictionary
for i in range(0,len(sentences)):
    e_pair=[]
    d= pmjay_nlp(sentences[i])
    for ent in d.ents:
        e_pair.append(ent)
        el_dict[ent.text]=ent.label_
    if(len(e_pair)>1):
        ent_pair.append((sentences[i],e_pair))
print("Entity Pairs:",len(ent_pair),"\n",ent_pair)
print("Entity Label Dict:",len(el_dict),"\n",el_dict)
```

Figure 4.13: Step 1: Extraction of Entities

In this [fig.4.13], spacy library is imported and a pre-trained spaCy model i.e., nlp_model_2 is loaded. This code snippet creates an entity pair list and an entity-label dictionary from a list of sentences. It initializes an empty list ent_pair to store sentence-entity pair tuples and an empty dictionary el_dict to store entity-label mappings. Then, it iterates through each sentence in the input list of sentences, and for each sentence, it applies a spaCy NLP pipeline, pmjay_nlp, to extract entities. It appends the entity pair (ent.text, ent.label_) to a list e_pair and maps the entity text to its label in the dictionary

el_dict. If the e_pair list has more than one entity, it appends the sentence-entity pair tuple (sentence, e_pair) to the ent_pair list. The output is list of entity pairs (ent_pair) and the entity-label dictionary (el_dict) shown in [fig.4.14].

```
Entity Pairs: 94
 [('The Patient checks eligibility through Call-Centre', [Patient, Call-Centre]),
 ('The Patient checks eligibility through PMJAY Kiosk', [Patient, PMJAY Kiosk]),
 ('The Patient checks eligibility through EHCP Registration Desk', [Patient, EHCP Registration Desk]),
 ('PMJAY Kiosk is assigned a PMAM', [PMJAY Kiosk, PMAM]), ('EHCP Registration Desk is assigned a PMAM', [EHCP Registration Desk, PMAM]),
 ('PMAM verifies identity and eligibility through BIS', [PMAM, BIS]), ('BIS creates Silver Record', [BIS, Silver Record]),
  .
  .
  .
 ('Feedback is forwarded to NHA', [Feedback, NHA]), ('Call-Centre requests for Feedback', [Call-Centre, Feedback]),
 ('General Doctor is a Doctor', [General Doctor, Doctor]), ('Treating Specialist is a Doctor', [Treating Specialist, Doctor]),
 ('General Specialist is a Doctor', [General Specialist, Doctor]), ('Treating Doctor is a Doctor', [Treating Doctor, Doctor]),
 ('Treating Doctor can also be MEDCO', [Treating Doctor, MEDCO]), ('General Doctor can also be MEDCO', [General Doctor, MEDCO]),
 ('General Specialist can also be MEDCO', [General Specialist, MEDCO]), ('Treating Specialist can also be MEDCO', [Treating Specialist,
 MEDCO]), ('Empanelled Hospital appoints a MEDCO', [Empanelled Hospital, MEDCO]), ('PMJAY Beneficiary is a Patient', [PMJAY Beneficiary,
 Patient]), ('Empanelled Hospital has Doctor', [Empanelled Hospital, Doctor])]

Entity Label Dict: 36
 {'Patient': 'PERSON', 'Call-Centre': 'ORG', 'PMJAY Kiosk': 'CSC', 'EHCP Registration Desk': 'CSC',
 'PMAM': 'PERSON', 'BIS': 'PORTAL', 'Silver Record': 'DOCUMENT', 'Golden Record': 'DOCUMENT', 'e-card': 'DOCUMENT',
 'PMJAY Beneficiary': 'PERSON', 'Consultation': 'PROCESS', 'General Doctor': 'PERSON', 'General Specialist': 'PERSON',
 'Diagnostics': 'PROCESS', 'Pre-Authorization Form': 'DOCUMENT', 'MEDCO': 'PERSON', 'Empanelled Hospital': 'ORG',
 'Pre-Authorization Request': 'PERMISSION', 'ISA': 'ORG', 'SHA': 'ORG', 'Insurance Company': 'ORG', 'TMS': 'PORTAL',
 'Picture': 'DOCUMENT', 'Treatment': 'PROCESS', 'Treating Doctor': 'PERSON', 'Treating Specialist': 'PERSON',
 'Treatment Report': 'DOCUMENT', 'Medical Package': 'DOCUMENT', 'PMJAY': 'ORG', 'Treatment Procedure': 'DOCUMENT',
 'Claim': 'DOCUMENT', 'NHA': 'ORG', 'Medical Report': 'DOCUMENT', 'EHCP': 'ORG', 'Feedback': 'DOCUMENT', 'Doctor': 'PERSON'}
```

Figure 4.14: Entities and Labels

ent_pair is a list that will store the pairs of entities found in each sentence. el_dict is a dictionary that will be used to store a key value pair of entities and labels in the sentence. For each sentence, it uses the spaCy model to process the text and identify any named entities. The code then loops through each sentence in the list. The named entity and its label are added to the el_dict dictionary.If there are more than one named entities in the sentence, the sentence and the list of entities are added to the ent_pair list.

Now that we have extracted entities [fig.4.14], next we extract the relations from those sentences and make triplets which is step 2 and 3 shown in fig.4.15. The extraction of relations from these sentences is simple as the sentences describing the process flow are simple. So the relations between two entities are nothing but the phrase between the two entities. The code below shows the extraction of relationships.

The code initializes an empty list called triplets_1. It then loops through each tuple in the ent_pair list, which contains a sentence and a list of named entities. For each tuple, the code extracts the first and second named entities (stored in e[1][0] and e[1][1], respectively) and the sentence that they appear in (e[0]). The re.search function is then used to search for a substring of the sentence that occurs between the two named entities.

```
#Extraction of Relations and creating Level 1 triplets for KG
import re
triplets_1=[] # [[ent_1, ent_2, relation],...]
for e in ent_pair:
    e1= str(e[1][0])
    e2= str(e[1][1])
    sent= e[0]
    result= re.search(e1+"(.*)"+e2,sent)
    triplets_1.append([e1,e2,result.group(1).strip()])
triplets_1
```

Figure 4.15: Relation Extraction and Triplets

This is done by constructing a regular expression pattern that matches the first named entity (e1), followed by any number of characters (.*), followed by the second named entity (e2). The result.group(1).strip() call extracts the substring matched by the .* pattern and strips any leading or trailing whitespace. Finally, a list containing the first named entity, the second named entity, and the extracted substring is appended to the triplets_1 list. So triplets_1 will be a list of lists, where each inner list contains three elements: the first named entity, the second named entity, and the substring that appears between them in the original sentence. This can be thought of as a "relation triplet" that captures the semantic relationship between the two entities. A sample of these triplets is shown in fig.4.16

In step 4 [fig.4.18], we add properties to the entities. These properties describe the entity in the context of the PMJAY scheme. The "unique_entities_added_properties.csv" [fig.4.17] contains the properties for each entity.

This code in fig.4.18 creates a dictionary that maps each named entity to a string of properties associated with it. It initializes an empty dictionary called ep_dict that will be used to store the entity:properties mappings. The ue dataframe contains information about named entities, including their names and some associated properties. The first column of the ue dataframe (which is index) is dropped and the resulting dataframe is stored in a variable called ue_2. The code then loops through each row of the ue_2 dataframe using an integer index i. Each row represents a single named entity and its associated properties. The values in the row are stored in a variable called d as a pandas

```
triplets

[['Patient', 'Call-Centre', 'checks eligibility through'],
 ['Patient', 'PMJAY Kiosk', 'checks eligibility through'],
 ['Patient', 'EHCP Registration Desk', 'checks eligibility through'],
 ['PMJAY Kiosk', 'PMAM', 'is assigned a'],
 ['EHCP Registration Desk', 'PMAM', 'is assigned a'],
 ['PMAM', 'BIS', 'verifies identity and eligibility through'],
 ['BIS', 'Silver Record', 'creates'],
 ['BIS', 'Golden Record', 'creates'],
 ['BIS', 'e-card', 'creates'],
 .
 .
 .
 ['Treating Specialist', 'MEDCO', 'can also be'],
 ['Empanelled Hospital', 'MEDCO', 'appoints a'],
 ['PMJAY Beneficiary', 'Patient', 'is a'],
 ['Empanelled Hospital', 'Doctor', 'has'],
 ['Patient', 'Am I Eligible App', 'checks eligibility through'],
 ['Patient', 'Am I Eligible Website', 'checks eligibility through'],
 ['General Doctor', 'Hospitalization', 'decides'],
 ['General Specialist', 'Hospitalization', 'decides'],
 ['General Doctor', 'Medication', 'decides'],
 ['General Specialist', 'Medication', 'decides']]
```

Figure 4.16: Triplets

0	Treatment Procedure	name:"Treatment Procedure"	description: "The standart treament guidelines...	url: "https://www.pmjay.gov.in/resources/docum...
1	Insurance Company	name:"Insurance Company"	description:"The colaborated Insurance Company...	NaN
2	Medication	name:"Medication"	description: "The medications are provided to ...	NaN
3	Am I Eligible App	name:"Am I Eligible App"	description:"The Am I Eligible App is a moblie...	NaN
4	TMS	name:"TMS"	full_form:"Transaction Management System"	description:"It is an online platform develope...
5	Picture	name:"Picture"	description:"The Picture of the PMJAY Benefici...	NaN
6	Claim	name:"Claim"	description:"When the beneficiary seeks treatm...	NaN
7	EHCP	name:"EHCP"	full_form:"Empanelled Health Care Provider"	description:"Empanelled Health Care Providers ...

Figure 4.17: Entity Properties

series. An empty string called prop is initialized that will be used to concatenate the properties associated with the named entity. For each value in the series, the code checks whether the value is a string. If it is, the value is appended to the prop string along with a comma separator. After looping through all the values in the series, the final property string is cleaned up by removing the trailing comma and any leading or trailing

```
#Getting Properties for each Entity
ep_dict={} #dictionary containing entity:properties
ue_2=ue.drop(['0'], axis=1)
for i in range(0,len(ue_2)):
    d=(ue_2.iloc[i])
    prop=""
    for v in d:
        #print(v,":",type(v))
        if(type(v)==type('')):
            prop=prop+" "+str(v)+","
    prop=(prop[:-1]).strip()
    ep_dict[ue['0'][i]]=prop
```

Figure 4.18: Step 4: Adding Properties to entities

whitespace.

```
Entity-Properties Dictionary

{'Treatment Procedure': 'name:"Treatment Procedure", description: "The standart
treament guidelines can be downloaded from the website.", url:
"https://www.pmjay.gov.in/resources/documents#stg"', 'Insurance Company':
'name:"Insurance Company", description:"The colaborated Insurance Company will
cover all the charges of the PMJAY Beneficiary subjected to their claim
criteria."', 'Medication': 'name:"Medication", description: "The medications are
provided to every eligible PMJAY Beneficiary by the Empanelled Hospital or any
other EHCP without any charges ."', 'Am I Eligible App': 'name:"Am I Eligible App",
description:"The Am I Eligible App is a moblie app using which a person can check
his/her own eligibility under PMJAY Scheme."',
.
.
.
'PMAM': 'name:"PMAM", full_form:"Pradhan Mantri Aarogya Mitra", description:"They
are trained professionals who are appointed by the empanelled hospitals to assist
and guide the beneficiaries of the scheme. They act as facilitators between the
beneficiary, the hospital, and the insurance provider to ensure that the
beneficiary receives timely and appropriate medical care under the scheme. PMAMs
help beneficiaries with the process of availing the benefits of the scheme,
including the verification of eligibility, generation of e-cards, and coordination
of hospitalization and treatment. They also provide information about the scheme,
guide beneficiaries through the treatment process, and address any concerns or
queries that the beneficiary may have."',
.
.
.
'General Doctor': 'name:"General Doctor", description:"A doctor appointed to a
PMJAY Beneficiary for intital consultation."'}
```

Figure 4.19: Entity Property Dictionary

Finally, the named entity and its associated properties are added as a key-value pair to the ep_dict dictionary shown in fig.4.19. So ep_dict will be a dictionary that maps each named entity (extracted from the ue dataframe) to a string of associated properties. The exact format of the property string will depend on the format of the ue dataframe and the specific properties associated with each named entity. This ep_dict dictionary will be used to create the required cypher script to build the knowledge graph for PMJAY process flow.

In he final step 5, the cypher script is created, which will be imported to Neo4j to visualize and query the knowledge graph thus built. In this step we firstly create a cypher script for creating unique nodes with their properties then we create the cypher script for adding relations amongst the nodes. And finally combine the two scripts and create the final cypher script which will be saved as pmjay_process_flow.cypher file.

```
#Cypher Query for Creating unique nodes with properties
c_query=[]
for i in range(0,len(ue)):
    en=ue['0'][i]
    prop=ue['1'][i]
    q="CREATE ("+ev_dict[en]+":"+el_dict[en]+"{"+ep_dict[en]+"})"
    c_query.append(q)
c_query
```

```
['CREATE (a:DOCUMENT{name:"Treatment Procedure", description: "The standart treament guidelines can
be downloaded from the website.", url: "https://www.pmjay.gov.in/resources/documents#stg"})',
 'CREATE (b:ORG{name:"Insurance Company", description:"The colaborated Insurance Company will cover
all the charges of the PMJAY Beneficiary subjected to their claim criteria."})',
 'CREATE (c:PROCESS{name:"Medication", description: "The medications are provided to every eligible
PMJAY Beneficiary by the Empanelled Hospital or any other EHCP without any charges ."})',
 'CREATE (d:MOBILE_APP{name:"Am I Eligible App", description:"The Am I Eligible App is a moblie app
using which a person can check his/her own eligibility under PMJAY Scheme."})',
 'CREATE (e:PORTAL{name:"TMS", full_form:"Transaction Management System", description:"It is an
online platform developed by the National Health Authority (NHA) for managing the transactions and
```

Figure 4.20: Creating unique nodes with properties

The code in fig.4.20 creates a list of Cypher queries for creating unique nodes with properties in a Neo4j graph database. It initializes an empty list called c_query that will be used to store the Cypher queries. The ue dataframe contains information about named entities, including their names and some associated properties. For each row in the ue dataframe, the code extracts the named entity name (en) and its associated properties (prop). Then a Cypher query is created that will create a unique node in the graph database with the entity name (en) as the node label and the associated properties as node properties. The node is assigned a unique variable name using the ev_dict dictionary (which is generated using generate_variables(n) function, please refer “6.ipynb”), which is a dictionary containing unique variables assigned for each entity. The node label is assigned using the el_dict dictionary, which stores a key-value pair of entities and labels in the sentence. The Cypher query is then appended to the c_query list. After looping through all the rows of the ue dataframe, c_query will be a list of Cypher queries, where each query creates a unique node in the graph database with a different entity name and

associated properties.

```
#Use rel_pair to make relations cypher query
rel_pairs=[] #(source entity's variable, relation, target entitiy's variable)
for r in triplets_1:
    rel_pairs.append((ev_dict[r[0]],r[2],ev_dict[r[1]]))
rel_pairs
```

```
[('z', 'checks eligibility through', 'ad'),
 ('z', 'checks eligibility through', 'al'),
 ('z', 'checks eligibility through', 'ai'),
 ('al', 'is assigned a', 'am'),
 ('ai', 'is assigned a', 'am'),
 ('am', 'verifies identity and eligibility through', 'w'),
 ('w', 'creates', 'ab'),
 ('w', 'creates', 'v'),
 ('w', 'creates', 'ah'),
 ('ah', 'is given to the', 't'),
 ('am', 'informs about entitlements to the', 't'),
 ('z', 'becomes', 't'),
 ('t', 'is sent for', 'x'),
 ('x', 'is done by', 'an'),
```

Figure 4.21: Creating relations between entities

Now, the code in fig.4.21 creates a cypher query for creating relations between the unique entities. For this first variables must be extracted that represents the node/entity uniquely.The code in fig.4.21 creates a list of tuples called rel_pairs that will be used to store information about relationships between entities. triplets_1 [fig.4.16 is a list of lists where each inner list contains three elements: the first entity, the second entity, and the relationship between them. For each list in triplets_1, the code extracts the first entity (r[0]), the second entity (r[1]), and the relationship between them (r[2]). Then a tuple with three elements: the variable name of the first entity's node (as assigned in ev_dict), the relationship between the entities (r[2]), and the variable name of the second entity's node (also as assigned in ev_dict) is created. The tuple is then appended to the rel_pairs list.

After looping through all the lists in triplets_1, rel_pairs will be a list of tuples, where each tuple contains information about a relationship between two entities in the graph database. The rel_pairs list can then be used to create Cypher queries that will add the relationships between the entities to the graph database.

And now we finally create the cypher query as shown in fig.4.22. This code loops

through a list of tuples called rel_pairs. Each tuple in the list contains information about a relationship between entities. For each tuple, the code creates a relationship label by replacing any spaces in the second element of the tuple (which represents the name of the relationship) with underscores. This is done using the re.sub method from the re module, which performs a search and replace operation using a regular expression.

```
for r in rel_pairs:
    rel= re.sub(' ','_',r[1])
    c_query.append("CREATE ("+r[0]+")-[:"+rel+"]->("+r[2]+")")

for q in c_query:
    print(q)
```

```
CREATE (ai:CSC{name:"EHCP Registration Desk", description:"The EHCP registration desk is an online pc
CREATE (aj:PERSON{name:"MEDCO", full_form:"Medical Coordinator", description:"A medical coordinator i
CREATE (ak:DOCUMENT{name:"Medical Report", description:"A medical report is a document that contains
CREATE (al:CSC{name:"PMJAY Kiosk", description:"A physical facility that is set up to provide informa
CREATE (am:PERSON{name:"PMAM", full_form:"Pradhan Mantri Aarogya Mitra", description:"They are traine
CREATE (an:PERSON{name:"General Doctor", description:"A doctor appointed to a PMJAY Beneficiary for i
CREATE (z)-[:checks_eligibility_through]->(ad)
CREATE (z)-[:checks_eligibility_through]->(al)
CREATE (z)-[:checks_eligibility_through]->(ai)
CREATE (al)-[:is_assigned_a]->(am)
CREATE (ai)-[:is_assigned_a]->(am)
CREATE (am)-[:verifies_identity_and_eligibility_through]->(w)
```

Figure 4.22: PMJAY Process Flow Cypher

The resulting label is then used in a Cypher query that is appended to a list called c_query. The query creates a relationship between two nodes in a graph database. The first node is represented by the first element of the tuple, and the second node is represented by the third element of the tuple. After all the queries have been created and added to c_query, all the queries are concatenated into a single string called cypher_text with each query separated by a newline character. Finally, any whitespace at the beginning or end of the cypher_text string is removed using the strip() method. cypher_text thus generated is saved as pmjay_cypher.cypher file. It used to populate a graph database with nodes and relationships between them based on the information provided in rel_pairs. A sample of the query is shown in fig.4.23.

In the next section we will used thus created pmjay_process_flow.cypher to see visualization of the Knowledge Graph for PMJAY process flow in Neo4j Desktop.

```
Cypher Query Sample

CREATE (h:ORG{name:"EHCP", full_form:"Empanelled Health Care Provider", description:"Empanelled Health Care Providers (EHCPs) ... out of pocket."})
CREATE (s:PROCESS{name:"Diagnostics", description:"The medical tests and procedures ... blood tests and biopsy."})
CREATE (t:PERSON{name:"PMJAY Beneficiary", description:"An individual who is eligible ... hospitalization expenses."})
CREATE (m:DOCUMENT{name:"Feedback", description:"Feedback refers to the opinions, ... its intended outcomes."})
CREATE (am:PERSON{name:"PMAM", full_form:"Pradhan Mantri Aarogya Mitra", description:"They are trained professionals ... may have."})
CREATE (r:ORG{name:"ISA", full_form:"Implementation Support Agency", description:"An organization responsible for ... and evaluation."})
CREATE (h)-[:provides]->(s)
CREATE (s)-[:is_provided_to]->(t)
CREATE (t)-[:gives]->(m)
CREATE (am)-[:requests]->(m)
CREATE (m)-[:is_forwarded_to]->(r)
```

Figure 4.23: PMJAY Process Flow Cypher Script Sample

4.6 Visualization and Query in Neo4j

The Cypher script (pmjay_process_flow.cypher) fig.4.23 procured from the previous section is used to populate a knowledge graph database in Neo4j with nodes and relationships. The installation guide for Neo4j is given in the Appendix.

The pmjay_process_flow.cypher is imported to Neo4j and run. Once the script runs successfully and populates the graph database, we use MATCH (n) RETURN n query which retrieves all nodes in the graph database and returns them as a result set to generate the resulting graph shown in fig.4.24.

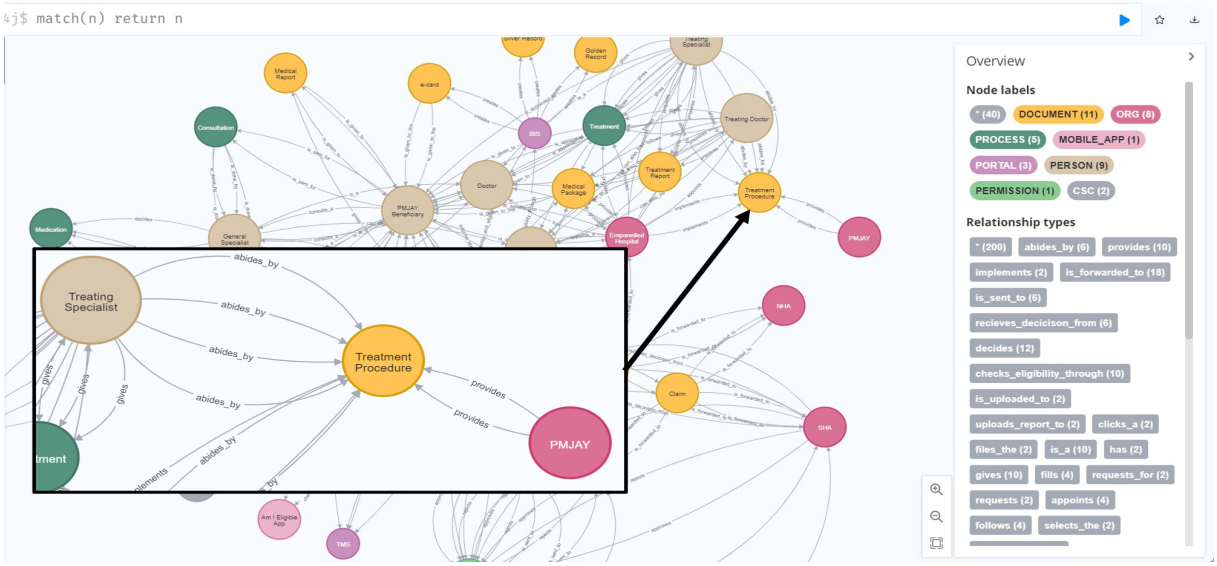


Figure 4.24: Knowledge Graph for PMJAY Process Flow

This visualization of how each node is connected to another node gives a comprehensive idea of the relations between these nodes. Each of the nodes are labeled which

makes it easier to distinguish them and filter. The properties that can be held in these nodes describes the node itself and each node has its own identity. Also the relations are capable of holding properties for itself.

The use cases of this Knowledge Graph is discussed in Chapter 6. In the next chapter 5, we will see the performance evaluation of NLP Models that were trained with different bases but on the same training set of corpus.

Chapter 5

Performance Evaluation of NLP Models

In Chapter 4, we have trained two NLP Models on the data set specific to the domain of PMJAY.The two models trained were (a) nlp_model_1 and (b) nlp_model_2. They were tested along with default "en_core_web_sm" model and the evaluation scores were recorded as depicted by the code in fig.5.1. nlp1 is the nlp model trained on a blank base and nlp2 is the trained nlp model trained on an extisting based model.

The code splits a training corpus into train and test sets using the train_test_split() function from the sklearn.model_selection module. The all_train_data is split into train_data and test_data using a test size of 0.2 and a random state of 42. The lengths of the train and test data are 1032 and 258, which means 258 sentences are used to test the performance of these models.

After splitting the data, the code loads two NLP models using the spacy.load() function from the spacy module. The first model is loaded with a blank English model and is assigned to nlp1. The second model is loaded with the en_core_web_sm model and is assigned to nlp2.

The code then loads a third NLP model, nlp3, with the en_core_web_sm model using the spacy.load() function. The labels are added to the ner pipeline of nlp3 using the add_label() method.

```
#Test the NLP Models
#Default NLP Model
nlp3= spacy.load('en_core_web_sm')
#Adding labels to default NLP Model
ner3= nlp3.get_pipe("ner")
for label in labels:
    ner3.add_label(label)
#Testing on test_data
texts, annotations = zip(*test_data)
example1=[]
example2=[]
example3=[]
for i in range(0,len(texts)):
    doc1 = nlp1.make_doc(texts[i]) #Blank
    doc2 = nlp2.make_doc(texts[i]) #Existing
    doc3 = nlp3.make_doc(texts[i]) #Default
    example1.append(Example.from_dict(doc1, annotations[i]))
    example2.append(Example.from_dict(doc2, annotations[i]))
    example3.append(Example.from_dict(doc3, annotations[i]))
scores1=nlp1.evaluate(example1) #Blank
scores2=nlp2.evaluate(example2) #Existing
scores3=nlp3.evaluate(example3) #Default
```

Figure 5.1: Testing NLP Models

The NLP models are then tested on the test_data using a for loop. The texts and annotations are extracted from the test_data using the zip() function and assigned to texts and annotations respectively. Three examples, example1, example2, and example3, are created using the Example.from_dict() method for each NLP model. The evaluation scores for each model are then calculated using the evaluate() method and assigned to scores1, scores2, and scores3 respectively.

5.1 Performance Metrics

The performance metrics [17] used for the evaluation are as follows:-

1. **Precision**: Precision is the ratio of correctly predicted positive results to the total number of predicted positive results. In the context of NLP, precision measures how many of the predicted entities or classifications were actually correct. For example, if the NLP model correctly identified 85 out of 100 entities, then the precision would be 85
2. **Recall**: Recall is the ratio of correctly predicted positive results to the total number

of actual positive results. In the context of NLP, recall measures how many of the actual entities or classifications were identified correctly by the model. For example, if the NLP model correctly identified 92 out of 100 entities, then the recall would be 92

3. **F1 score**: F1 score is a weighted average of precision and recall, calculated as $2 * (precision * recall)/(precision + recall)$. It ranges from 0 to 1, where a score of 1 represents perfect precision and recall, and a score of 0 represents the worst possible performance.
4. **Speed**: Speed measures the time taken by the NLP model to process a given text. It is usually reported in terms of seconds per document or tokens per second.

5.2 Results

The fig.5.2 shows the performance evaluation results for three NLP models. The metrics used for evaluation are Precision, Recall, F1 Score, and Speed (examples processed per second). The first NLP model is the default NLP model loaded using the en_core_web_sm model. The second NLP model, NLP Model 1, is a blank base model loaded using the en model. The third NLP model, NLP Model 2, is a pre-trained base model loaded using the en_core_web_sm model.

Performance Evaluation

	Metric	Default NLP Model	NLP Model 1 -[Blank Base]	NLP Model 2 -[Pre-trained Base]
0	Precision	0.3414 (34.14 %)	0.9895 (98.95 %)	0.9961 (99.61 %)
1	Recall	0.2415 (24.15 %)	0.9921 (99.21 %)	1.0 (100.0 %)
2	F1 Score	0.2829 (28.29 %)	0.9908 (99.08 %)	0.998 (99.8 %)
3	Speed (examples processed/sec)	4338.99	15324.52	4242.04

Figure 5.2: Performance of NLP Models

The precision, recall, and F1 score for each model are presented in the table as percentages, along with their numerical values in parentheses. The speed of each model, measured in examples processed per second, is also provided in the table.

The performance of each NLP model can be explained by the way the models were

trained and their underlying architectures. The default NLP model is a general-purpose model trained on a large dataset with a wide range of text types and topics. As such, it may not be optimized for the specific task at hand, and its performance may suffer as a result.

On the other hand, NLP Model 1 with a blank base was trained from scratch on the task-specific data used in this evaluation. This allows the model to focus solely on the specific task at hand and achieve high accuracy on that task. Similarly, NLP Model 2 with a pre-trained base was initialized with a pre-trained model that had already learned many linguistic features from a large corpus of text. This pre-training allows the model to leverage the learned features to better handle the task at hand, resulting in high accuracy.

The speed of each model can also be explained by their architectures. The default NLP model may be faster than the other two models because it is a simpler model with fewer added components, whereas the other two models have more added components and more complex architectures. However, this tradeoff between speed and accuracy should be considered depending on the specific use case and the importance of accuracy versus processing speed.

Conclusion Based on the results, NLP Model 2 with the pre-trained base performs the best, with precision, recall, and F1 score close to or equal to 1.0, indicating high accuracy. NLP Model 1 with the blank base also performs well, with precision, recall, and F1 score above 0.98. The default NLP model performs the worst, with precision, recall, and F1 score below 0.35. However, it is the fastest model in terms of processing examples, with a speed of 4338.99 examples per second. So, the choice of using NLP Model 2 that has a pre-trained base is the conclusion of this performance evaluation. NLP Model 2 is saved as nlp_model_2.

Chapter 6

Use Cases

In this chapter, we will discuss the uses cases of two Knowledge Graphs of PMJAY Process Flow with aid of Neo4j Desktop.

6.1 Knowledge Graphs

The two Knowledge Graphs are:-

1. **PMJAY Process Flow Knowledge Graph**: It captures the process involved in the scheme for a patient to avail the benefits of PMJAY. It has the stakeholders, their interactions with each other, the relation which connects two entities. the visualization of this Knowledge Graph in given in fig.6.1. This Knowledge Graph is created from the Process Flow Documents of PMJAY.

2. **Knowledge Graph of dummy records**: It has data about 60 patients under who went through the processes involved in the PMJAY scheme. The visualization of this Knowledge Graph in given in fig.6.2. This Knowledge graph [fig. 6.2] is a dummy record that holds the data related to each and every node/entity in the PMJAY Process Flow with patients.

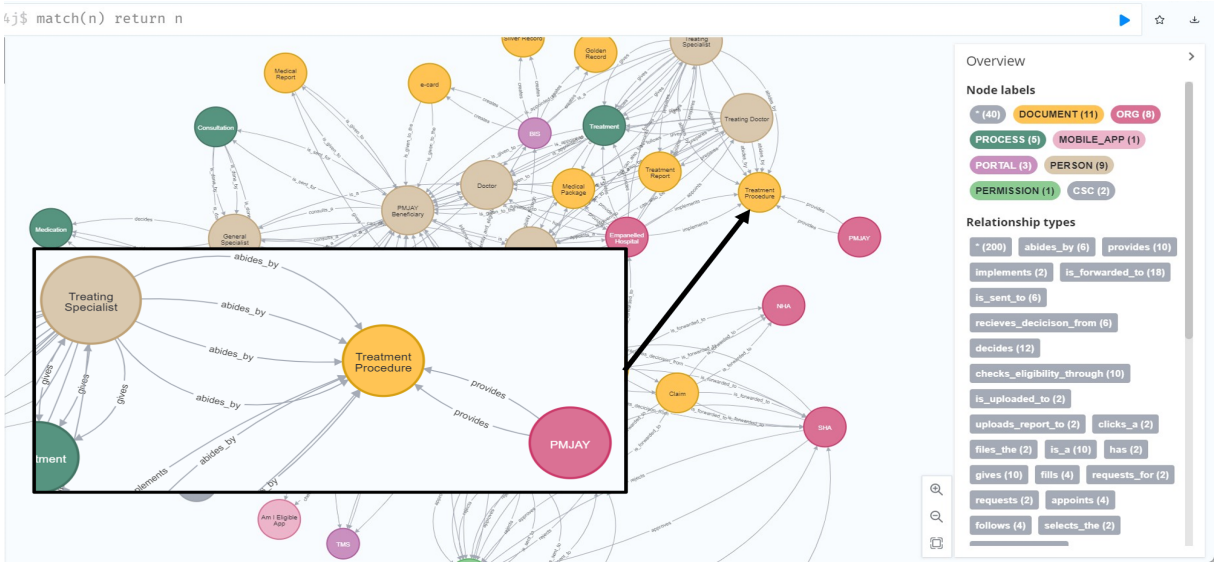


Figure 6.1: PMJAY Process Flow Knowledge Graph

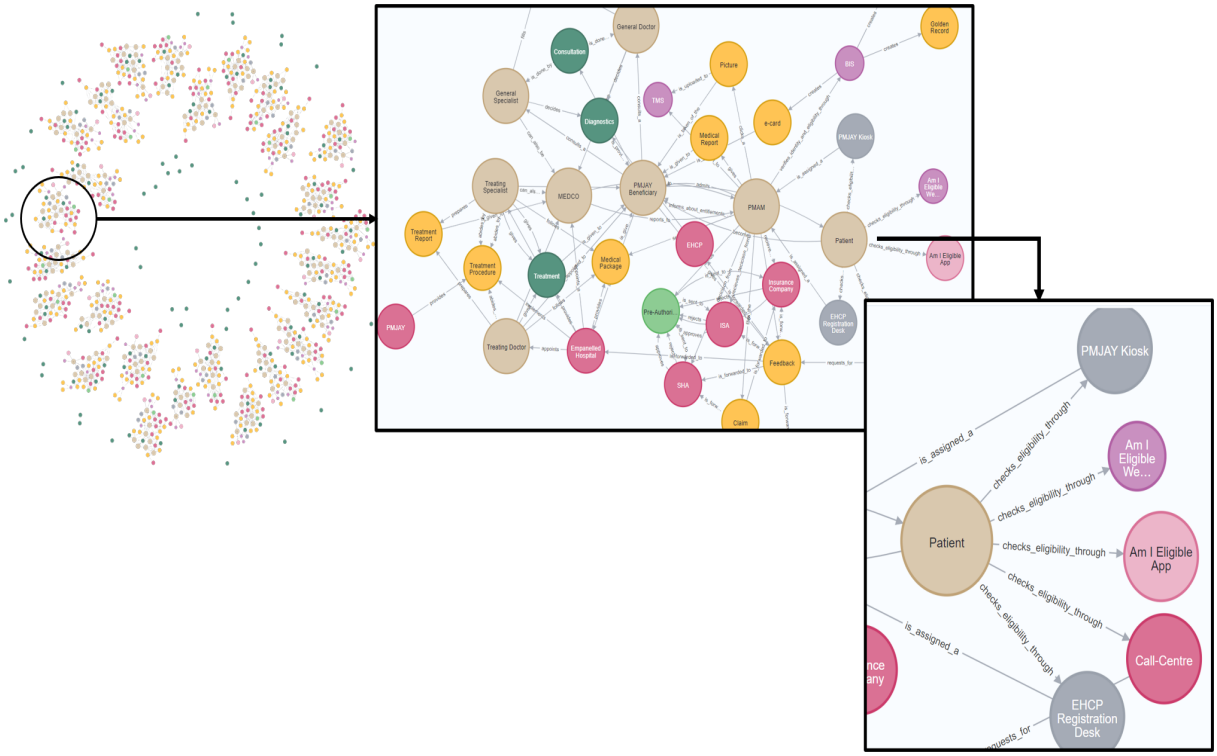


Figure 6.2: Dummy Records Graph

6.2 Queries

The following types of queries distinguishes a Knowledge Graph from a relational database based on strengths of graph databases like Neo4j, which are optimized for working with complex, interconnected data. We shall see examples of each in sections 6.2 and 6.3 for

the above mentioned two Knowledge Graphs. [1]

6.2.1 Traversing Relationships

It's easy to traverse relationships between nodes, which is not as straightforward in a relational database. For example, you can use the Cypher query language to find all nodes that are connected to a particular node through a specific relationship type:

```
Cypher Query
MATCH (n1)-[:REL]->(n2)
WHERE n1.name = "Node 1"
RETURN n2
```

This query finds all nodes that are related to "Node 1" through the "REL" relationship.

PMJAY Process Flow Knowledge Graph:

The code given in fig.6.4 finds the nodes which are connected to the node n1 through the relationship "checks_eligibility_through", thus giving the entities through which a patient checks his or her eligibility in PMJAY Scheme. The Graph output and the text output is shown in fig.6.3 and fig.6.4.

Knowledge Graph of PMJAY dummy records:

It's easy to traverse relationships between nodes, which is not as straightforward in a relational database. For example, to find all nodes that are connected to a particular node through a specific relationship type. The code given below finds the nodes which are connected to the node n1 through the relationship "checks_eligibility_through", thus giving the entities through which a patient checks his or her eligibility in PMJAY Scheme. The Graph output and the text output is shown in fig.6.5.

```
//Traversing relationships between nodes:
//Find all nodes that are connected to a particular node
//through a specific relationship type
```

```
1 //Traversal:
2 MATCH (n1)-[:checks_eligibility_through]→(n2)
3 RETURN n2 limit 5
```

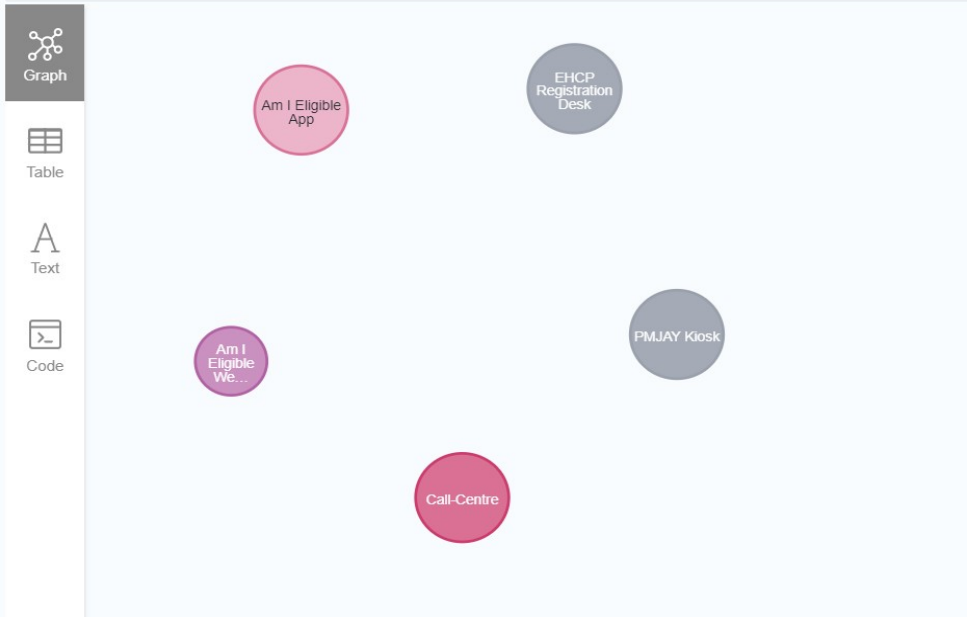


Figure 6.3: (a)Traversing Relationship 1: Graph Nodes

```
1 //Traversal:
2 MATCH (n1)-[:checks_eligibility_through]→(n2)
3 RETURN n2 limit 5
```

Graph
Table
Text
Code

```
"n2"
{"name":"Call-Centre","description":"A call centre to provide information and ass
ddress queries related to eligibility, enrolment, e-cards, empanelled hospitals,
{"name":"PMJAY Kiosk","description":"A physical facility that is set up to provid
 government offices, and community centers, and are equipped with computers, prin
trained personnel who help beneficiaries with the registration process, generate
{"name":"EHCP Registration Desk","description":"The EHCP registration desk is an
ties can create their profiles, and submit their credentials, infrastructure deta
{"name":"Am I Eligible App","description":"The Am I Eligible App is a moblie app
{"name":"Am I Eligible Website","description":"The Am I Eligible website is a fea
e is an online tool that allows individuals to check their eligibility for PMJAY
```

Figure 6.4: (b)Traversing Relationship 1: Text

```
MATCH (n1)-[:checks_eligibility_through]->(n2)
WHERE n1.patient_id = "3"
RETURN n2
```

6.2.2 Pattern Matching

We can use pattern matching to find nodes and relationships that match a particular pattern. For example, you can use the Cypher query language to find all nodes that have a particular label and are connected to other nodes through specific relationships:

```
Cypher Query
MATCH (n1:Label1)-[:REL]->(n2:Label2)
RETURN n1, n2
```

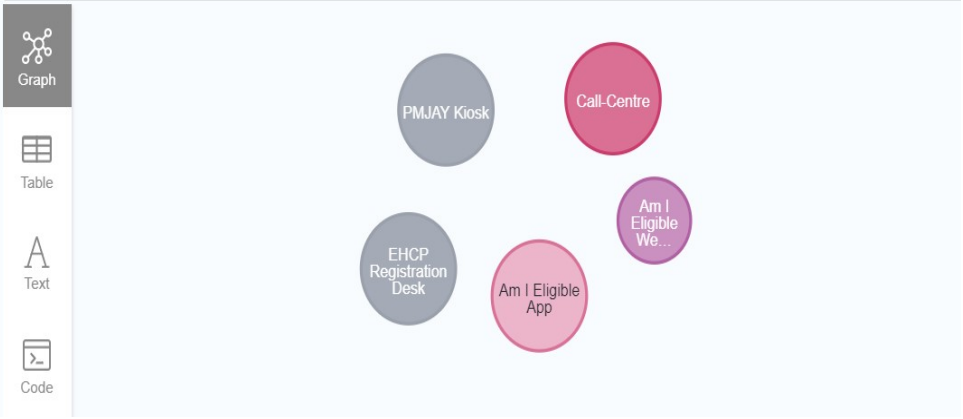
This query finds all nodes with the "Label1" label that are connected to nodes with the "Label2" label through the "REL" relationship.

PMJAY Process Flow Knowledge Graph The query given in fig.6.6 finds all nodes with the "PERSON" label that are connected to nodes with the "PERSON" label through the any "*" relationship. The Graph output is shown in fig.6.6.

Knowledge Graph of PMJAY dummy records The query given fig.6.7 finds all nodes with the "PERSON" label that are connected to nodes with the "PERSON" label through the any "*" relationship. The Graph output is shown in fig.6.7.

```
//Pattern Matching:
//finds all nodes with the "PERSON" label
//that are connected to nodes with the "PERSON" label
//through the any "*" relationship.
MATCH (n1:PERSON)-[*]->(n2:PERSON)
RETURN n1, n2 LIMIT 50
```

```
1 //Traversing relationships between nodes:
2 //Find all nodes that are connected to a particular node
3 //through a specific relationship type
4
5 MATCH (n1)-[:checks_eligibility_through]→(n2)
6 WHERE n1.patient_id = "3"
7 RETURN n2
```



(a) Graph Nodes

```
1 //Traversing relationships between nodes:
2 //Find all nodes that are connected to a particular node
3 //through a specific relationship type
4
5 MATCH (n1)-[:checks_eligibility_through]→(n2)
6 WHERE n1.patient_id = "3"
7 RETURN n2
```

Graph
Table
Text
Code

```
"n2"
{"app_acc_id":"3","type":"Am I Eligible App"}
{"website_acc_id":"n/a","type":"Am I Eligible Website","url":"n/a"}
{"caller_id":"n/a","type":"Call-Centre","call_centre_no":"n/a"}
{"location":"n/a","type":"PMJAY Kiosk","kiosk_id":"n/a"}
{"ehcp_name":"ECHP1","type":"EHCP Registration Desk"}
```

(b) Text

Figure 6.5: Traversing Relationship 2

```
1 //Pattern Matching:
2 //finds all nodes with the "PERSON" label
3 //that are connected to nodes with the "PERSON" label
4 // through the any "*" relationship.
5 MATCH (n1:PERSON)-[*]→(n2:PERSON)
6 RETURN n1, n2 LIMIT 10
```

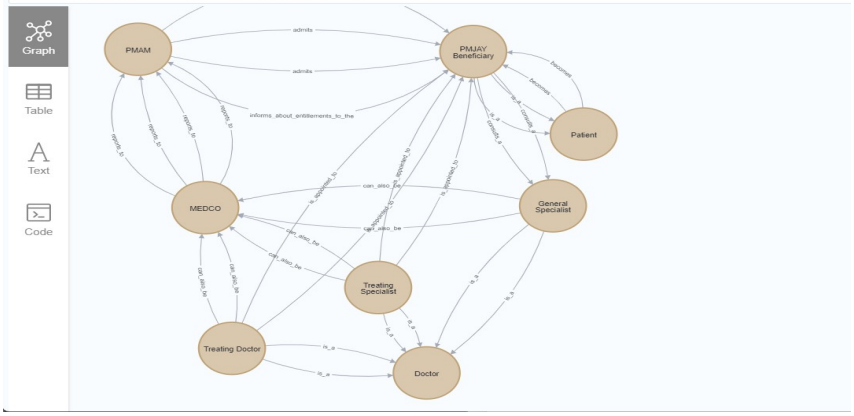


Figure 6.6: Pattern Matching 1

```
1 //Pattern Matching:
2 //finds all nodes with the "PERSON" label
3 //that are connected to nodes with the "PERSON" label
4 //through the any "*" relationship.
5 MATCH (n1:PERSON)-[*]→(n2:PERSON)
6 RETURN n1, n2 LIMIT 50
```

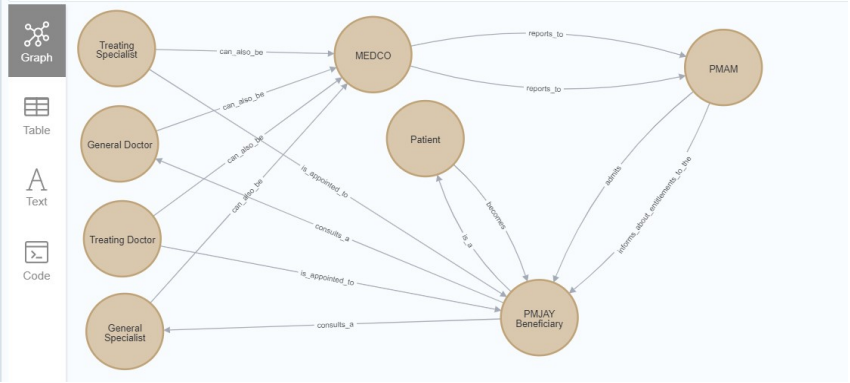


Figure 6.7: Pattern Matching 2

6.2.3 Additional Queries and Results

1. Workload: fig.6.8 shows which node has the maximum workload based on the no. of edges connected to it.

2. Treatment: fig.6.9 shows what treatment a certain patient recieved.

3. Schema Relations: fig.6.10 shows how are the schemas related to one another.

```
1  //Workload: Node with Max edges
2  MATCH (n)
3  OPTIONAL MATCH (n)-[r]-()
4  WITH n, count(distinct r) AS edge_count
5  ORDER BY edge_count DESC
6  LIMIT 1
7  RETURN n, edge_count
```

Graph Table

"n"	"edge_count"
{"pmam_id":"1","pmam_name":"PMAM1","type":"PMAM"}	17

Figure 6.8: Workload

```
1  //What treatment a patient got
2  match (n:PERSON{type:"Patient",patient_id:"2"})-[*]→
   (m:PROCESS{type:"Treatment"})
3  return m limit 1
```

Graph Table

"m"
{"t_id":"2","t_name":"T1","type":"Treatment"}

Figure 6.9: Treatment

```
1  // What is related, and how
2  CALL db.schema.visualization()
```

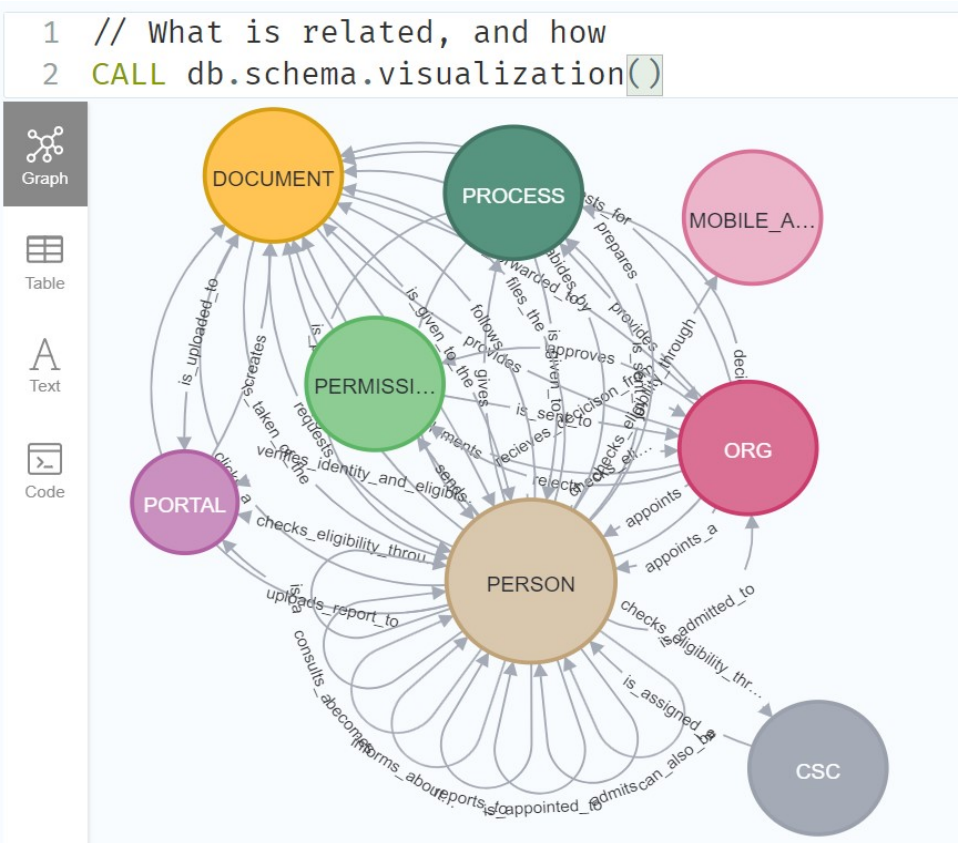


Figure 6.10: Schema Relation

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, an open-source framework for building knowledge graphs from domain-specific unstructured data has been presented. This framework is designed to counter some of the problems of domain-specific (PMJAY) unstructured data refining and entity recognition by building a custom NLP model having Named Entity Recognition (NER) trained on identifying domain-specific entities.

We made a set of regular expressions that can be used to refine PMJAY related unstructured data. The refined data is used to build the NLP model by training domain-specific NER component. Relation extraction between the extracted entities has been done using rule-based approach by analysing the dependencies in the data and also by a simple logic of "what's between the two entities is the relation between them" for specific process flow sentences procured from analysis of ER Diagrams depicting the process flow of PMJAY scheme.

With the extracted entities and relations, we have added properties for each entity and created a Cypher script to populate a graph database and thereby building a Knowledge Graph which is used to analyse the process flow from different perspectives of various stakeholders like Patients, Doctors, authorities such as NHA, SHA, ISA and insurance companies involved in the PMJAY scheme.

7.2 Future Work

1. Any NLP model is never 100% accurate. This deficiency is inherent in our NLP model too. However, the accuracy can be improved by training the model with more corpus annotated with relevant labels with the help of domain experts and also by use of BioBERT models which are highly trained on medical dataset. With proper identification of entities and their properties, the resulting knowledge graph can be applicable for a wide range of applications like Fraud Detection and Recommendation System.

2. The training of NER component here is done with non-overlapping entities which can also be extended to the capability of training overlapping entities. The pre-trained base model can also be experimented with other variations of en_core_web_sm like en_core_web_md and en_core_web_lg models.

3. The NLP model used here is in the English language. This framework can be applied to different language models and make the end result available in other native languages for a larger spectrum of people which will help in spreading the knowledge of the PMJAY scheme.

4. This project only trains the NER component of NLP in spacy. The other components of NLP Model like tagger and parser can also be trained on domain-specific data. This would give us a more robust domain-specific NLP Model. and this would improve the relation extraction process for more complex sentences with analysis of dependencies.

5. The latest technology, ChatGPT, an artificial intelligence chatbot developed by OpenAI and launched in November 2022, can be used to re-create the whole project by interactively communicating with it for the structures and ideas to build the necessary modules.

Appendix A: Code Book

A.1 1.py: Entities Recognition

```
"""1.ipynb:

Unique Entities and Possible Combination of Two Related Entities
Input:-
1. process_flow_sentences.csv
Output:-
1. unique_entities.csv: /content/drive/MyDrive/Final/
   unique_entities.csv
2. all_entities_pair.csv: /content/drive/MyDrive/Final/
   all_entities_pair.csv
"""

import pandas as pd
from google.colab import drive
drive.mount('/content/drive')

pfs=pd.read_csv('/content/drive/MyDrive/Final_(Completed)/
   process_flow_sentences_2.csv')
pfs
```

```
#Get All the Unique Entities from process_flow_sentences.csv
ent1=(pfs['ent_1'].drop_duplicates()).reset_index(drop=True)
ent2=(pfs['ent_2'].drop_duplicates()).reset_index(drop=True)
ent=[]
for e in ent1:
    ent.append(e)
for e in ent2:
    ent.append(e)
ent=list(set(ent))
ent

ue=pd.DataFrame({'unique_entities':ent})
ue

ue.to_csv('/content/drive/MyDrive/Final_(Completed)/
   unique_entities.csv',index=False)

#Just getting all entities pairs from process flow sentences
ent1=pfs['ent_1'].reset_index(drop=True)
ent2=pfs['ent_2'].reset_index(drop=True)

#All Entities Pair - aep
aep=pd.DataFrame({'ent1':ent1,'ent2':ent2})
print(len(aep))
aep

aep.to_csv('/content/drive/MyDrive/Final_(Completed)/
   all_entities_pair.csv',index=False)
```

```
"""#End Of File"""
```

A.2 2.py: OpenAI Generated Sentences

```
# -*- coding: utf-8 -*-
"""2.ipynb
AI Sentence Generation for training PMJAY specific NER Component
1. Get entity List from process_flow_sentences.csv
2. Use two entities at a time as keyword to generate 5 sentences
    having those two entities in the sentence.
3. The generated sentences can be used to train the NER Model
   more accurately.

Input : all_entities_pair.csv (/content/drive/MyDrive/Final/
   all_entities_pair.csv)
Output: ai_train_corpus.csv (/content/drive/MyDrive/Final/
   ai_train_corpus.csv)
"""

!pip install openai

"""Get openai_api_key from https://platform.openai.com/docs/
   quickstart/build-your-application"""

import os
import openai
openai.api_key='sk-
   rnCzb7xisTxsZPtzf1vxT3BlbkFJ3y3OaGPltEj4Pzl8HMdB'

import time
```

```
from tenacity import (
    retry,
    stop_after_attempt,
    wait_random_exponential,
) # for exponential backoff


@retry(wait=wait_random_exponential(min=1, max=60), stop=
   stop_after_attempt(6))
def completion_with_backoff(**kwargs):
    return openai.Completion.create(**kwargs)

""" generate_sentence(entity_pairs): To generate 5 sentences
   containing a pair of related entities.
Input: DataFrame having two ent1 and ent2 columns
Output: List """

def generate_sentences(keywords):
    sent_text=[]
    count=0
    for i in range(0,len(keywords)):
        if(count==1):
            print('Wait for 2 seconds to avoid -> [429: \'Too
               Many Requests\' or RateLimitError]...')
            time.sleep(2)
            count=0
        p='Create 10 sentence relating to Ayushman Bharat PMJAY
           with the following: '+keywords['ent1'][i]+','+keywords
           ['ent2'][i]
```

```
        print('Request:',i,': [',keywords['ent1'][i],',',
            keywords['ent2'][i],']')
        response= completion_with_backoff(model="text-davinci
            -003", prompt=p, temperature=0.7, max_tokens=3970)
        sent_text.append(response['choices'][0]['text'])
        count+=1
    return (sent_text)

from google.colab import drive
drive.mount('/content/drive/')

import pandas as pd
all_ent_pairs= pd.read_csv('/content/drive/MyDrive/Final (
    Completed)/all_entities_pair.csv')
sent_text= generate_sentences(all_ent_pairs)
ai_train_corp=''
for s in sent_text:
    ai_train_corp=ai_train_corp+" "+s
ai_train_corpus= pd.DataFrame({'ent_pair_sentences':sent_text})
ai_train_corpus.to_csv('/content/drive/MyDrive/Final (Completed)
    /ai_train_corpus2.csv',index=False)

"""Conclusion:
We have 1020 AI Generated Sentences in ai_train_corpus2.csv,
    which has 10 sentences for each entity pair. These sentences
    having domain-specific entities.
*Note: the generated sentences are un-refined.*
"""
```

A.3 3.py: Raw Data Extraction and Refinement

```
"""3.ipynb
3: Text Extraction and Refinement
3.1. Text Extraction From Govt. Documents
3.2. Text Refinement (Data Cleansing)
3.2.1. Level 1 Refinement
3.2.1.1. Level 1 Refinement of OpenAI generated domain specific
   sentences and joining with existing sentences.
3.2.2. Level 2 Refinement
3.2.3. Level 3 Refinement

'Input: Govt. Documents [PDFs] /content/drive/MyDrive/Final/
   PMJAY_Documents/' <br>
'Output: refined_text.csv /content/mydrive/MyDrive/Final/
   refine_text.csv'

3.1. Text Extraction
"""

#Installing Tika Library for Text Extration from PDF
!pip install tika

import pandas as pd #For Data Manipulation
from google.colab import drive #For Getting Documents From
   Google Drive
drive.mount('/content/drive') #Mounting the Google Drive
from tika import parser
```

```
# opening pdf file
pdf_1 = parser.from_file("/content/drive/MyDrive/Final (
    Completed)/PMJAY_Documents/Process Flow Guidebook.pdf")
pdf_2 = parser.from_file("/content/drive/MyDrive/Final (
    Completed)/PMJAY_Documents/PM-JAY Process Flow at Empanelled
    Hospitals.pdf")
raw_text = pdf_1['content']+pdf_2['content']
"""
3.2. Text Refinement (Data Cleansing)
3.2.1. Level 1 Refinement
Remove unnecessary spaces, newlines, characters and numbering
    from `raw_text`
"""
#Regular Expression Library
import re

#Level 1 Refinement Function for Raw Text
def refine_1(text):
    #Ordering Alphabets, Numerals and Roman Numerals
    a= re.compile(r'(\b(\d|\d\d)\.)|(\b(i|v|x)+\.)|(\b(I|V|X)
        +\.)|(\b([a-z]|[A-Z])\.($|\ )+)')
    b= re.compile(r'(\n|\s|!|#|%|\^|&|\*|\(|\))+')#New Lines,
        Spaces and Special characters
    c= re.compile(r'(\.+)') #Multiple dots (...)
    d= re.compile(r'(\d*\ Section\ \d\ *:\ *)') #Section
    e= re.compile(r'(\d*\ Chapter\ \d\ *:\ *)') #Chapter
    f= re.compile(r'(\d*\ Annexe\ \d\ *:\ *)') #Annexe
    g= re.compile(r'(^\ *\d*\ )') #Starting Numbers in a
        Sentence
```

```
    h= re.compile(r'    ') # Marking Dot (    )
    i= re.compile(r'(^[aA-zZ]*\.*$)') #Single-Word Sentences
    text= i.sub('',(h.sub('',(g.sub('',(f.sub('',(e.sub('',(d.
       sub('',(c.sub('.',(b.sub('␣',(a.sub('',text))))))))))))))
       )))
    return(text)

def text_to_sent_char(t):
    sent=pd.DataFrame()
    #print(sent)
    text=''
    for c in t:
        text=text+c
        if(c=='.'):
            sent= sent.append([text])
            #sent= sent.concat([text])
            text=''
    sent.rename(columns={0:'sentences'},inplace=True)
    sent= sent.reset_index(drop=True) #Resetting the Index
    return(sent)

import spacy
nlp= spacy.load('en_core_web_sm')

def text_to_sent_nlp(text):
    d=nlp(text)
    sentences=[]
    sent=''
    for tok in d:
```

```
        if(tok.text=='.' or tok.text==':'):
            sent=sent+tok.text
            sentences.append(sent.strip())
            sent=''
        else:
            sent=sent+" "+tok.text

    sent_df=pd.DataFrame(sentences) #Converting the list into
       DataFrame for ease of visual analysis
    for i in range(0,len(sent_df)):
        for j in range(0,6):
            sent_df[0][i]=(refine_1(str(sent_df[0][i]))).strip()
                #Level 1 Refinement of Each Sentence done 5
               times.
        if(sent_df[0][i]==''): #Checking for Empty Rows
            sent_df= sent_df.drop(i) #Removing Empty Rows
    sent_df= sent_df.reset_index(drop=True) #Resetting the Index
    sent_df.rename(columns={0:'sentences'},inplace=True) #
       Renaming the Column

    return(sent_df)

#Level 1 Refinement
text_1= (refine_1(raw_text)).strip()

"""3.2.1.1. Getting OpenAI generated domain specific sentences
   from `ai_train_corpus.csv`"""

#Getting AI Train Corpus
```

```
ai_text_df= pd.read_csv('/content/drive/MyDrive/Final_(Completed
    )/ai_train_corpus2.csv')
text_2=''
#Level 1 Refinement
for i in range(0, len(ai_text_df['ent_pair_sentences'])):
    text_2= text_2 + refine_1(ai_text_df['ent_pair_sentences'][i
        ])

#Splitting the text into Sentences
sent_df_1= text_to_sent_nlp(text_1)
sent_df_2= text_to_sent_char(text_2)

for i in range(0,len(sent_df_2['sentences'])):
    sent_df_2['sentences'][i]=(refine_1(sent_df_2['sentences'][i
        ])).strip()
    if(sent_df_2['sentences'][i]==''):
        sent_df_2= sent_df_2.drop(i) #Removing Empty Rows
sent_df_2= sent_df_2.reset_index(drop=True) #Resetting the Index

print('Lengths:\nsent_df_1:',len(sent_df_1),'\nsent_df_2:',len(
    sent_df_2))
sent_df= sent_df_1.append(sent_df_2)
print('sent_df:',len(sent_df))
sent_df= sent_df.reset_index(drop=True) #Resetting the Index
"""
Level 2 Refinement
Prepare possible regular expressions for variations of domain
    specific entities that can be present in a text/sentence.
```

```
    regex_pattern': 'A list of tuples (entity, reg_exp for
        variations of entity)
    Using this 'regex_pattern' replace the variations with
        corresponding domain specific entity.
    """
    #Regex Patterns for possible variations of unique entities
    regex_pattern=[]
    #am i eligible, aam - i-eligible
    regex_pattern.append(('Am I Eligible App',(re.compile(r'(\'*[aA
        ]+?m[\ -]*?[iI]+?[\ -]*?[eE]+?ligible[\ -]*?[aA]+?pp(lication
        )*s*([\ \']*?)*)')))))
    regex_pattern.append(('Am I Eligible Website',(re.compile(r'(\'
        *[aA]+?m[\ -]*?[iI]+?[\ -]*?[eE]+?ligible[\ -]*?[wW]+?ebsites
        *([\ \']*?)*)'))))
    regex_pattern.append(('BIS',(re.compile(r'((\bbis\b)|([bB]+?
        eneficiary\ *[iI]+?dentification\ *[sS]+?ystems*))'))))
    regex_pattern.append(('Call-Centre',(re.compile(r'[cC]+?all[\
        -]*?[cC]+?ent[re]?[er]?s*'))))
    regex_pattern.append(('Claim',(re.compile(r'\b[cC]+?laims*'))))
    regex_pattern.append(('Consultation',(re.compile(r'\b[cC]+?
        onsultations*'))))
    regex_pattern.append(('Diagnostics',(re.compile(r'[dD]+?
        iagnostics*'))))
    regex_pattern.append(('Doctor',(re.compile(r'\b[dD]+?octors*')))
        )
    regex_pattern.append(('e-card',(re.compile(r'[eE]+?[-\ ]*?[cC]+?
        ards*'))))
    regex_pattern.append(('EHCP',(re.compile(r'(\b[eE]+?[hH]+?[cC
        ]+?[pP]+?\b)|[eE]+?mpanelled\ *[hH]+?ealth\ *[cC]+?are\ *[pP
```

```
    ]+?roviders*'))))
regex_pattern.append(('EHCP␣Registration␣Desk',(re.compile(r'([
    eE]+?[hH]+?[cC]+?[pP]+?)+[-\␣]*?[rR]+?egistration[-\␣]*?([dD
    ]+?esks*)+'))))
regex_pattern.append(('Empanelled␣Hospital',(re.compile(r'([eE
    ]+?[hH]+?[cC]+?[pP]+?\␣*)*([eE]+?mpanelled\␣*)*[hH]+?ospitals
    *'))))
regex_pattern.append(('Feedback',(re.compile(r'[fF]+?eedbacks*')
    )))
regex_pattern.append(('General␣Doctor',(re.compile(r'[gG]+?
    eneral\␣*[dD]+?octors*'))))
regex_pattern.append(('General␣Doctor␣or␣General␣Specialist',(re
    .compile(r'[gG]+?eneral\␣*[dD]+?octors*((\␣*|/)|(/|\␣*))+([gG
    ]+?eneral\␣*)*[sS]+?pecialists*'))))
regex_pattern.append(('General␣Specialist',(re.compile(r'[gG]+?
    eneral\␣*[sS]+?pecialists*'))))
regex_pattern.append(('Golden␣Record',(re.compile(r'[gG]+?olden
    [\␣-]*?[rR]+ecords*'))))
regex_pattern.append(('Hospitalization',(re.compile(r'[hH]+?
    ispitali[zs]?ations*'))))
regex_pattern.append(('Insurance␣Company',(re.compile(r'(([iI]+?
    nsurers*)|([iI]+?nsurance\␣*([cC]+?ompan(y|ies))*)')))))
regex_pattern.append(('ISA',(re.compile(r'(\bisa\b)|([iI]+?
    mplementations*\␣*[sS]+?upport\␣*[aA]+?genc(y|ies))')))))
regex_pattern.append(('MEDCO',(re.compile(r'([mM]+?edco)|([mM]+
    edical\␣*[cC]+?o(-)*ordinators*)')))))
regex_pattern.append(('Medical␣Package',(re.compile(r'(([mM]+?
    edical\␣*)*([pP]+?ackages*))')))))
```

```
regex_pattern.append(('Medical␣Report',(re.compile(r'(([mM]+?
    edical\␣*)*([dD]+?iagnostics*\␣*)*[rR]+eports*)|([cC]+?
    linical\␣*[nN]+?otes*)|([dD]+?ischarge\␣*[sS]+?ummar(y|ies))'
    ))))
regex_pattern.append(('NHA',(re.compile(r'(\bnha\b)|([nN]+?
    ational\␣*[hH]+?ealth\␣*[aA]+?uthorit(y|ies))'))))
regex_pattern.append(('Patient',(re.compile(r'\bpatients*\b'))))
regex_pattern.append(('Picture',(re.compile(r'\bpictures*\b'))))
regex_pattern.append(('PMAM',(re.compile(r'(\b[pP]+?radhan\␣*[mM
    ]+?antri\␣*[aA]+?rogya\␣*[mM]+?itra\b)|\bpmam\b'))))
regex_pattern.append(('PMJAY',(re.compile(r'(\b[pP]+?radhan\␣*[
    mM]+?antri\␣*-*\␣*[jJ]+?an\␣*[aA]+?rogya\␣*[yY]+?ojana\b)|(\b
    [pP]+?[mM]+?\␣*-*\␣*[jJ]+?[aA]+?[yY]+?\b)'))))
regex_pattern.append(('PMJAY',(re.compile(r'([aA]+?yushman␣[bB
    ]+?harat\␣[sS]+?cheme)'))))
regex_pattern.append(('PMJAY',(re.compile(r'([aA]+?yushman␣[bB
    ]+?harat\␣(PMJAY)*)'))))
regex_pattern.append(('PMJAY',(re.compile(r'((AYUSHMAN␣BHARAT\␣)
    *PRADHAN\␣*MANTRI\␣*-*\␣*JAN\␣*AROGYA\␣*YOJANA)'))))
regex_pattern.append(('PMJAY␣Beneficiary',(re.compile(r'(\bPMJAY
    \␣*)*[bB]+?eneficiar(y|ies)'))))
regex_pattern.append(('PMJAY␣Kiosk',(re.compile(r'(\bPMJAY\␣*)*[
    kK]+?iosks*'))))
regex_pattern.append(('Pre-Authorization␣Form',(re.compile(r'[pP
    ]+?re\␣*-\␣*[aA]+?uthorization\␣*[fF]orms*\b'))))
regex_pattern.append(('Pre-Authorization␣Request',(re.compile(r'
    [pP]+?re\␣*-\␣*[aA]+?uthorization\␣*[rR]equests*\b'))))
regex_pattern.append(('SHA',(re.compile(r'(\b[sS]+?[hH]+?[aA]+?[
    sS]?\b)|([sS]+?tate\␣*[hH]+?ealth\␣*[aA]+?genc(y|ies))'))))
```

```
regex_pattern.append(('SHA␣or␣NHA',(re.compile(r'(SHA[sS]?\␣*/\␣
    *NHA)|(NHA*\␣*/\␣*SHA[sS]?)'))))
regex_pattern.append(('Silver␣Record',(re.compile(r'[sS]+?ilver
    [\␣-]*?[rR]+ecords*'))))
regex_pattern.append(('TMS',(re.compile(r'(\b[tT]+?[mM]+?[sS]+?\
    b)|([tT]+?ransactions*\␣*[mM]+?anagement\␣*[sS]+?ystems*)'))))
    )
regex_pattern.append(('Treating␣Doctor',(re.compile(r'[tT]+?
    reating\␣*[dD]+?octors*'))))
regex_pattern.append(('Treating␣Doctor␣or␣Treating␣Specialist',(
    re.compile(r'[tT]+?reating\␣*[dD]+?octors*((\␣*|/)|(/|\␣*))
    +([tT]+?reating\␣*)*[sS]+?pecialists*'))))
regex_pattern.append(('Treating␣Specialist',(re.compile(r'[tT]+?
    reating\␣*[sS]+?pecialists*'))))
regex_pattern.append(('Treatment',(re.compile(r'[tT]+?reatments*
    '))))
regex_pattern.append(('Treatment␣Procedure',(re.compile(r'[tT]+?
    reatment\␣*[pP]+?rocedures*'))))
regex_pattern.append(('Treatment␣Report',(re.compile(r'[tT]+?
    reatment\␣*[rR]+?eports*'))))

def refine_2(sent):
    for p in regex_pattern:
        sent=p[1].sub(p[0],sent)
    return(sent)

text=[]
for sent in sent_df['sentences']:
    text.append(refine_2(sent))
```

```
text=pd.DataFrame({'sentences':text})


"""Level 3 Refinement
Remove adjacent duplicate words from sentences of `text`
"""
def refine_3(sent):
    p= re.compile(r'\b(\w+)\s+\1\b')
    sent= p.sub(r'\1',sent)
    return(sent)


refined_t=[]
for sent in text['sentences']:
    refined_t.append(refine_3(sent))
text=pd.DataFrame({'sentences':refined_t})
text.to_csv('/content/drive/MyDrive/Final_(Completed)/
   refined_text2.csv')
"""#End Of File"""
```

A.4 4.py: NLP Training Corpus Creation

```
"""4.ipynb
Training Corpus Creation
1. Creation of Training Corpus for SpaCy NER Training**<br>
`Input: refined_text.csv`<br>
`Output: training_corpus.csv`
"""
from google.colab import drive
drive.mount('/content/drive/')
import pandas as pd
corpus=pd.read_csv('/content/drive/MyDrive/Final (Completed)/
    refined_text2.csv')#,index_col=0)
pfs=pd.read_csv('/content/drive/MyDrive/Final (Completed)/
    process_flow_sentences_2.csv')#,index_col=0)
"""Get entities and labels from `process_flow_sentences.csv`"""
#Create Unique Entity-Label Pairs
#Get ent_1, lab_1 and ent_2, lab_2 as tuples('entity','label')
    in a list
#Remove Duplicates
entities=[] #list of unique entities
labels=[] #list of unique labels
el_dict={} #Dictionary of Entities and Labels
el= [] #list of entities and labels
#Getting List of tuples(entity,labels)
for i in range(0,len(pfs)):
    el.append((pfs['ent_1'][i],pfs['lab_1'][i]))
    el.append((pfs['ent_2'][i],pfs['lab_2'][i]))
el= [*set(el)]
```

```
#Getting dictionary{entity:label} and list of Unique Labels
for t in el:
    el_dict[t[0]]=t[1]
    labels.append(t[1])
    entities.append(t[0])
labels=[*set(labels)]
entities=[*set(entities)]
entities.sort(key=len,reverse=True)
"""Below we save to disk, the **unique labes** as 'labels.csv'
   """
l=pd.DataFrame(labels)
l.to_csv('/content/drive/MyDrive/Final_(Completed)/labels.csv',
   index=False)
l.rename(columns={0:"labels"},inplace=True)
"""Now we have the following:
entities : List of all unique entities.
labels : List of all unique labels.
el : List of unique Tuples(entity, label)
el_dict :Dictionary Entity:Label
Use entity from 'entities' to find its start and end index in a
   sentence
Find entities in the sentences:
* Find it's start and end index
* Create the format: 't_data= [(sentence,{'entities':[(start_ind
   ,end_ind,label),...]}),...]'
"""
#Find entities in the sentences and annotate them with relevant
   labels
```

```
print("Finding entities in the sentences and annotating them
    with relevant labels")
import re
t=[] #List of Annotated sentences
for s in corpus['sentences']:
    index=[] #Stores start and end indices of entities in a
        sentence along with their labels
    for e in entities:
        #get start and end index of tok.text
        for i in re.finditer(e, s): #Finds indices of entity e
            in sentence s
            #index.append((e,i.start(),i.end(),el_dict[e])) #For
                Analysis Purpose Only
            index.append((i.start(),i.end(),el_dict[e]))
    index=sorted(index)
    t.append((s,index))
#Remove Sentences with no entities
delete_ind=[]
count=1
for i in range(0,len(t)):
    if(t[i][1]==[]):
        delete_ind.append(i)
delete_ind.sort(reverse=True)
len1= len(t)
for ind in delete_ind:
    t.pop(ind)
len2= len(t)
""" Removing Overlapping Entities
Steps :-
```

```
1. Identify entities with same start_index OR same end_index
2. Find difference in indices
3. Remove the entity with lesser difference value
"""
#Removing Overlapping Entities
for k in range(0,len(t)):
    checked_ind=[] #indices that has already been checked
    delete_ind=[] #Indices of those entities that needs to be
       removed
    for i in range(0,len(t[k][1])-1):
        if(i in checked_ind):
            continue
        else:
            for j in range(i+1,len(t[k][1])):
                s1=t[k][1][i][0] #start index of 1st entity
                e1=t[k][1][i][1] #end index of 1st entity
                s2=t[k][1][j][0] #start index of 2nd entity
                e2=t[k][1][j][1] #start index of 2nd entity
                if(e1 >= s2):#Step 1: Detection of Overlapping
                   Entities
                    checked_ind.append(j)
                    #Step 2: Getting indices that need to be
                       deleted
                    e1_diff= e1-s1
                    e2_diff= e2-s2
                    if(e1_diff > e2_diff):
                        delete_ind.append(j)
                    else:
                        delete_ind.append(i)
```

```
    if(delete_ind != []):#Step 3: Deletion
        delete_ind.sort(reverse=True)
        for ind in range(0,len(delete_ind)): #Pop
            #print(t[k][1])
            t[k][1].pop(delete_ind[ind])
training_corpus=[]
for r in t:
    training_corpus.append((r[0],{'entities':r[1]}))
training_corpus
#Saving the training corpus
tc=pd.DataFrame(training_corpus)
tc.rename(columns={0:"sentences",1:"annotations"},inplace=True)
tc.to_csv('/content/drive/MyDrive/Final_(Completed)/
   training_corpus.csv',index=False)
"""# End Of File"""
```

A.5 5.py: NLP Training and Performance Evaluation

```
"""5.ipynb
5: Train 2 NLP Models
1. From Blank NLP model
2. From Existing NLP model "en_core_web_sm"
* Split the training corpus into train and test sets.
* Train the two models with the training set
* Test the two models with the test set
    * Also test the default untrained NLP Model
* Compare the performances
"""
#Mounting Google Drive
from google.colab import drive
drive.mount('/content/drive/')
#Getting the training corpus as tc
import pandas as pd
tc= pd.read_csv('/content/drive/MyDrive/Final_(Completed)/
   training_corpus.csv')
#Converting the dataframe into list
all_train_data= list(tc.values)
#Converting the annotations into dictionary
import ast
for i in range(0,len(all_train_data)):
    all_train_data[i][1]=ast.literal_eval(str(all_train_data[i
       ][1]))
#Getting the labels
```

```
labels= pd.read_csv('/content/drive/MyDrive/Final_(Completed)/
    labels.csv')
#Splitting the Training Corpus into Train and Test Sets
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(all_train_data,
    test_size=0.2, random_state=42)
import spacy
# A Blank Model
nlp1= spacy.blank('en')
nlp1.add_pipe('ner')
nlp1.begin_training()
# An Existing NER Model
nlp2= spacy.load('en_core_web_sm')
# Getting pipeline component
ner1= nlp1.get_pipe("ner")
ner2= nlp2.get_pipe("ner")
# Adding labels to the 'ner1' and 'ner2'
for label in labels:
    ner1.add_label(label)
    ner2.add_label(label)
#Disable pipeline components that doesn't need to change in nlp2
pipe_exceptions=["ner","trf_wordpiecer","trf_tok2vec"]
unaffected_pipes=[pipe for pipe in nlp2.pipe_names if pipe not
    in pipe_exceptions]
# Train the NER component
import random
from spacy.util import minibatch, compounding
from spacy.training.example import Example
n_iter = 10
```

```
other_pipes = [pipe for pipe in nlp2.pipe_names if pipe != 'ner'
    ]
with nlp2.disable_pipes(*other_pipes):
    optimizer1 = nlp1.create_optimizer()
    optimizer2 = nlp2.create_optimizer()
    for i in range(n_iter):
        random.shuffle(train_data)
        losses1 = {}
        losses2 = {}
        batches = minibatch(train_data, size=compounding(4.0,
            32.0, 1.001))
        for batch in batches:
            texts, annotations = zip(*batch)
            for j in range(0,len(texts)):
                doc1 = nlp1.make_doc(texts[j])
                doc2 = nlp2.make_doc(texts[j])
                example1 = Example.from_dict(doc1, annotations[j
                    ])
                example2 = Example.from_dict(doc2, annotations[j
                    ])
                nlp1.update([example1], sgd=optimizer1, drop
                    =0.35, losses=losses1)
                nlp2.update([example2], sgd=optimizer2, drop
                    =0.35, losses=losses2)
        print(f'nlp1 Epoch {i}: {losses1}')
        print(f'nlp2 Epoch {i}: {losses2}')
# Save the trained model to disk
nlp1.to_disk('/content/drive/MyDrive/Final (Completed)/
    nlp_model_1')
```

```
nlp2.to_disk('/content/drive/MyDrive/Final_(Completed)/
    nlp_model_2')
#Tesing the Models along with default NLP Model - Performance
    Evaluation
#Default NLP Model
nlp3= spacy.load('en_core_web_sm')
#Adding labels to default NLP Model
ner3= nlp3.get_pipe("ner")
for label in labels:
    ner3.add_label(label)
#Testing on test_data
texts, annotations = zip(*test_data)
example1=[]
example2=[]
example3=[]
for i in range(0,len(texts)):
    doc1 = nlp1.make_doc(texts[i]) #Blank
    doc2 = nlp2.make_doc(texts[i]) #Existing
    doc3 = nlp3.make_doc(texts[i]) #Default
    example1.append(Example.from_dict(doc1, annotations[i]))
    example2.append(Example.from_dict(doc2, annotations[i]))
    example3.append(Example.from_dict(doc3, annotations[i]))
scores1=nlp1.evaluate(example1) #Blank
scores2=nlp2.evaluate(example2) #Existing
scores3=nlp3.evaluate(example3) #Default
#Dropping the Columns with no values
score_df1= (pd.DataFrame.from_dict(scores1)).dropna(axis=1) #
    Blank
```

```
score_df2= (pd.DataFrame.from_dict(scores2)).dropna(axis=1) #
    Existing
score_df3= (pd.DataFrame.from_dict(scores3)).dropna(axis=1) #
    Default
#Renaming the required columns
score_df1.rename(columns = {'ents_p':'ents_p1', 'ents_r':'
    ents_r1', 'ents_f':'ents_f1', 'speed':'speed1'}, inplace =
    True)
score_df2.rename(columns = {'ents_p':'ents_p2', 'ents_r':'
    ents_r2', 'ents_f':'ents_f2', 'speed':'speed2'}, inplace =
    True)
score_df3.rename(columns = {'ents_p':'ents_p3', 'ents_r':'
    ents_r3', 'ents_f':'ents_f3', 'speed':'speed3'}, inplace =
    True)
#Dropping the irrelevant Columns
score_df1.drop(['token_acc','token_p','token_r','token_f','
    ents_per_type'], axis=1,inplace=True)
score_df2.drop(['token_acc','token_p','token_r','token_f','
    ents_per_type'], axis=1,inplace=True)
score_df3.drop(['token_acc','token_p','token_r','token_f','
    ents_per_type'], axis=1,inplace=True)


res_score=pd.concat([score_df1,score_df2,score_df3],axis=1,join=
    'inner')
#Getting Average of all the scores
avg_val=[]
for v in res_score:
    sum=0
    for i in res_score[v].values:
```

```
            sum= sum+i
        sum= sum/len(res_score)
        avg_val.append(sum)
#For NLP Model 1 - [Trained on Blank Model]
avg_precision_1=str(round(avg_val[0],4))+" ("+str(round(avg_val
    [0]*100,2))+" %)"
avg_recall_1=str(round(avg_val[1],4))+" ("+str(round(avg_val
    [1]*100,2))+" %)"
avg_f1_score_1=str(round(avg_val[2],4))+" ("+str(round(avg_val
    [2]*100,2))+" %)"
avg_speed_1=round(avg_val[3],2)
#For NLP Model 2 - [Trained on Pre-existing Model]
avg_precision_2=str(round(avg_val[4],4))+" ("+str(round(avg_val
    [4]*100,2))+" %)"
avg_recall_2=str(round(avg_val[5],4))+" ("+str(round(avg_val
    [5]*100,2))+" %)"
avg_f1_score_2=str(round(avg_val[6],4))+" ("+str(round(avg_val
    [6]*100,2))+" %)"
avg_speed_2=round(avg_val[7],2)
#For Default NLP Model
avg_precision_3=str(round(avg_val[8],4))+" ("+str(round(avg_val
    [8]*100,2))+" %)"
avg_recall_3=str(round(avg_val[9],4))+" ("+str(round(avg_val
    [9]*100,2))+" %)"
avg_f1_score_3=str(round(avg_val[10],4))+" ("+str(round(avg_val
    [10]*100,2))+" %)"
avg_speed_3=round(avg_val[11],2)
#Creating a DataFrame for all the three Scores
```

```
data = {'Metric': ['Precision', 'Recall', 'F1 Score', 'Speed (
    examples processed/sec)'],
        'Default NLP Model': [avg_precision_3, avg_recall_3,
            avg_f1_score_3, avg_speed_3],
        'NLP Model 1 -[Blank Base]': [avg_precision_1,
            avg_recall_1, avg_f1_score_1, avg_speed_1],
        'NLP Model 2 -[Pre-trained Base]': [avg_precision_2,
            avg_recall_2, avg_f1_score_2, avg_speed_2]}
performance_df = pd.DataFrame(data)
from tabulate import tabulate
print("Performance Evaluation\n",tabulate(performance_df,
    headers='keys', tablefmt='psql'))
#End Of File
```

A.6 6.py: Creation of PMJAY Cypher Script

```
"""6.ipynb
1. Entity-Relation Extraction
2. Creation of Triples for KG along with properties of Nodes/
   Entities
Input: process_flow_sentneces.csv, PMJAY_NLP_Model
Output: triples.csv or triple.json
"""
from google.colab import drive
drive.mount('/content/drive/')
import pandas as pd
pfs= pd.read_csv('/content/drive/MyDrive/Final (Completed)/
   process_flow_sentences_2.csv')
sentences=pfs['sentences']
import spacy
pmjay_nlp= spacy.load('/content/drive/MyDrive/Final (Completed)/
   nlp_model_2')
#Extration of Entities from pfs sentences also create ent-label
   dictionary
ent_pair=[] # [(sentence,[ent_1, ent_2]),...]
el_dict={} #Entity-Label dictionary
for i in range(0,len(sentences)):
    e_pair=[]
    d= pmjay_nlp(sentences[i])
    for ent in d.ents:
        e_pair.append(ent)
        el_dict[ent.text]=ent.label_
    if(len(e_pair)>1):
```

```
            ent_pair.append((sentences[i],e_pair))
print("Entity Pairs:",len(ent_pair),"\n",ent_pair)
print("Entity Label Dict:",len(el_dict),"\n",el_dict)

#Adding Missing Data
el_dict['Am I Eligible App']='MOBILE_APP'
el_dict['Am I Eligible Website']='PORTAL'
el_dict['Medication']='PROCESS'
el_dict['Hospitalization']='PROCESS'
ent_pair.append(("The Patient checks eligibility through Am I
    Eligible App"),['Patient','Am I Eligible App']))
ent_pair.append(("The Patient checks eligibility through Am I
    Eligible Website"),['Patient','Am I Eligible Website']))
ent_pair.append(("General Doctor decides Hospitalization"),['
    General Doctor','Hospitalization']))
ent_pair.append(("General Specialist decides Hospitalization")
    ,['General Specialist','Hospitalization']))
ent_pair.append(("General Doctor decides Medication"),['General
    Doctor','Medication']))
ent_pair.append(("General Specialist decides Medication"),['
    General Specialist','Medication']))
print("Entity Pairs:",len(ent_pair),"\n",ent_pair)
print("Entity Label Dict:",len(el_dict),"\n",el_dict)

#Extraction of Relations and creating Level 1 triplets for KG
import re
triplets_1=[] # [[ent_1, ent_2, relation],...]
for e in ent_pair:
    e1= str(e[1][0])
```

```
    e2= str(e[1][1])
    sent= e[0]
    result= re.search(e1+"(.*)"+e2,sent)
    triplets_1.append([e1,e2,result.group(1).strip()])


trip_df=pd.DataFrame(triplets_1)
trip_df.rename(columns={0:'source',1:'target',2:'edge'},inplace=
    True)
# Adding Properties to unique Entities
ue= pd.read_csv('/content/drive/MyDrive/Final_(Completed)/
    unique_entities_added_properties.csv')
ue.sample(5)
#Getting Properties for each Entity
ep_dict={} #dictionary containing entity:properties
ue_2=ue.drop(['0'], axis=1)
for i in range(0,len(ue_2)):
    d=(ue_2.iloc[i])
    prop=""
    for v in d:
        if(type(v)==type('')):
            prop=prop+" "+str(v)+","
    prop=(prop[:-1]).strip()
    ep_dict[ue['0'][i]]=prop
#assign unique variables to each entity
#maintain a dict of {entity:variable} to use it for creating
    relations
def generate_variable(n):
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    sequences = []
```

```
    if n < 26:
        return(alphabet[n])
    else:
        first = alphabet[(n // 26) - 1]
        second = alphabet[n % 26]
        return(first + second)
    return sequences
ev_dict={}
for i in range(0,len(ue)):
    var=generate_variable(i)
    e=str(ue['0'][i])
    ev_dict[e]=var
#Cypher Query for Creating unique nodes with properties
c_query=[]
for i in range(0,len(ue)):
    en=ue['0'][i]
    prop=ue['1'][i]
    q="CREATE ("+ev_dict[en]+":"+el_dict[en]+"{"+ep_dict[en]+"})
       "
    c_query.append(q)
c_query
#Use rel_pair to make relations cypher query
rel_pairs=[] #(source entity's variable, relation, target
   entitiy's variable)
for r in triplets_1:
    rel_pairs.append((ev_dict[r[0]],r[2],ev_dict[r[1]]))
rel_pairs
#Cypher Query for Creating Relations between entities
# CREATE (a)-[:relation]->(b)
```

```
for r in rel_pairs:
    rel= re.sub(' ','_',r[1])
    c_query.append("CREATE (")
for r in rel_pairs:
    rel= re.sub(' ','_',r[1])
    c_query.append("CREATE (")
cypher_text=""
for q in c_query:
    cypher_text=cypher_text+q+"\n"
cypher_text=cypher_text.strip()
cypher_text=""
for q in c_query:
    cypher_text=cypher_text+q+"\n"
cypher_text=cypher_text.strip()
#Saving the Cypher Script
text_file = open("/content/drive/MyDrive/Final (Completed)/
   pmjay_process_flow.cypher", "w")
n = text_file.write(cypher_text)
text_file.close()
"""# End Of file"""
```

A.7 7.py: Creating PMJAY Dummy Record Cypher Script

```
"""use_case.ipynb
1. Make PMJAY Process Flow Cypher Script for Knowledge Graph
2. Create a dictionary of properties for each entity so that it
   can be used to get appropriate data for each process flow of
   a patient
"""
#Mounting Google Drive
from google.colab import drive
drive.mount('/content/drive/')
"""Get entities from the Process Flow Sentences"""
import pandas as pd
pfs= pd.read_csv('/content/drive/MyDrive/Final_(Completed)/
   process_flow_sentences_2.csv')
sentences= pfs['sentences']
import spacy
pmjay_nlp= spacy.load('/content/drive/MyDrive/Final_(Completed)/
   nlp_model_2')
"""Extracting entities from Process Flow Sentences"""
#Extration of Entities from pfs sentences also create ent-label
   dictionary
ent_pair=[] # [(sentence,[ent_1, ent_2]),...]
el_dict={} #Entity-Label dictionary
for i in range(0,len(sentences)):
    e_pair=[]
    d= pmjay_nlp(sentences[i])
    for ent in d.ents:
```

```
            e_pair.append(ent)
            el_dict[ent.text]=ent.label_
        if(len(e_pair)>1):
            ent_pair.append((sentences[i],e_pair))
print("Entity Pairs:",len(ent_pair),"\n",ent_pair)
print("Entity Label Dict:",len(el_dict),"\n",el_dict)
#Adding Missing Data
el_dict['Am I Eligible App']='MOBILE_APP'
el_dict['Am I Eligible Website']='PORTAL'
el_dict['Medication']='PROCESS'
el_dict['Hospitalization']='PROCESS'
ent_pair.append((("The Patient checks eligibility through Am I
    Eligible App"),['Patient','Am I Eligible App']))
ent_pair.append((("The Patient checks eligibility through Am I
    Eligible Website"),['Patient','Am I Eligible Website']))
ent_pair.append((("General Doctor decides Hospitalization"),['
    General Doctor','Hospitalization']))
ent_pair.append((("General Specialist decides Hospitalization")
    ,['General Specialist','Hospitalization']))
ent_pair.append((("General Doctor decides Medication"),['General
    Doctor','Medication']))
ent_pair.append((("General Specialist decides Medication"),['
    General Specialist','Medication']))
print("Entity Pairs:",len(ent_pair),"\n",ent_pair)
print("Entity Label Dict:",len(el_dict),"\n",el_dict)
el_df= pd.DataFrame(el_dict, index=[0])
entities=pd.DataFrame(el_df.columns)
entities.sample(3)
"""Output: entities
```

```
Getting Relations from the Process Flow Sentences
"""
#Extraction of Relations and creating Level 1 triplets for KG
import re
triplets_1=[] # [[ent_1, ent_2, relation],...]
for e in ent_pair:
    e1= str(e[1][0])
    e2= str(e[1][1])
    #if(e1!='Doctor' and e2!='Doctor'):
        #sent= e[0]
        #result= re.search(e1+"(.*)"+e2,sent)
        #triplets_1.append([e1,e2,result.group(1).strip()])
    sent= e[0]
    result= re.search(e1+"(.*)"+e2,sent)
    triplets_1.append([e1,e2,result.group(1).strip()])
"""Output: triplets_1
"""
#Assigning Variables to each Entity
#Method to generate variables
def generate_variable(n):
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    sequences = []
    if n < 26:
        return(alphabet[n])
    elif n < 702:
        first = alphabet[(n // 26) - 1]
        second = alphabet[n % 26]
        return(first + second)
    else:
```

```
        first = alphabet[(((n//26)-1)//26)-1]
        second = alphabet[(((n//26)-1)%26)]
        third = alphabet[n%26]
        return(first+second+third)
    return sequences
ev_dict={}
var=[]
for i in range(0,len(entities)):
    var=generate_variable(i)
    e=str(entities[0][i])
    ev_dict[e]=var
ev_dict
"""Preparing a dictionary of entities and properties"""
ent_prop= pd.read_csv('/content/drive/MyDrive/Final_(Completed)/
   ent_properties.csv')
ent_prop_dict={} #A dictionary to store properties of each
   entity
#Now make ent_properties dictionary from ent_prop dataframe with
    entity as the key
for index,row in ent_prop.iterrows():
    entity= row['entities']
    prop=row['properties']
    if (entity not in ent_prop_dict):
        ent_prop_dict[entity]=[prop]
    else:
        ent_prop_dict[entity].append(prop)
#Output: ent_prop_dict
#Get the PMJAY Dummy Records Document
import pandas as pd
```

```
pdr= pd.read_csv('/content/drive/MyDrive/Final_(Completed)/
    pmjay_dummy_record.csv')
value=pdr['app_acc_id'][0]
value_str= str(value)
#Data Cleaning: Convertin Nan Values to "n/a" and float type
    into integer type
for index, row in pdr.iterrows():
    for col, value in row.iteritems():
        # check if value is NaN
        if isinstance(value,float):
            # convert NaN to "n/a"
            if str(value)=='nan':
                pdr.at[index, col] = "n/a"
            # convert float to integer
            else:
                pdr.at[index, col] = int(value)
"""For each record of 'pdr', build a process flow knowledge
    graph"""
#Starting with a single record
rec= pdr.iloc[0]
query_chunk=[]
c_query=[]
for i in range(0,len(pdr)):
    rec= pdr.iloc[i]
    #Add values to the properties of each entity for each record
    ent_prop_dict #Entity-Properties Names Dictionary
    pv_dict={} #Property-Values Dictionary
    ep_dict={} #Entity-Properties with values Dictionary
    for e in entities[0]:
```

```
        if e != 'Doctor':
            for p in ent_prop_dict[e]:
                pv_dict[p]=rec[p]
            ep_dict[e]=pv_dict.copy()
        pv_dict.clear()
    #Converting the dictionary into required cypher format to
       create cypher query to create nodes
    for key,value in ep_dict.items():
        properties=""
        for key2,val2 in value.items():
            if isinstance(val2,float):
                val2= int(val2)
            properties+= f"{key2}:\"{val2}\","
        properties+=f"type:\"{key}\""
        c_query.append("CREATE (a"+ev_dict[key]+":"+el_dict[key]+
           "{"+properties+"})")
    #Use rel_pair to make relations cypher query
    rel_pairs=[] #(source entity's variable, relation, target
       entitiy's variable)
    for r in triplets_1:
        rel_pairs.append((ev_dict[r[0]],r[2],ev_dict[r[1]]))
    #Cypher Query for Creating Relations between entities
    # CREATE (a)-[:relation]->(b)
    for r in rel_pairs:
        rel= re.sub(' ','_',r[1])
        c_query.append("CREATE (a"+r[0]+")-[:"+rel+"]->(a"+r[2]+"
           )")
    cypher_text=""
    for q in c_query:
```

```
            cypher_text=cypher_text+q+"\n"
        cypher_text=cypher_text.strip()
        query_chunk.append(cypher_text)
        c_query.clear()
query_df= pd.DataFrame(query_chunk)
query_df.to_csv("/content/drive/MyDrive/Final_(Completed)/
    pmjay_dummy_cypher.csv",index=False)
"""Conclusion:
The 'pmjay_dummy_cypher.csv' contains the cypher query for each
    record of a patient. And since all the query for all the
    patient records could not be run simultaneously, the query
    needs to be given one ata a time. <br>
Given Google Colab runtime is Google Cloud, it cannot connect
    with Neo4j Desktop which runs on local IP, hence we move to
    anaconda jupyter environment whose runtime in the local
    system which matches with Neo4j Desktop runtime.
"""
```

A.8 8.py: Connecting to Neo4j Desktop

```
#Connecting to Neo4j Desktop and running the pmjay_dummy cypher
    script
from neo4j import GraphDatabase
uri = "bolt://localhost:7687"
user = "neo4j"
password = "12345678"
driver = GraphDatabase.driver(uri, auth=(user, password))
#Get cypher scripts from Local disk
import pandas as pd
cy_query= pd.read_csv("C:\\Users\\gurun\\Desktop\\Final_(
    Completed)\\pmjay_dummy_cypher.csv",)
with driver.session() as session:
    for rec in cy_query['0']:
        result = session.run(rec)
driver.close()
```

Appendix B: Neo4j Desktop Installation

B.1 Step-by-Step Guide

1. Download the Neo4j Desktop installer for Windows 11 from the official website: https://neo4j.com/download-center/#desktop. Choose the appropriate version based on your operating system architecture (32-bit or 64-bit).

2. Once the download is complete, locate the downloaded file in your downloads folder and double-click it to start the installation process.

3. You will be prompted to choose the installation directory. The default directory is usually fine, but you can choose a different directory if you prefer.

4. You will be asked to select the components you want to install. By default, all components are selected, but you can choose to install only specific components if you wish.

5. The installer will then start copying files and installing components. This may take a few minutes, depending on your system speed.

6. Once the installation is complete, you will be asked whether you want to start Neo4j Desktop immediately. If you select "Yes," Neo4j Desktop will start and prompt you to create a new project.

7. To create a new project, click the "Create a New Project" button and follow the prompts. You will need to give your project a name, select a database version, and choose a directory to store your project files.

8. Once you have created your project, you can start working with Neo4j. You can create a new database, import data, run queries, and much more.

Bibliography

[1] Accern. Named entity recognizition (ner) uses for unstructured text. "https://accern.com/blog/extracting- information-from-unstructured-text/".

[2] Bharat, A. Ayushman bharat - pradhan mantri jan arogya yojana (pm-jay), national health authority, government of india. "https://pmjay.gov.in/about/pmjay". "Accessed: 2022-12-06".

[3] Bharat, A. Ayushman bharat: An overview. national health authority, government of india. "https://www.nhp.gov.in/ayushman-bharat-yojana$_pg$". [*Online*, *Accessed* : $2022 - 11 - 16$].

[4] Bonatti, P. A., Decker, S., Polleres, A., and Presutti, V. Knowledge graphs: New directions for knowledge representation on the semantic web (dagstuhl seminar 18371). In *Dagstuhl reports* (2019), vol. 8, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[5] Google. Google colab. "https://research.google.com/colaboratory/faq.html: :text=Colaboratory%2C%20or%20%E2%80%9CColab%E2%80%9D%20for, learning%2C%20data%20analysis%20and%20education.".

[6] Guru99. Guru99 python tutorial. "https://www.guru99.com/python-tutorials.html".

[7] IBM, W. What is knowledge graph? "https://www.ibm.com/in-en/topics/knowledge-graph: :text=A%20knowledge%20graph%2C%20also%20known, the%20term%20knowledge%20%E2%80%9Cgraph.%E2%80%9D".

[8] Medium. Comparison of python nlp libraries. "https://medium.com/activewizards-machine-learning-company/comparison-of-top-6-python-nlp-libraries-c4ce160237eb".

[9] MOHFW. Pradhan mantri jan arogya yojana. ministry of health and family welfare, government of india. "https://www.mohfw.gov.in/pdf/MoHFW1YearofModi20050520.pdf". [Online,Accessed: 2022-10-15].

[10] Neo4j. Neo4j desktop. "https://neo4j.com/".

[11] NTU. Regular expressions. "https://www3.ntu.edu.sg/home/ehchua/ programming/howto/Regexe.html".

[12] of India, T. Pmjay. "https://timesofindia.indiatimes.com/topic/pmjay". [Online,Accessed: 2022-6-16].

[13] OpenAI. Openai. "https://platform.openai.com/docs/quickstart".

[14] Sitharaman, N. Union budget 2021-22. "https://www.indiabudget.gov.in/doc/Budget_Speech.pdf". [Online,Accessed: 2022-8-24].

[15] SpaCy. Natural language processing using spacy. "https://spacy.io/".

[16] SpaCy. Spacy api. "https://spacy.io/api".

[17] SpaCy. Spacy nlp model performance evaluation. "https://spacy.io/api/scorer".

[18] Talib, R., Hanif, M. K., Ayesha, S., and Fatima, F. Text mining: techniques, applications and issues. *International Journal of Advanced Computer Science and Applications 7*, 11 (2016).

[19] Tutorialspoint. Neo4j tutorial. "https://www.tutorialspoint.com/neo4j/index.htm".

[20] Tutorialspoint. Tutorialspoint: Python. "https://www.tutorialspoint.com/python/python_tutorial.pdf".

[21] Vidya, A. Building knowledge graph. "https://www.analyticsvidhya.com/blog/2019/10/how-to-build-knowledge-graph-text-using-spacy/".

[22] Wikipedia. Knowledge graph? "https://en.wikipedia.org/wiki/Knowledge_graph".