CS552J\_DMDL\_2023\_Assessment\_2

# Introduction

This document contains a detailed report of the project, one is exploring the provided dataset of conversations between speakers from the British National Corpus, elaborating upon a few principles of data mining, and different components of deep learning techniques that can be used in text mining. It also contains investigation, modelling and report on insights from corpus data as the second task.

## Description of Data and Methods

The given dataset British National Corpus (BNC) 2014 spoken corpus data contains

* **Metadata:** Metadata defines the Subcorpora according to different features of the speakers (e.g. age) or of the recordings themselves (e.g. number of speakers in the conversation). We henceforth refer to the former type as ‘speaker metadata’ and the latter as ‘text metadata’.
* **Tagged text data** – Subcorpora of conversations with tagged data, the texts are tokenized, POS tagged and lemmatized with details of the speaker, signal overlaps and other significant meta details and text-processed details
* **Untagged text data** - Subcorpora of conversations with untagged data, details of speaker and their respective utterances in order with unprocessed texts in an utterance.

**Pre-processing the data:**

For this dataset, the following NLTK pre-processing techniques are used:

* Tokenization, which involves separating the text into individual words or tokens, deleting stop words
* Stemming or lemmatization to reduce the number of unique words, and dealing with missing values are examples of this.
* Furthermore, text data could also require encoding or embedding before it can be used by machine learning models. This can be accomplished through the use of one-hot encoding, word embeddings like word2vec, or pre-trained language models like BERT.

**Feature extraction and analysis approaches:**

Dataset given is explored by traversing through the metadata, traversing all the tagged and untagged files and age demographic, and utterance length features are extracted to get insights into the data prior to modelling. Basic counts such as the average utterance length of speakers in a conversation, the fraction of parts-of-speech tags, and the frequency of certain words or phrases are included, Classifying the frequency by age groups.

Clustering algorithms(K-means, DB Scan) can be used to group together similar utterances based on content or linguistic traits.

Additionally, Linguistic resources like named entity identification and sentiment analysis can also be employed to glean insights from text data.

**Common deep learning approaches for extracting information from textual data:** Text classification, sentiment analysis, language modelling, and summarization (Extractive and abstractive) are examples of common deep-learning algorithms for extracting information from textual data. Because of their ability to learn contextual links between words, transformers such as T5, BERT and GPT have grown in popularity for natural language processing tasks. However, one issue in adopting deep learning methodologies is the requirement for big datasets to adequately train these models. There are also trade-offs between model complexity and interpretability, as well as between training time and model accuracy.

**Evaluation of machine learning systems that create or classify text data:**

Metrics like Accuracy, precision, recall, and F1 score can be used to evaluate machine learning systems that generate or classify text data where the gold standard can vary. Human evaluators can also assess fluency and fidelity. Confusion matrices can be used to identify sections of data where the model is making mistakes and guide improvements. It is also critical to assess performance on different subsets of data to ensure that the model generalises well.

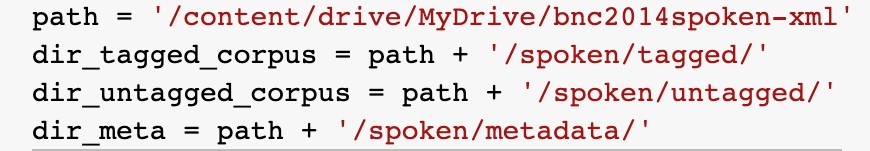
## Investigate, model, and report on insights from dialogue data

Before we tackle the individual tasks, required libraries such as os, pandas, numpy, altair, matplotlib, lxml are imported for visualization and parsing XML files.

The paths to the tagged and untagged directories of the corpus are defined and lists of file paths are created, to read the XML files (read\_xml() function) and store the required features in data frames.

The speakers metadata is stored in the dataframe – speakers

The texts metadata is stored in the dataframe – texts



### Statistics Summary and Texts summary

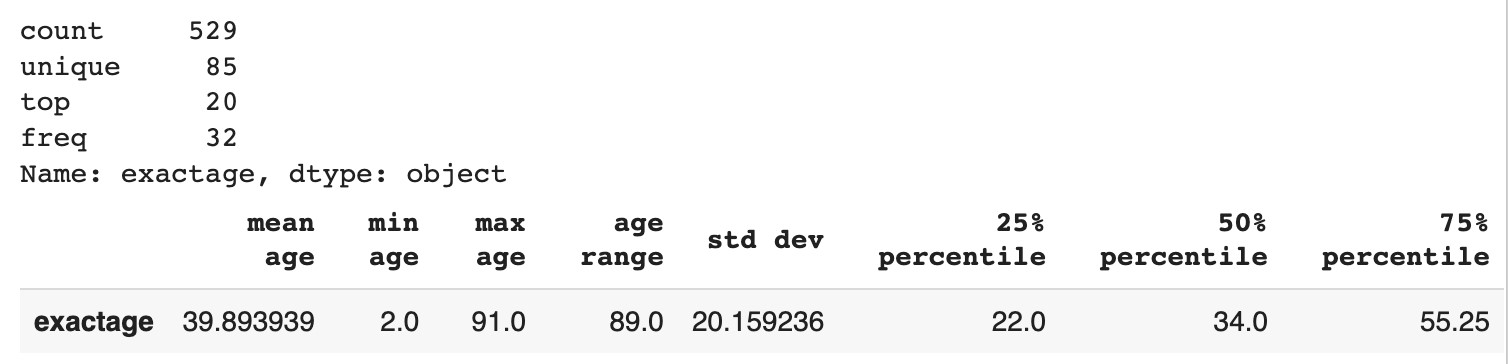
Summary statistics of this dataset are extracted, i.e. mean values, range, standard deviations, min/max values, median values and 25%/50%/75% percentile values.

Features of interest for these statistics are the speaker’s age demographics, statistics of the text data in utterance text, such as average utterance length per speaker, number of words spoken by speakers in the corpus and once extracted are saved to data frames.

#### Age demographics –

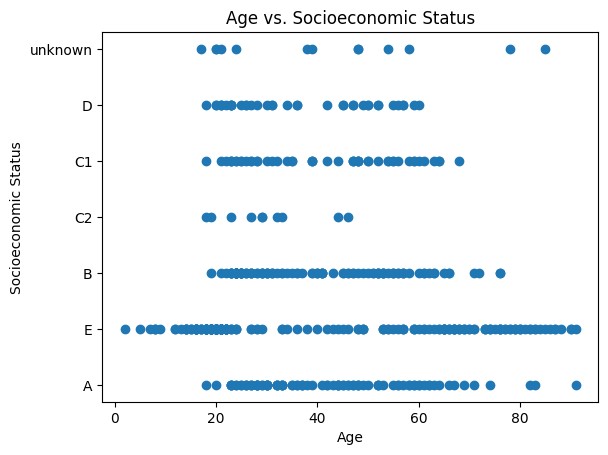
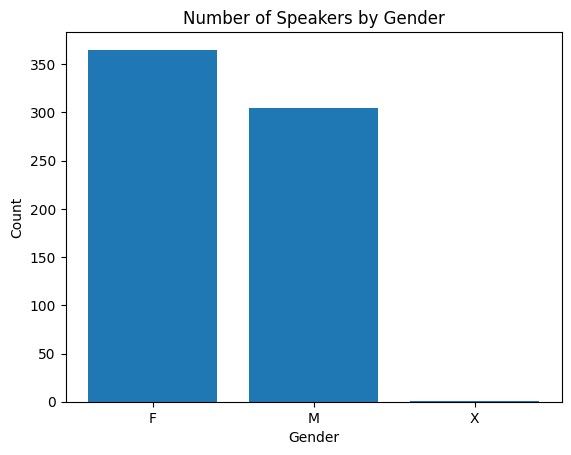
This code extracts the 'exactage' column from a Pandas DataFrame 'speakers', which presumably contains information about speakers in the dataset. It then computes summary statistics for this column using the Pandas `describe` function, which provides the count, unique, top and frequency I the data.

The code then coerces the 'exactage' column to a numeric data type, and computes additional statistics including mean age, minimum age, maximum age, age range, standard deviation, and percentile values using the Pandas `mean`, `min`, `max`, `std`, and `quantile` functions.



We also analyze the age demographics with respect to gender, socioeconomic grade, and birthplace.

First, summary statistics are computed for the exact age column and stored in the `age\_stats` variable. Then, the `exactage` column is converted to numeric and a new DataFrame is created containing only the `exactage`, `gender`, `socgrade`, and `birthplace` columns. Three graphs are generated from this data: a bar chart showing the number of speakers by gender, a scatter plot of age versus socioeconomic status, and a box plot of age by birthplace. These visualizations provide insights into the distribution of speakers in the dataset and the relationships between age, gender, socioeconomic status, and birthplace.





#### Statistics of average utterance lengths –

This code creates a dictionary of speaker data by iterating through dialogues and calculating the average length of each speaker's utterances. It then converts the dictionary to a DataFrame and saves it to a CSV file.

Next, it converts the 'exactage' column in the 'speakers' DataFrame to numeric and creates a color map for age groups based on the 'exactage' column. Finally, it creates a scatter plot of the number of utterances versus age, where each point is coloured based on the speaker's age group.

speaker n\_utterances

S0255 26.436364

S0259 26.953782

UNKFEMALE 2.625000

S0041 26.899396

S0046 23.894309

... ...

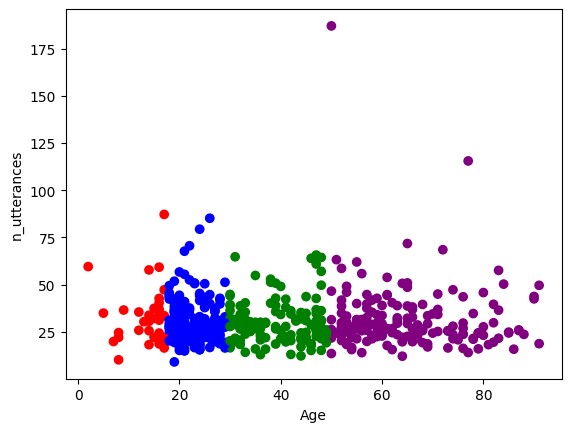
S0101 20.600000

S0102 20.465116

S0117 19.793651

S0156 14.181818

S0345 32.729839



### Data Visualization

### 

For Data visualization, we extract few other additional insights and plot them as follows:

#### 1) Frequency of words from tagged files, age groups

This code snippet processes tagged XML files containing spoken data, extracts individual words from each utterance, and counts the frequency of each word across all the files. The code starts by defining a directory path containing the tagged XML files and getting a list of all the XML files in the directory. It then loops through each XML file, extracts the words spoken by each speaker in each utterance, and updates a word frequency counter using the Counter class from the collections module. Finally, it prints the 10 most common words and their frequencies. This code can be used to analyze spoken data to gain insights into the vocabulary and usage patterns of the speakers.

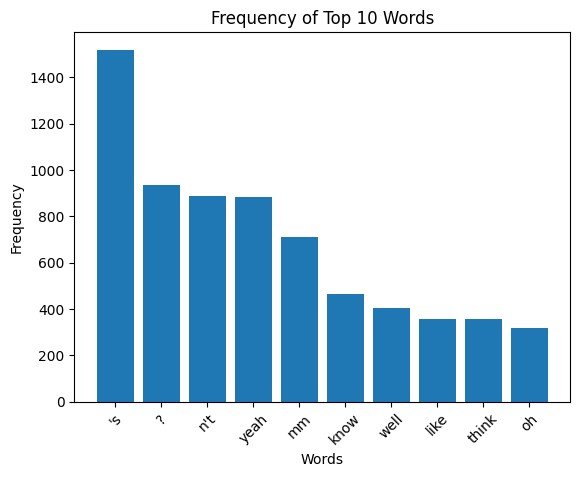
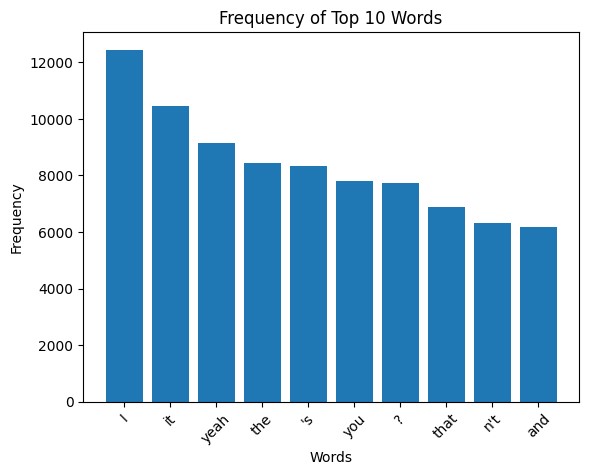
Frequency of words with stop counter

By downloading the stopwords from the NLTK library. We remove the stopwords in English (I, not and and etc..)

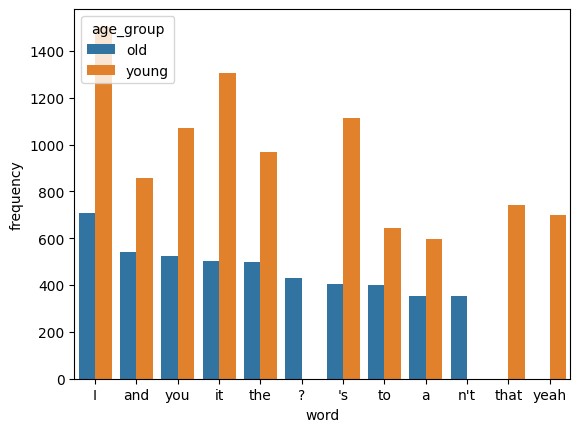
The code then loops over the tagged XML files, parses each file, and extracts the words spoken by each speaker while excluding stopwords. It updates a counter to keep track of the frequency of each word and gives the most used common 10 words excluding the stopwords

|  |  |
| --- | --- |
| Without nltk stopping: | With NLTK stopping: |
| I: 2211 it: 1811 you: 1596 's: 1518 the: 1465 and: 1399 to: 1047 that: 1047 a: 949 ?: 937 | 's: 1518 ?: 937 n't: 888 yeah: 885 mm: 710 know: 464 well: 405 like: 359 think: 357  oh: 317 |

Plotting the above frequency



We then group by age categories of old and young and plot the frequencies in age demographics as shown below.



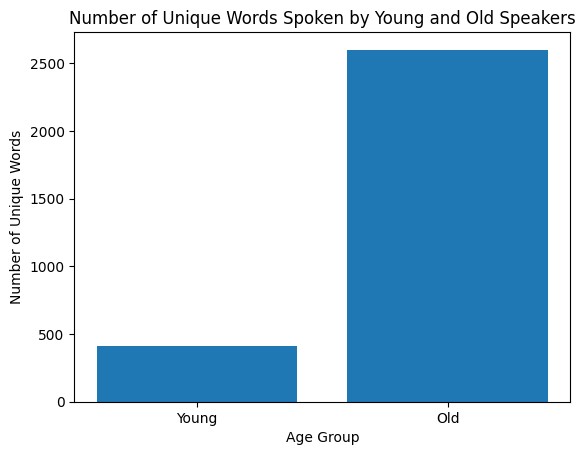
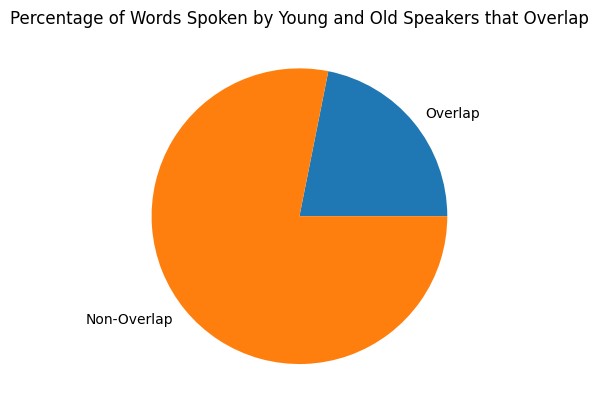
#### 2) Vocabulary overlap frequency

This code analyzes dialogues between multiple speakers and performs several tasks. Firstly, it stores the words spoken by each speaker in a dictionary where each speaker is a key, and the set of words spoken by that speaker is the value. It then checks if there is any overlap between the sets of words spoken by different speakers. If an overlap exists, it prints a message saying that there is vocabulary overlap between the speakers; otherwise, it prints a message saying that there is no vocabulary overlap between the speakers. The speaker\_words dictionary is merged with a speakers dataframe, and speakers are grouped by age into two categories: young (age < 50) and old (age >= 50). Finally, it calculates the vocabulary statistics for each age group, including the number of unique words spoken by young speakers, the number of unique words spoken by old speakers, and the number of words spoken by both young and old speakers.

Number of unique words spoken by young speakers: 414

Number of unique words spoken by old speakers: 2601

Number of words spoken by both young and old speakers: 844



Overlap between speakers 4

{"'re", 'there', 'when', 'eat', 'in', 'she', 'say', 'no', 'be', 'er', "'m", 'on', 'yeah', 'I', 'you', "'ll", 'well', 'what',

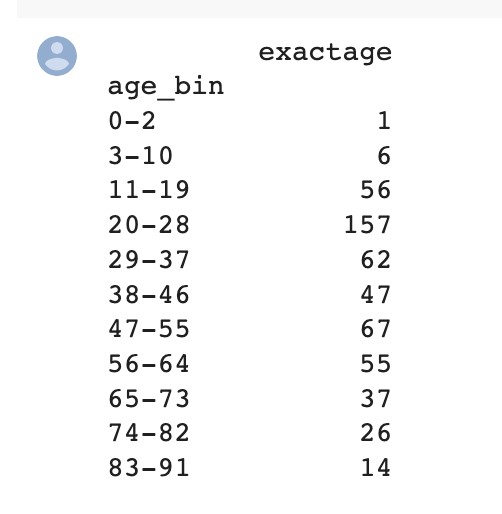
'something', '?', '--ANONnameF', "'s", 'a'}

There is vocabulary overlap between the speakers.

### Summarization Techniques

#### Age – bins creation

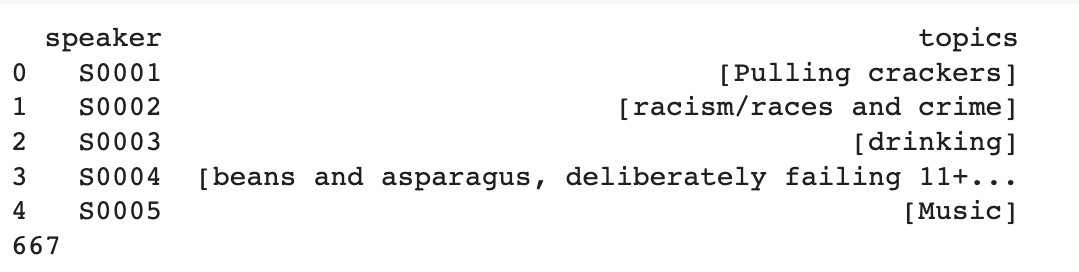
The age bins code creates age range categories from the 'exactage' column in the 'speakers' dataframe using the `pd.cut()` method. The `bins` variable defines the bin edges as 10 evenly spaced values between the minimum and maximum ages in the 'exactage' column. The `labels` parameter creates the bin labels as age ranges. A new column 'age\_bin' is added to the 'speakers' dataframe with the bin labels.



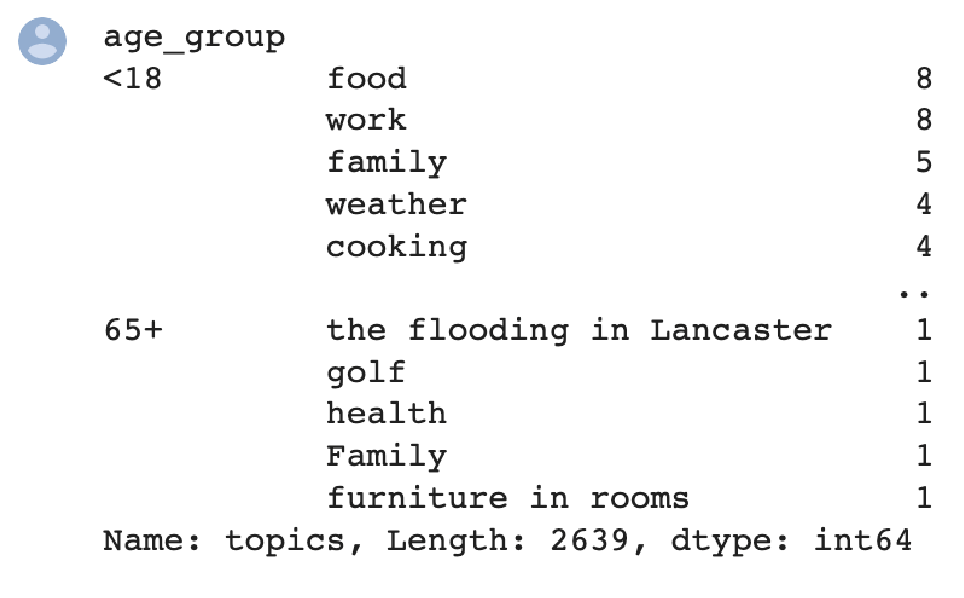
#### Topics summaries

The code above extracts insights on summaries of topics in dialogues between younger and older speakers. It starts by splitting the list of speakers into individual speaker names. Then, it splits the topics by comma and creates a new row for each topic-speaker combination. The code creates a new dataframe by merging the speakers and topics dataframes. Finally, the dataframe is grouped by speaker names and the topics are aggregated into a list. The resulting dataframe contains information on the topics discussed by each speaker, which can be used to gain insights into the types of topics that are more prevalent among younger or older speakers.

Extarcting Topics of specific speaker

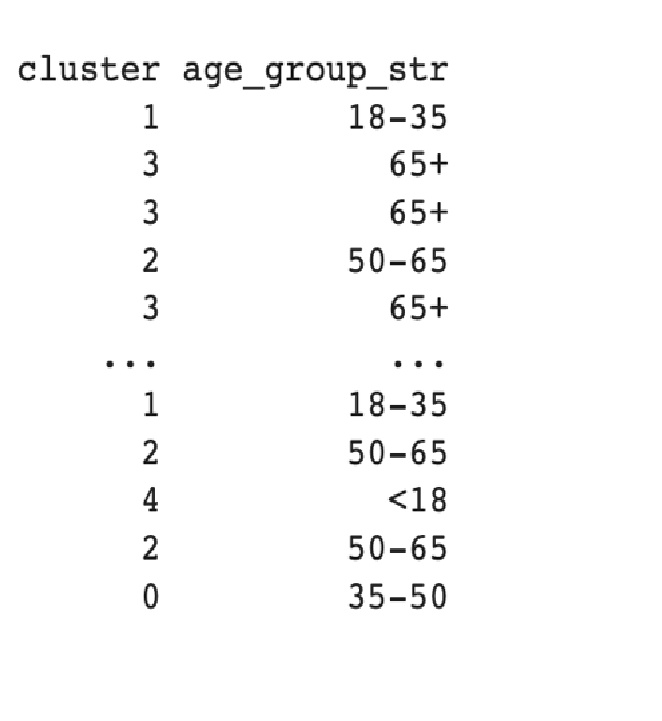
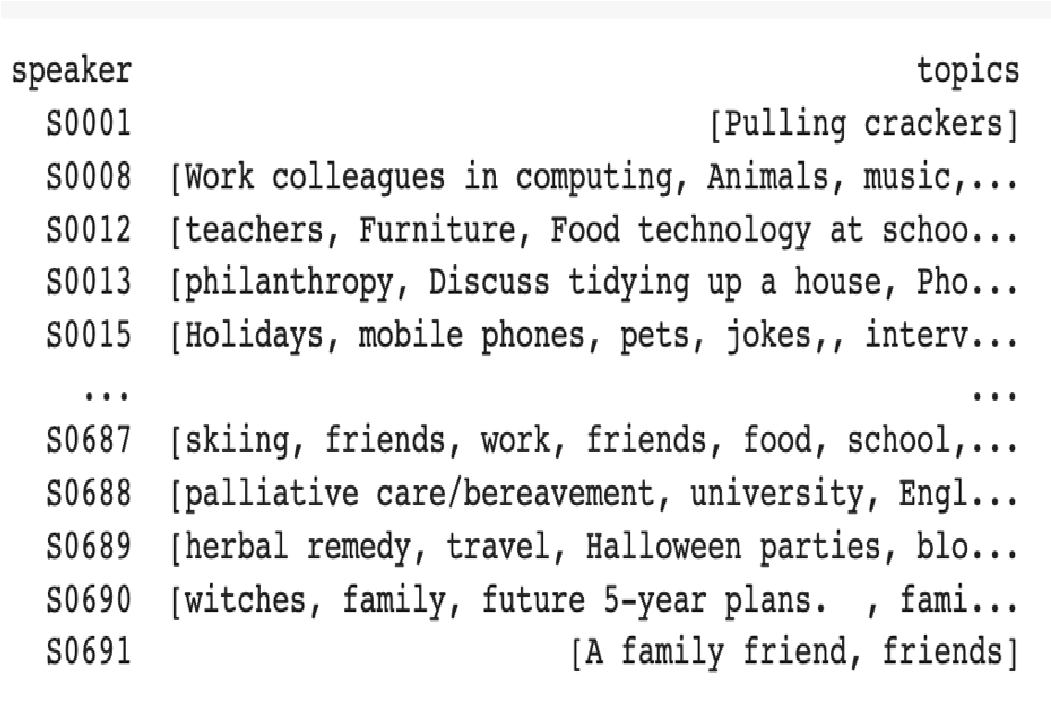


Clustering the speakers as per age and topics



#### Clustering using algorithms

Clustering utilizes scikit-learn to perform topic clustering on speaker data. It starts by splitting the speakers' names and topics into separate dataframes and then merges them together. The resulting dataframe is grouped by speaker names, aggregating their topics into lists. The age information is merged and processed, converting it to numeric and creating age groups. Next, the topics are vectorized using TF-IDF, and K-means clustering is applied with five clusters. Finally, the cluster labels are added to the dataframe for further analysis or visualization.



**Additional Summarization techniques –**

#### Extractive summary

Extractive summarization involves selecting the most important sentences or phrases from the original text and presenting them as a summary. This approach can be relatively straightforward and effective, but it can also result in summaries that are disjointed or difficult to read.

*Extractive Summary:* okay --UNCLEARWORD I yeah 's I come 's way I go yeah I 'm always like ooh yeah 's --ANONnameM 's car I 'm like whose black car ? whole pram yeah I n't seen 's fat 's -ANONnameF talking earlier n't go --UNCLEARWORD oh 's cute see 's thing like -UNCLEARWORD base bit ?

#### Abstractive summary

Abstractive summarization, on the other hand, involves generating new text that summarizes the original content in a more natural and readable way. This approach is more complex and requires natural language generation (NLG) techniques, such as neural machine translation or language modeling.

The code uses the T5 model and tokenizer from the Transformers library to generate a summary of the input text. The input text is tokenized and passed to the model to generate a summary, which is then printed.

*Abstarctive Summary:* I'm like ooh yeah's --ANONnameM's car I'm like ooh yeah's --

UNCLEARWORD oh yeah's I come's way I go yeah I'm like ooh yeah's --ANONnameF talking earliern't go --UNCLEARWORD oh yeah's I come's way I go yeah I'm like ooh yeah's -ANONnameM's car

### Train a logistic classifier using the features extracted from the subtask1

#### 1) Logistic regression classifier to predict age based on average utterance length

The code trains and evaluates a logistic regression classifier to predict age based on the features "age" and "avg\_utterance\_length". The dataset is split into training and test sets using a 80/20 ratio. The most informative features are reported, and the model is evaluated using several metrics, including accuracy, precision, and recall. The R2 score is used as the evaluation metric. The results are printed to the console.

|  |  |  |
| --- | --- | --- |
|  | **Feature** | **Coefficient** |
| **1** | **avg\_utterance\_length** | **0.343698** |
| **0** | **age** | **0.599704** |

**Accuracy: 0.7538823056468924**

#### 2) Random forest classifier to predict age based on average utterance length

We are training the same dataset mentioned above using a more complex classifier and it results in improved accuracy

We generate a confusion matrix as well

#### Accuracy: 0.8903 F1 score: 0.7430

**Precision: 0.8881 Recall: 0.6903 Confusion matrix:**

**[[ 48 0 0 0] [ 0 32 0 0] [ 0 0 37 0] [ 0 0 0 43]]**

#### 3) Logistic regression model for vocabulary overlap prediction based on age

This code performs text classification using a logistic regression classifier. The dataset is split into training and testing data, with 80% of the data being used for training. The `CountVectorizer` from scikit-learn is used to vectorize the words. The `stop\_words` parameter is set to "english", indicating that stop words such as "the", "a", and "an" should be removed from the text. The age group labels are encoded as 1 if the age group is "old", and 0 otherwise. The logistic regression classifier is then trained using the training data, and the performance of the classifier is evaluated on the testing data using accuracy score.

The accuracy of the logistic regression classifier is printed to the console using f-strings. The performance of the classifier can be further improved by tuning hyperparameters of the logistic regression model, such as the regularization parameter, or by using more advanced machine learning algorithms, such as support vector machines or deep neural networks. Additionally, exploring other vectorization techniques such as TF-IDF may help improve the classification performance.

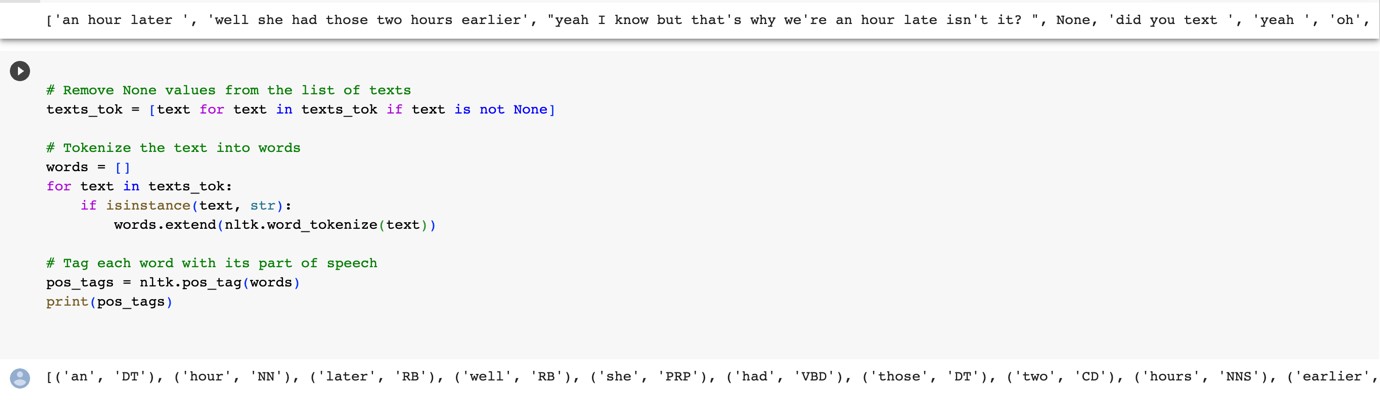
**Accuracy: 0.678056927**

**Bonus Optional Tasks**

#### Data- preprocessing the untagged xml files

This code loads untagged XML files from a directory, extracts the text content, tokenizes it into words, and creates a frequency distribution of the words. It uses the NLTK library for part-of-speech tagging, named entity recognition, and collocation finding. It also defines a regular expression for detecting named entities, creates a chunk parser using the regular expression, and performs chunking and named entity recognition on the text. The code prints the most common words and collocations using the chi-squared measure. It uses the ElementTree module to load the XML files and stores the extracted text in a list.

#### 1) Tokenize the texts



#### 2) Part of speech chunking and Named entity recognition



#### 3) Collocations finder

