Assessing Impact of Government Policies on Vulnerable Populations

Updated Problem Formulation:

“Policymakers need to assess the impact of their policy on vulnerable populations because understanding the impact of policies is crucial for determining whether they effectively support vulnerable groups or exacerbate their challenges.”

For this, we are developing a chatbot to assess the impact of government policies on vulnerable populations, as it presents a critical need to address disparities and ensure equitable policy outcomes. The project aims to create a conversational system capable of collecting data on government initiatives, understanding their implementation, and assessing effects on vulnerable populations, supporting policymakers and the marginalized.

Finding and Extracting Government Policies:
We began by exploring the website https://www.myscheme.gov.in/ to analyze its layout and content. The site presented schemes categorized by state, each with a list of schemes accompanied by their respective URLs. However, it became apparent that the site couldn't be dynamically scraped. Consequently, we manually inspected each state's scheme list to gather raw HTML data, and we then utilized BeautifulSoup, a Python library, for web scraping. From the scheme URLs, we extracted detailed information such as scheme name, description, eligibility criteria, benefits, and application process. This data was meticulously compiled into a structured format and stored in a JSON file, ensuring accessibility for future analysis and reference.

Having the information of various policies, we then begin to get the reviews on these policies from news media or social media and then perform emotional mapping on these reviews to present them to the policy maker, to analyze the impact of their policy and make better implementations and changes.

Literature review:

1. Adaptive policies, policy analysis, and policy-making:

This paper advocates for an adaptive policy making approach that addresses uncertainties and dynamic conditions, proposing fundamental changes in analysis, policy types, and decision-making processes, with an illustrative example of national civil aviation policy.

Paper Link: Adaptive policies, policy analysis, and policy-making

2. A Government Decision Analytics Framework Based on Citizen Opinion

The paper introduces a Bayesian Predictive Model to analyze existing media reviews and public sentiment towards policies, aiding policymakers in predicting and understanding public responses to new policies before implementation.

Paper Link: A Government Decision Analytics Framework Based on Citizen Opinion

3. Policy Impact and Evaluation

The paper elucidates policy dimensions, categorizes impact types, assigns weights for evaluation, and proposes using relevance scores for ranking results in information retrieval systems based on policy variables.

Paper Link: Policy Impact and Evaluation

4. Web Scraping with Python, 3rd Edition by Ryan Mitchell

The book demystifies web scraping concepts, enabling users to parse complex HTML pages and extract desired information through simple automated programs, facilitating querying, data retrieval, and parsing from web servers.

Book Link: Web Scraping with Python, 3rd Edition by Ryan Mitchell

5. Context-based News Articles Retrieval using CLSM

This paper proposes a context-based analysis method utilizing the CLSM model to understand the contextual nuances of tweets/news regarding policy outcomes, extracting features from queries and documents through convolutional and max-pooling layers for comprehensive contextual understanding.

Paper Link: Context-based News Articles Retrieval using CLSM

Baseline Results

In this report, we outline the data collection process of the policies from the website https://www.myscheme.gov.in/. The objective was to gather information about various government schemes listed on the website, organize it by state and ministry, and compile the data into a structured format for further analysis.

Methodology:

1) Website Inspection: The website https://www.myscheme.gov.in/ was visited to understand its structure and content. We identified that the website categorizes schemes state-wise, providing a list of schemes along with their URLs.
2) Data Collection: State-wise inspection was conducted to gather data on schemes. Raw HTML data of the scheme lists and their URLs were copied for each state.
3) Data Scraping: BeautifulSoup, a Python library, was utilized for web scraping. Details of each scheme were extracted from the scheme URLs. Information such as scheme name, description, eligibility criteria, benefits, and application process were gathered.
4) Data Compilation: The scraped data was organized into a structured format. A JSON file was generated to store the compiled data, ensuring ease of access and future analysis.

Code Snippets and Explanation:

```
def get_scheme_list(self,path,state):
    filepath='./BaselineResults/RawHTMlfiles'+path
    with open(filepath,"rb") as f:
        data=f.read()

    soup=BeautifulSoup(data,'html.parser')

    target_div = soup.find_all('div', class_='flex flex-col')
    self.progress_bar(0,self.total)
    for divs in target_div:
        h2_element = divs.find('h2')
        a_element = divs.find('a')
        policyname=" ".join(a_element.text.replace('\r\n', ' ').split())
        ministry=" ".join(h2_element.text.replace('\r\n', '').split())
        scheme_description=self.get_scheme_details(a_element.get("href"))
        Dataset.policy_list.append({"Sno":Dataset.sno,"Policy name":policyname,"Description":scheme_description,
        "Ministry":ministry,"State":state})
        Dataset.sno+=1
        self.progress_bar(Dataset.sno,self.total)

    return
```

Using BeautifulSoup to extract the urls from raw HTML data copied from https://www.myscheme.gov.in/

```
def get_scheme_details(self,path):
    url="https://www.myscheme.gov.in"+path
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    rep=soup.find('div', id='details')
    details=rep.find('div').text

    rep2=soup.find('div', id='benefits')
    benefits=rep2.find('div').text

    rep3=soup.find('div', id='eligibility')
    eligibility=rep3.find('div').text

    return {"details":details,"benefits":benefits,
    "eligibility":eligibility}
```

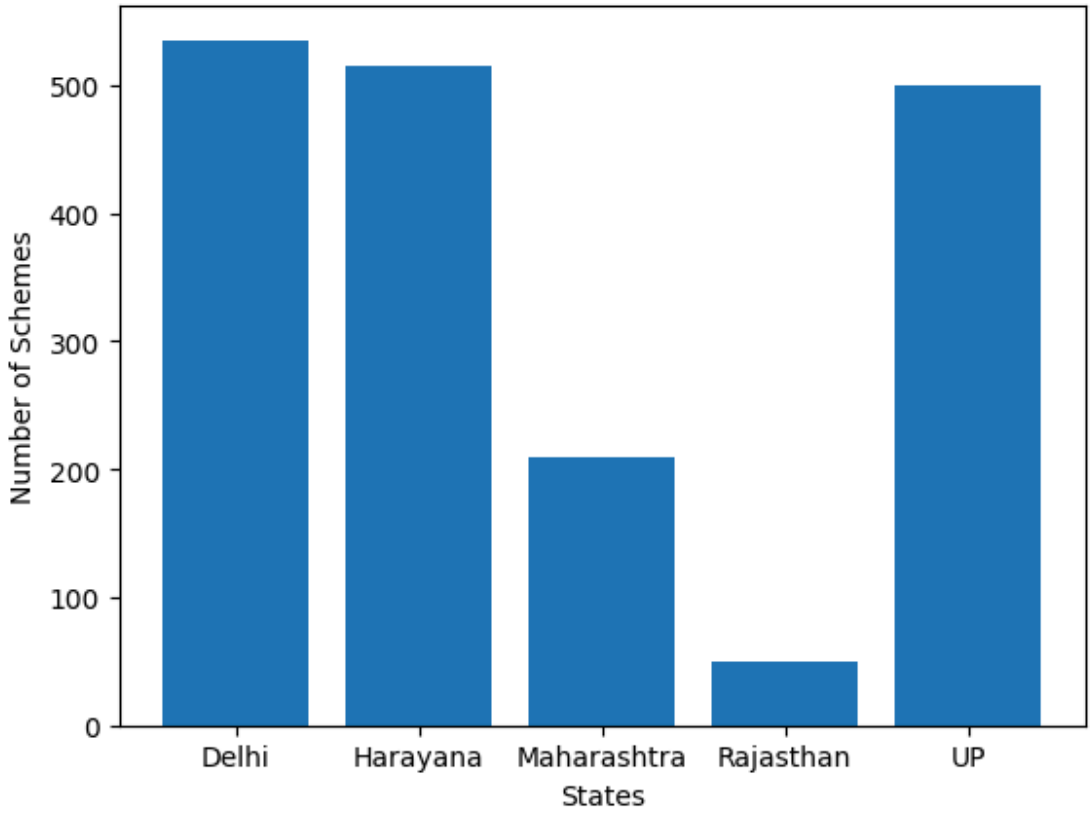
Using Beautifulsoup to extract description of the schemes from the extracted Urls

```
Total number of schemes to be compiled:  1810
Compiling Raw HTML files...
| |----------------------| 78.23%
```

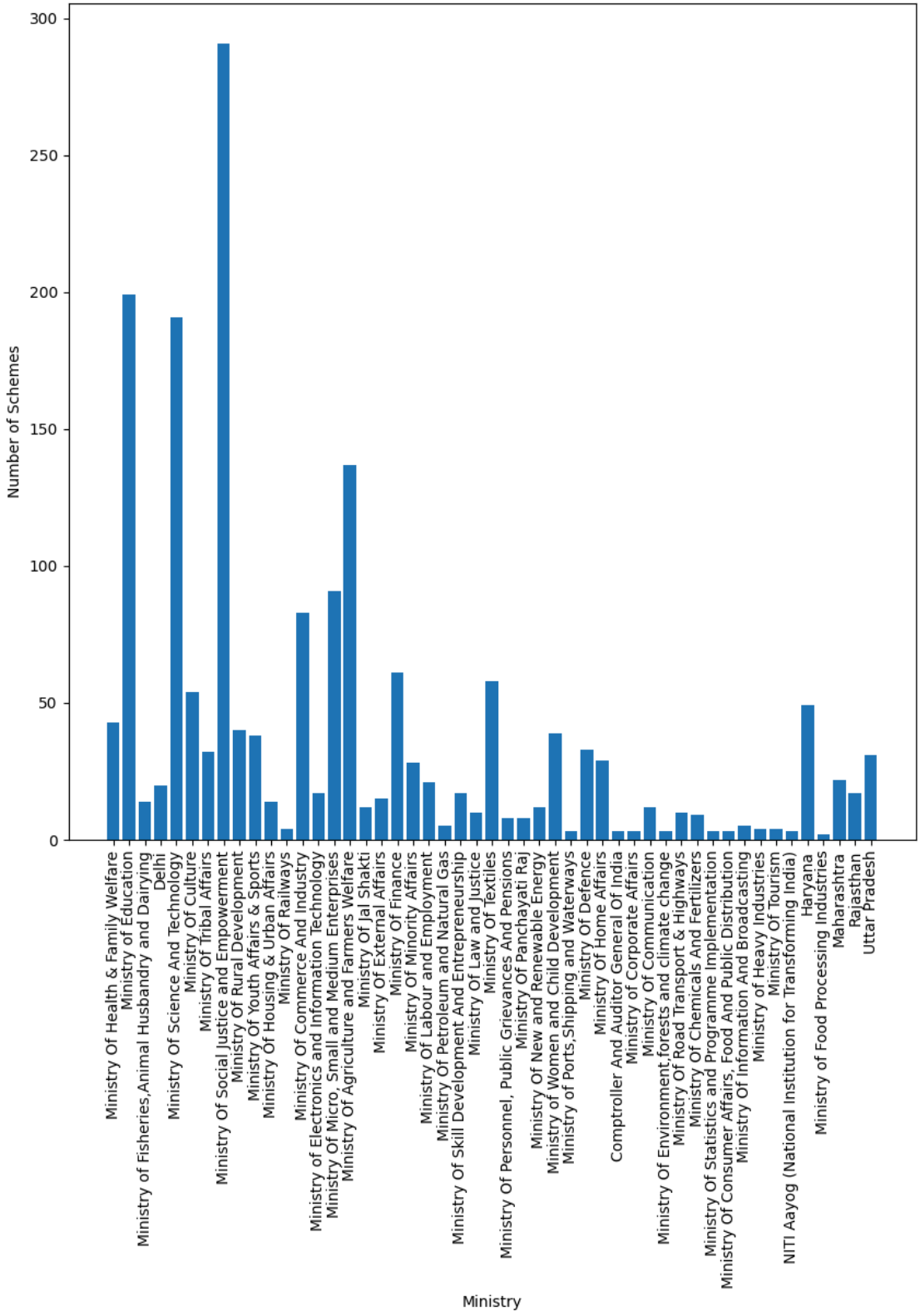
```
Total number of schemes to be compiled:  1810
Compiling Raw HTML files...
 |                                                                                | 100.06%
Compilation completed......
```

Data Analysis:

1. Graph of Number of Schemes vs State:
   - A bar graph illustrating the number of schemes available in each state was generated.
   - This graph provides insights into the distribution of schemes across different states.

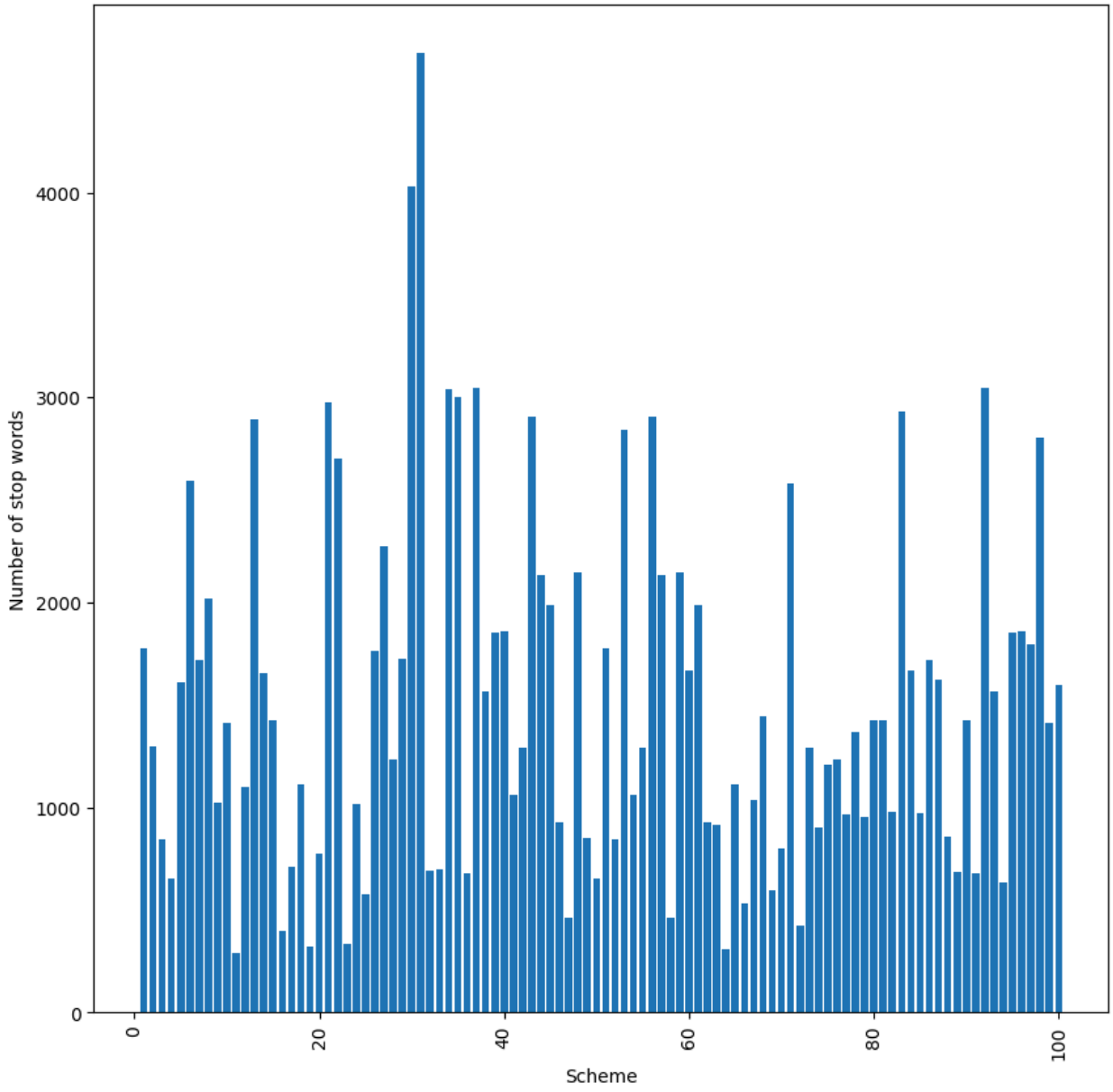


2. Graph of Number of Schemes vs Ministry:
   - A bar graph displaying the number of schemes under each ministry was created.
   - This graph helps in understanding the distribution of schemes across various ministries.



3. Graph of Number of Stopwords in Each Scheme:

- A histogram depicting the frequency distribution of stopwords in scheme descriptions was generated.
- This graph aids in identifying the prevalence of stopwords in scheme descriptions, which can inform further preprocessing steps.



Preprocessing Steps:

1. Lowercasing:
   - All words in the dataset were converted to lowercase to ensure consistency and facilitate subsequent processing steps.
   - lower() method was applied to convert all text to lowercase

2. Tokenization:
   - The text was tokenized to split it into individual words or tokens, which are essential for further analysis.
   - NLTK's word_tokenize() function was used to tokenize the text, splitting it into individual words or tokens.

3. Punctuation Removal:
   - Punctuation marks were eliminated from the text to clean and standardize the dataset.
   - String.punctuation method was applied to convert all text to lowercase

4. Stopword Removal:
   - NLTK provides a list of common stopwords for various languages. The stopwords corpus in NLTK was utilized to obtain a list of English stopwords.
   - NLTK's stopwords.words('english') function was used to retrieve the list of English stopwords.
   - Each word in the tokenized text was checked against the list of stopwords, and stop words were removed to focus on meaningful content.

5. Stemming:
   - Words were stemmed to reduce them to their root form, which helps in consolidating variations of words.
   - NLTK's Porter Stemmer algorithm was employed for stemming, reducing words to their root or base form.

6. Lemmatization:
   - Lemmatization was applied to further normalize words by reducing them to their base or dictionary form.
   - NLTK's WordNet Lemmatizer was used for lemmatization, which reduces words to their base or dictionary form.

```python
class TextPreprocessor:
    def __init__(self):
        # nltk.download('punkt')
        # nltk.download('stopwords')
        # nltk.download('wordnet')
        pass

    def lower_case(self, text):
        return text.lower()

    def tokenize(self, text):
        return word_tokenize(text)

    def remove_punctuation(self, tokens):
        return [token for token in tokens if token not in string.punctuation]

    def remove_stopwords(self, tokens):
        stop_words = set(stopwords.words('english'))
        return [token for token in tokens if token not in stop_words]

    def stem(self, tokens):
        stemmer = PorterStemmer()
        return [stemmer.stem(token) for token in tokens]

    def lemmatize(self, tokens):
        lemmatizer = WordNetLemmatizer()
        return [lemmatizer.lemmatize(token) for token in tokens]

    def preprocess_text(self, text):
        text = self.lower_case(text)
        tokens = self.tokenize(text)
        tokens = self.remove_punctuation(tokens)
        tokens = self.remove_stopwords(tokens)
        tokens = self.stem(tokens)
        tokens = self.lemmatize(tokens)
        return ' '.join(tokens)
```

Github: https://github.com/arpan21020/Information-Retrieval