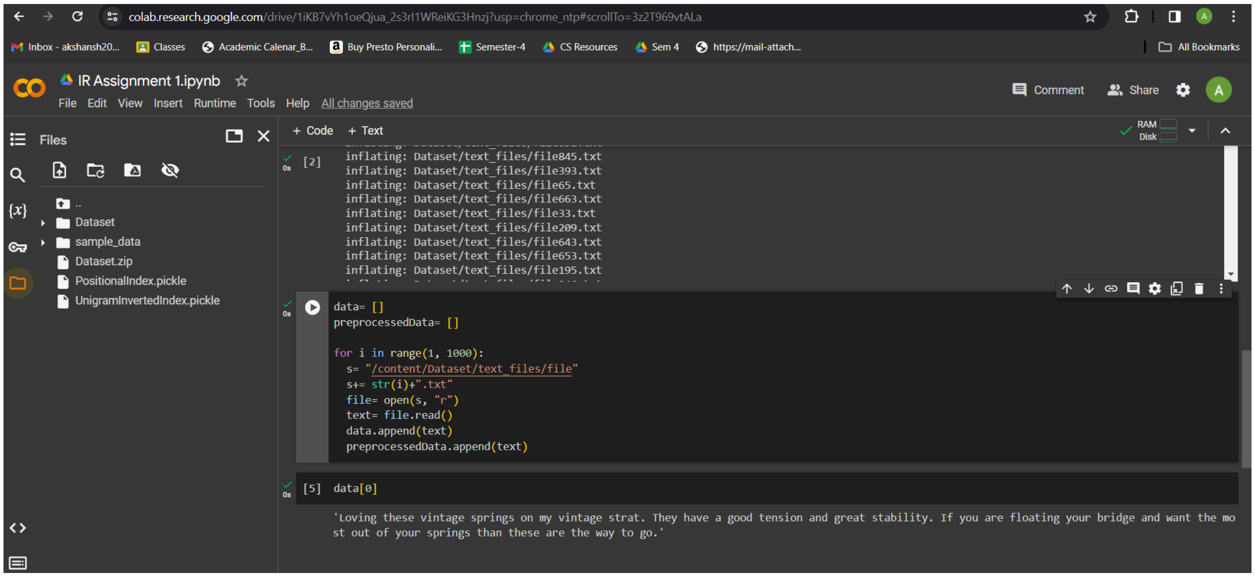
IR Assignment - 1

Akshansh Jaiswal (2020018)

Data Extraction

First step for the assignment was to extract and accumulate the data together from 999 different files.
To do the same, I uploaded a zip folder in the colab environment and extracted it. After extraction, I read the files and stored them in a variable as a list of strings for easier access.



```
[2]
     inflating: Dataset/text_files/file845.txt
     inflating: Dataset/text_files/file393.txt
     inflating: Dataset/text_files/file65.txt
     inflating: Dataset/text_files/file663.txt
     inflating: Dataset/text_files/file33.txt
     inflating: Dataset/text_files/file209.txt
     inflating: Dataset/text_files/file643.txt
     inflating: Dataset/text_files/file653.txt
     inflating: Dataset/text_files/file195.txt
```

```
data= []
preprocessedData= []

for i in range(1, 1000):
  s= "/content/Dataset/text_files/file"
  s+= str(i)+".txt"
  file= open(s, "r")
  text= file.read()
  data.append(text)
  preprocessedData.append(text)
```

```
[5] data[0]

'Loving these vintage springs on my vintage strat. They have a good tension and great stability. If you are floating your bridge and want the mo
st out of your springs than these are the way to go.'
```

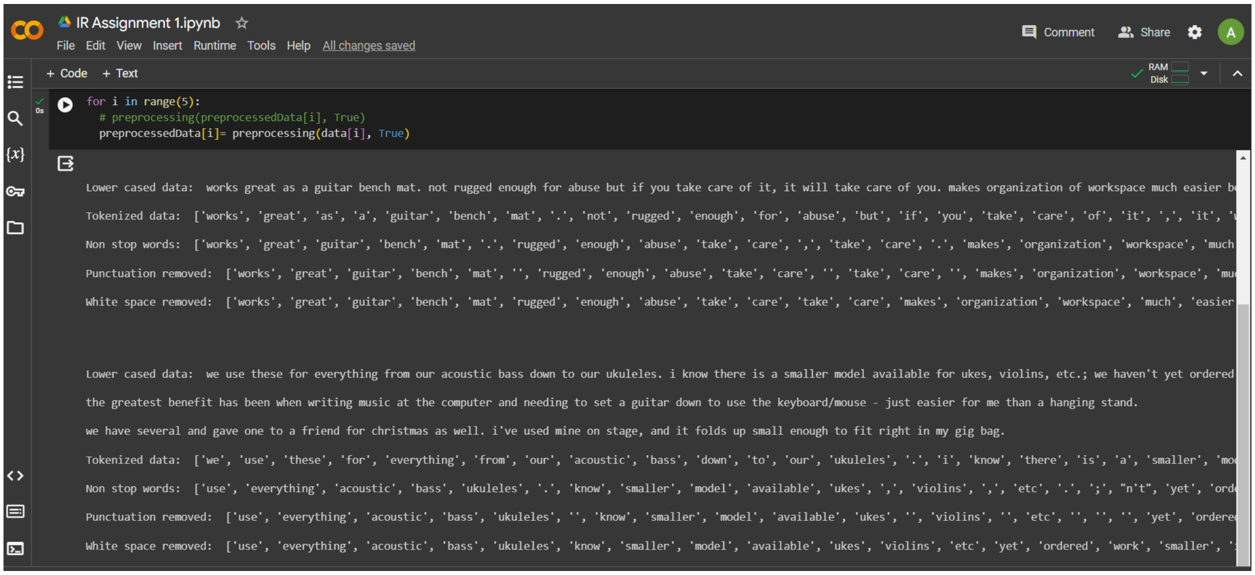
Data Preprocessing

Following data extraction, the next step was to pre-process the data. It comprised of 5 steps namely:

- Lowercase text
- Performing tokenization
- Stopwords removal
- Punctuation removal
- Removal of white space

Each of the 999 files were preprocessed with each step.
For converting to lowercase, '.lower()' function was used. For tokenization, I used the *word_tokenize()* function from the **nltk** library. After tokenization, I imported the set of stop words and performed their removal from the tokens.

Following the removal of stop words, punctuation had to be removed, the same was achieved with use of *'string.punctuation'.* Lastly, to remove white spaces, a for loop was used to iterate through the list of remaining tokens and remove white spaces.



```
for i in range(5):
  # preprocessing(preprocessedData[i], True)
  preprocessedData[i]= preprocessing(data[i], True)
```

```
Lower cased data:  works great as a guitar bench mat. not rugged enough for abuse but if you take care of it, it will take care of you. makes organization of workspace much easier b
Tokenized data:  ['works', 'great', 'as', 'a', 'guitar', 'bench', 'mat', '.', 'not', 'rugged', 'enough', 'for', 'abuse', 'but', 'if', 'you', 'take', 'care', 'of', 'it', ',', 'it', '
Non stop words:  ['works', 'great', 'guitar', 'bench', 'mat', '.', 'rugged', 'enough', 'abuse', 'take', 'care', ',', 'take', 'care', '.', 'makes', 'organization', 'workspace', 'much
Punctuation removed:  ['works', 'great', 'guitar', 'bench', 'mat', '', 'rugged', 'enough', 'abuse', 'take', 'care', '', 'take', 'care', '', 'makes', 'organization', 'workspace', 'mu
White space removed:  ['works', 'great', 'guitar', 'bench', 'mat', 'rugged', 'enough', 'abuse', 'take', 'care', 'take', 'care', 'makes', 'organization', 'workspace', 'much', 'easier


Lower cased data:  we use these for everything from our acoustic bass down to our ukuleles. i know there is a smaller model available for ukes, violins, etc.; we haven't yet ordered
the greatest benefit has been when writing music at the computer and needing to set a guitar down to use the keyboard/mouse - just easier for me than a hanging stand.
we have several and gave one to a friend for christmas as well. i've used mine on stage, and it folds up small enough to fit right in my gig bag.
Tokenized data:  ['we', 'use', 'these', 'for', 'everything', 'from', 'our', 'acoustic', 'bass', 'down', 'to', 'our', 'ukuleles', '.', 'i', 'know', 'there', 'is', 'a', 'smaller', 'mo
Non stop words:  ['use', 'everything', 'acoustic', 'bass', 'ukuleles', '.', 'know', 'smaller', 'model', 'available', 'ukes', ',', 'violins', ',', 'etc', '.', ';', "n't", 'yet', 'ord
Punctuation removed:  ['use', 'everything', 'acoustic', 'bass', 'ukuleles', '', 'know', 'smaller', 'model', 'available', 'ukes', '', 'violins', '', 'etc', '', '', '', 'yet', 'ordere
White space removed:  ['use', 'everything', 'acoustic', 'bass', 'ukuleles', 'know', 'smaller', 'model', 'available', 'ukes', 'violins', 'etc', 'yet', 'ordered', 'work', 'smaller', '
```

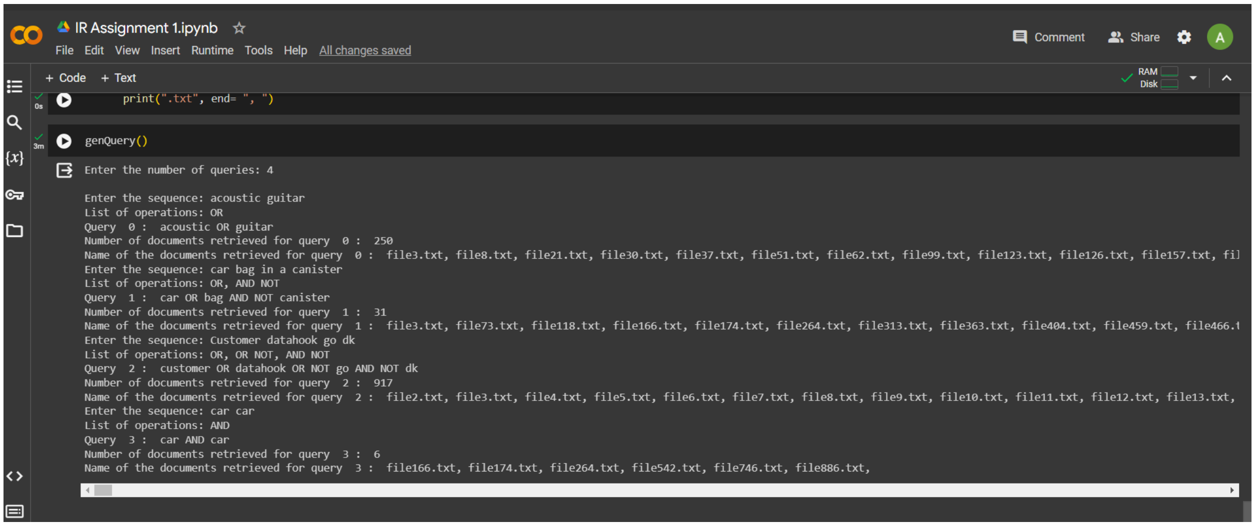
Unigram Inverted Index

For storing Unigram Indices, I used a dictionary. To do the complete computation I iterated over all the 999 files and stored the occurrences of different words with corresponding file names. The words acted as keys in the dictionary and it corresponded to a list of file names containing the word.

uii=   {word1: [file1, file2, …]
word2: [file3, file4, …]
…}

Boolean Queries

Following the computation of *Unigram Inverted Index* boolean queries were written. The queries supported AND, OR, AND NOT, OR NOT
The input was also preprocessed before performing the computation.
Each had a separate function to be called whenever required. A separate function was written to accommodate for complex queries allowing as boolean queries as required in a sequence. The output to such a query would be the number of matches followed by a list of file names.



```
        print(".txt", end= ", ")
```

```
genQuery()
```

```
Enter the number of queries: 4

Enter the sequence: acoustic guitar
List of operations: OR
Query  0 :  acoustic OR guitar
Number of documents retrieved for query  0 :  250
Name of the documents retrieved for query  0 :  file3.txt, file8.txt, file21.txt, file30.txt, file37.txt, file51.txt, file62.txt, file99.txt, file123.txt, file126.txt, file157.txt, fil
Enter the sequence: car bag in a canister
List of operations: OR, AND NOT
Query  1 :  car OR bag AND NOT canister
Number of documents retrieved for query  1 :  31
Name of the documents retrieved for query  1 :  file3.txt, file73.txt, file118.txt, file166.txt, file174.txt, file264.txt, file313.txt, file363.txt, file404.txt, file459.txt, file466.t
Enter the sequence: Customer datahook go dk
List of operations: OR, OR NOT, AND NOT
Query  2 :  customer OR datahook OR NOT go AND NOT dk
Number of documents retrieved for query  2 :  917
Name of the documents retrieved for query  2 :  file2.txt, file3.txt, file4.txt, file5.txt, file6.txt, file7.txt, file8.txt, file9.txt, file10.txt, file11.txt, file12.txt, file13.txt,
Enter the sequence: car car
List of operations: AND
Query  3 :  car AND car
Number of documents retrieved for query  3 :  6
Name of the documents retrieved for query  3 :  file166.txt, file174.txt, file264.txt, file542.txt, file746.txt, file886.txt,
```

Positional Index

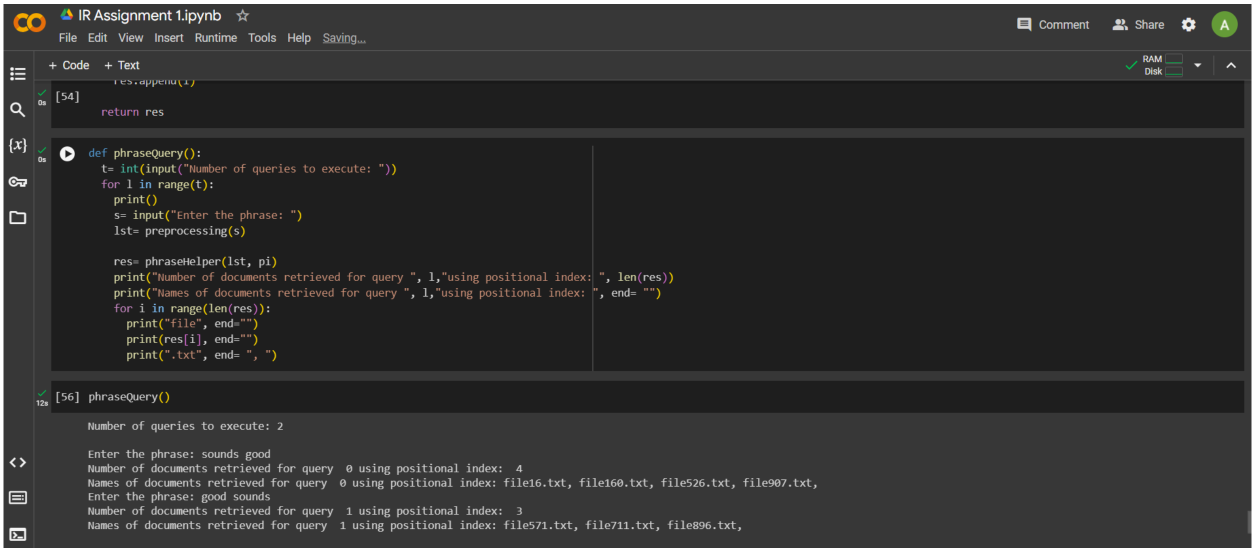
Again to store positional information, a dictionary was used. I iterated through the whole data and stored it in a dictionary with tokens as keys. The keys corresponded to different dictionaries, which had file number as the key and the list of locations of the token as the corresponding values.

```
pi= {word1:   {file1: [location1, location2, …]
               file2: [location3, location4, …]
               …}
     word2:   {file3: [location5, location6, …]
               file4: [location7, location8, …]
               …}
     …}
```

Phrase Queries

To provide the feature of phrase queries, the input was taken and preprocessed. Then using *positional index* information I found files with the same terms in similar order in proximity.

Finally as output, the number of files and their names were printed out.



```
[54]
      return res
```

```
def phraseQuery():
  t= int(input("Number of queries to execute: "))
  for l in range(t):
    print()
    s= input("Enter the phrase: ")
    lst= preprocessing(s)

    res= phraseHelper(lst, pi)
    print("Number of documents retrieved for query ", l,"using positional index: ", len(res))
    print("Names of documents retrieved for query ", l,"using positional index: ", end= "")
    for i in range(len(res)):
      print("file", end="")
      print(res[i], end="")
      print(".txt", end= ", ")
```

```
[56] phraseQuery()
```

```
Number of queries to execute: 2

Enter the phrase: sounds good
Number of documents retrieved for query  0 using positional index:  4
Names of documents retrieved for query  0 using positional index: file16.txt, file160.txt, file526.txt, file907.txt,
Enter the phrase: good sounds
Number of documents retrieved for query  1 using positional index:  3
Names of documents retrieved for query  1 using positional index: file571.txt, file711.txt, file896.txt,
```