Configuration Manual

MSc Research Project
Data Analytics

Priyanka Ashok Sujgure
Student ID: X20136706

School of Computing
National College of Ireland

Supervisor: Prof. Christian Horn

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Priyanka Ashok Sujgure
Student ID:	X20136706
Programme:	Data Analytics
Year:	2020-2021
Module:	MSc Research Project
Supervisor:	Prof. Christian Horn
Submission Due Date:	16th Dec 2021
Project Title:	Configuration Manual
Word Count:	916
Page Count:	14

Signature: Priyanka Ashok Sujgure

Date: 16th Dec 2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Priyanka Ashok Sujgure
Student ID: X20136706

1 Introduction

This document consists of a detailed description of all the hardware, software requirements and the code used to implement the “Automatic question generator using spaCy”.

Note: As the data is on cloud to execute the code simply a run all will execute the entire code.

2 System configuration

2.1 Hardware

- Processor: Intel(R) Core (TM) i5-10210U CPU @ 1.60GHz   2.11 GHz
- RAM: 8.00 GB (25.51GB GPU , TPU available on Google Colab Pro)
- System type: 64-bit operating system, x64-based processor
- Hard Disk Storage: 100GB (Google Drive Storage)

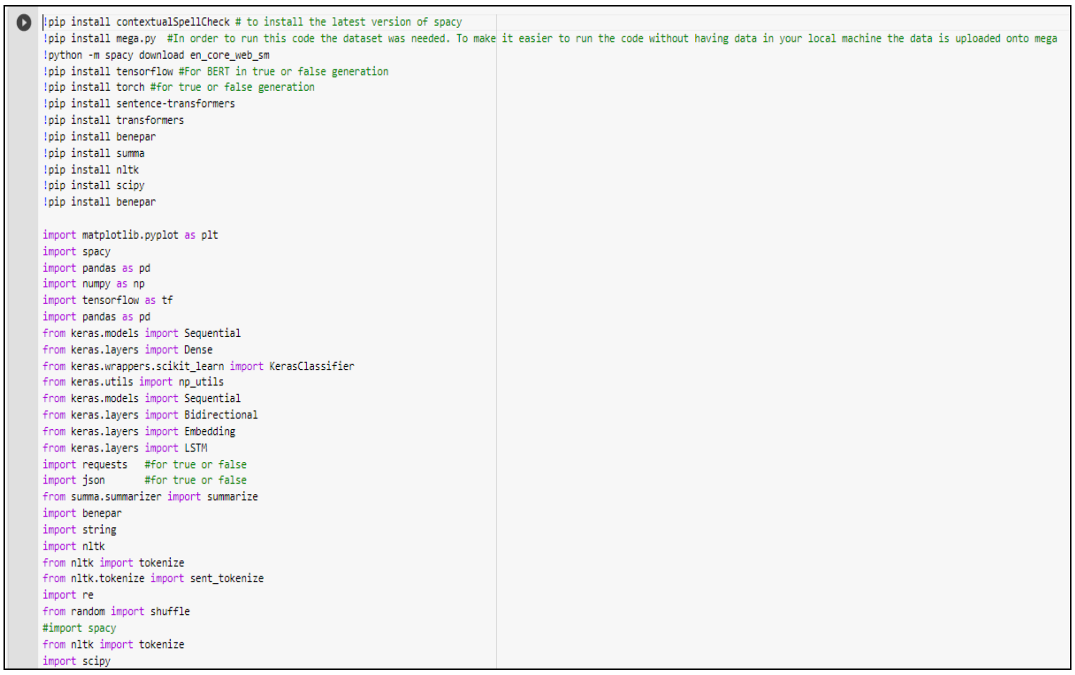
2.2 Software

- Software Computing Tools Used: Python 3 Jupyter Notebook (Google Colab), Overleaf, Microsoft Excel, DB Browser, R.
- Browser Engine: Google Chrome/ Firefox
- Email: Gmail login to access Colab Pro.

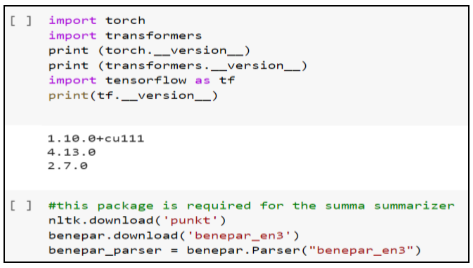
3 Project Development

As a start, some of the basic libraries needed for spaCy, and other NLP functions have been installed.

Among these tools and libraries are: spaCy, pandas, NLTK,etc.



```
!pip install contextualSpellCheck # to install the latest version of spacy
!pip install mega.py  #In order to run this code the dataset was needed. To make it easier to run the code without having data in your local machine the data is uploaded onto mega
!python -m spacy download en_core_web_sm
!pip install tensorflow #For BERT in true or false generation
!pip install torch #for true or false generation
!pip install sentence-transformers
!pip install transformers
!pip install benepar
!pip install summa
!pip install nltk
!pip install scipy
!pip install benepar

import matplotlib.pyplot as plt
import spacy
import pandas as pd
import numpy as np
import tensorflow as tf
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Bidirectional
from keras.layers import Embedding
from keras.layers import LSTM
import requests   #for true or false
import json       #for true or false
from summa.summarizer import summarize
import benepar
import string
import nltk
from nltk import tokenize
from nltk.tokenize import sent_tokenize
import re
from random import shuffle
#import spacy
from nltk import tokenize
import scipy
```

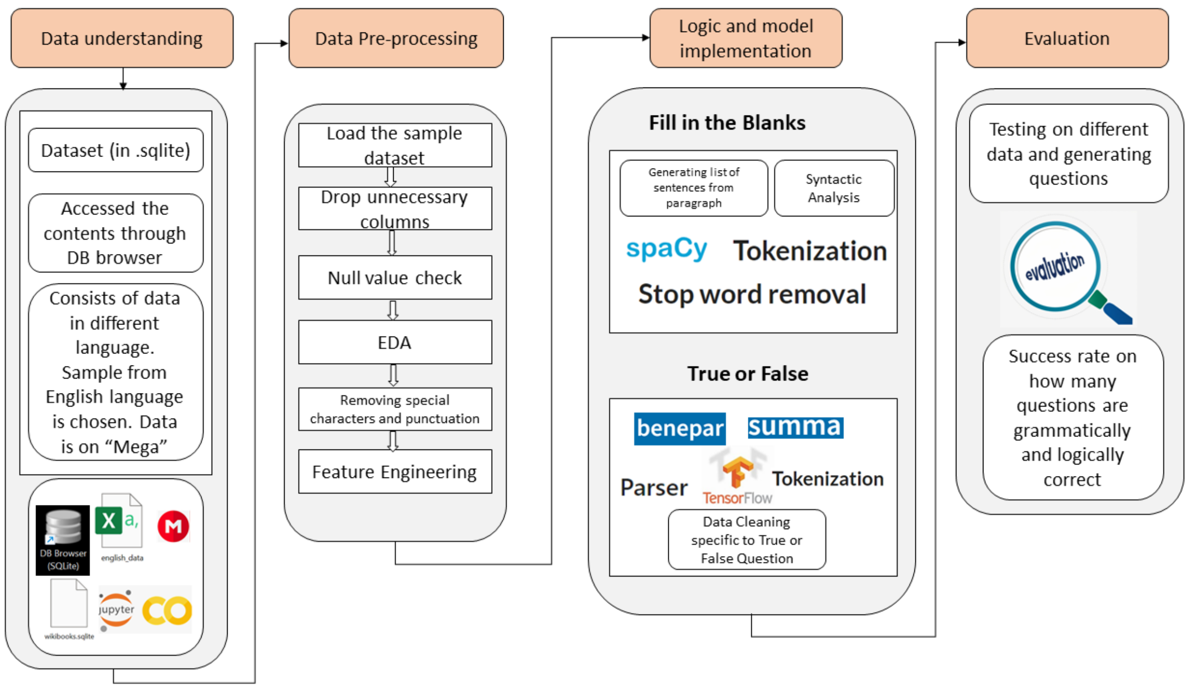


```
[ ]  import torch
     import transformers
     print (torch.__version__)
     print (transformers.__version__)
     import tensorflow as tf
     print(tf.__version__)

     1.10.0+cu111
     4.13.0
     2.7.0

[ ]  #this package is required for the summa summarizer
     nltk.download('punkt')
     benepar.download('benepar_en3')
     benepar_parser = benepar.Parser("benepar_en3")
```

Figure 1: All the libraries imported at the start of the code

3.1 Design flow

As mentioned, perform major steps in Design process Stage 1: Data understanding Stage 2: Data Pre-Processing Stage 3: Building logic and models implementation Stage 4: Evaluation of the outputs.

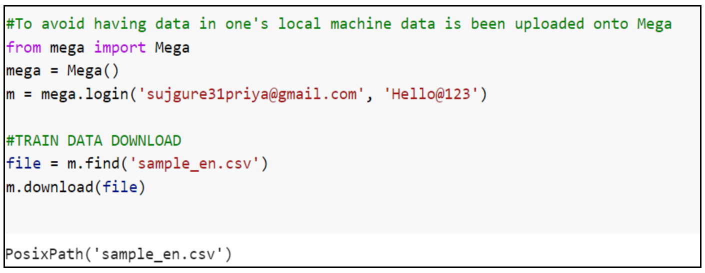


Data understanding
Dataset (in .sqlite)
Accessed the contents through DB browser
Consists of data in different language. Sample from English language is chosen. Data is on "Mega"
DB Browser (SQLite)
english_data
wikibooks.sqlite
jupyter
CO
Data Pre-processing
Load the sample dataset
Drop unnecessary columns
Null value check
EDA
Removing special characters and punctuation
Feature Engineering
Logic and model implementation
Fill in the Blanks
Generating list of sentences from paragraph
Syntactic Analysis
spaCy
Tokenization
Stop word removal
True or False
benepar
summa
Parser
TensorFlow
Tokenization
Data Cleaning specific to True or False Question
Evaluation
Testing on different data and generating questions
evaluation
Success rate on how many questions are grammatically and logically correct

Figure 2: Design flow

3.2 Data Collection

The wikibooks dataset which is available on Kaggle can be downloaded from " https://www.kaggle.com/dhruvildave/wikibooks-dataset" and a sample from that shall be selected. It has the dataset in 12 different languages Out of which English language dataset is to be selected. The available format is in .sqlite which should be converted into .csv format. After that the data should be loaded into the collab. The step of loading data on to "mega" cloud storage can be eliminated and the path where the data is stored can be used.

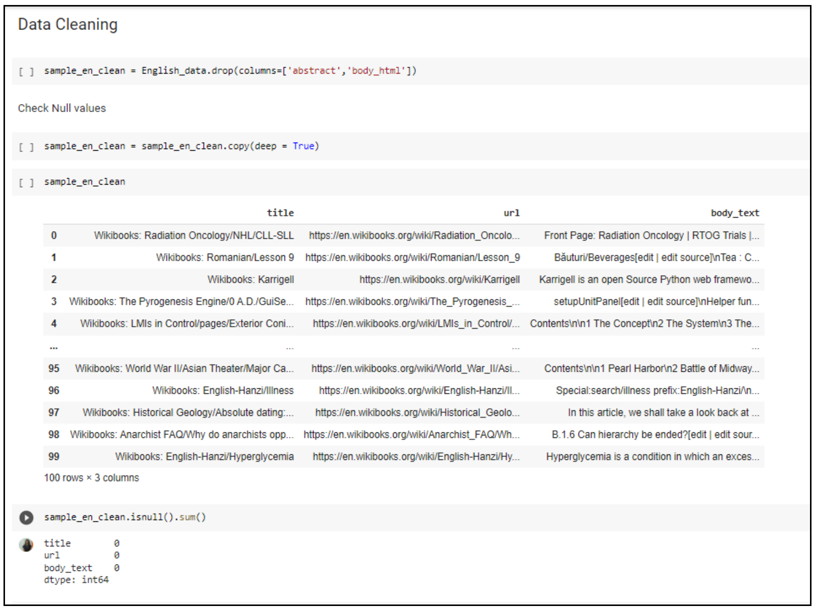


```
#To avoid having data in one's local machine data is been uploaded onto Mega
from mega import Mega
mega = Mega()
m = mega.login('sujgure31priya@gmail.com', 'Hello@123')

#TRAIN DATA DOWNLOAD
file = m.find('sample_en.csv')
m.download(file)

PosixPath('sample_en.csv')
```

Figure 3: Loading the data

3.3 Data Cleaning

The basic cleaning steps performed consists of removing null values, duplicates, dropping unnecessary columns.



Data Cleaning

```
[ ] sample_en_clean = English_data.drop(columns=['abstract','body_html'])
```

Check Null values

```
[ ] sample_en_clean = sample_en_clean.copy(deep = True)
```

```
[ ] sample_en_clean
```

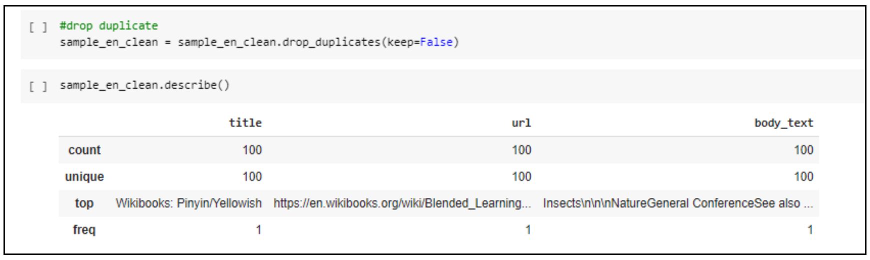
	title	url	body_text
0	Wikibooks: Radiation Oncology/NHL/CLL-SLL	https://en.wikibooks.org/wiki/Radiation_Oncolo...	Front Page: Radiation Oncology | RTOG Trials |...
1	Wikibooks: Romanian/Lesson 9	https://en.wikibooks.org/wiki/Romanian/Lesson_9	Băuturi/Beverages[edit | edit source]\nTea : C...
2	Wikibooks: Karrigell	https://en.wikibooks.org/wiki/Karrigell	Karrigell is an open Source Python web framewo...
3	Wikibooks: The Pyrogenesis Engine/0 A.D./GuiSe...	https://en.wikibooks.org/wiki/The_Pyrogenesis_...	setupUnitPanel[edit | edit source]\nHelper fun...
4	Wikibooks: LMIs in Control/pages/Exterior Coni...	https://en.wikibooks.org/wiki/LMIs_in_Control/...	Contents\n\n1 The Concept\n2 The System\n3 The...
...	...	...	...
95	Wikibooks: World War II/Asian Theater/Major Ca...	https://en.wikibooks.org/wiki/World_War_II/Asi...	Contents\n\n1 Pearl Harbor\n2 Battle of Midway...
96	Wikibooks: English-Hanzi/Illness	https://en.wikibooks.org/wiki/English-Hanzi/Il...	Special:search/illness prefix:English-Hanzi/\n...
97	Wikibooks: Historical Geology/Absolute dating:...	https://en.wikibooks.org/wiki/Historical_Geolo...	In this article, we shall take a look back at ...
98	Wikibooks: Anarchist FAQ/Why do anarchists opp...	https://en.wikibooks.org/wiki/Anarchist_FAQ/Wh...	B.1.6 Can hierarchy be ended?[edit | edit sour...
99	Wikibooks: English-Hanzi/Hyperglycemia	https://en.wikibooks.org/wiki/English-Hanzi/Hy...	Hyperglycemia is a condition in which an exces...

100 rows × 3 columns

```
sample_en_clean.isnull().sum()
```

```
title        0
url          0
body_text    0
dtype: int64
```

Figure 4: Data Cleaning steps



```
[ ] #drop duplicate
    sample_en_clean = sample_en_clean.drop_duplicates(keep=False)
```

```
[ ] sample_en_clean.describe()
```

	title	url	body_text
count	100	100	100
unique	100	100	100
top	Wikibooks: Pinyin/Yellowish	https://en.wikibooks.org/wiki/Blended_Learning...	Insects\n\n\nNatureGeneral ConferenceSee also ...
freq	1	1	1

Figure 5: Data Cleaning steps

4 Exploratory Data Analysis

The data needs to be understood before performing any of the transformations or pre-processing. So, from the histogram performed using R studio it shows much of the data consists of garbage data.

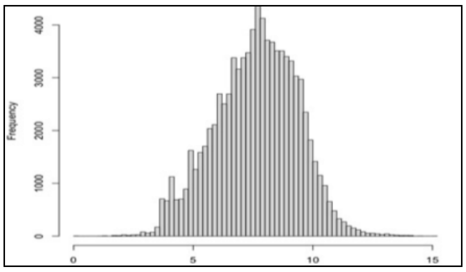
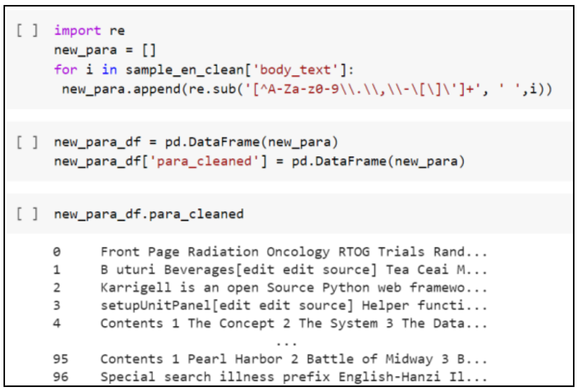


Figure 6: Histogram plotted to understand the data

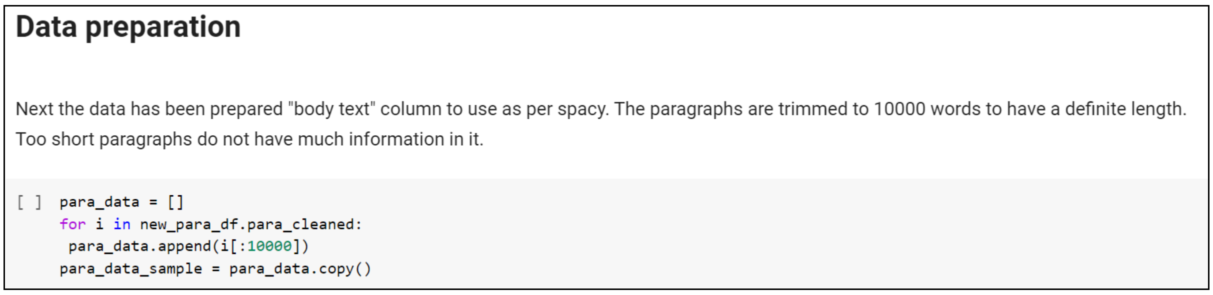
5 Data Transformation/Pre-processing and preparation

First the focus is on selecting correct data. For that the regex to select letters, numbers and punctuations is written and executed.



```
[ ]  import re
     new_para = []
     for i in sample_en_clean['body_text']:
      new_para.append(re.sub('[^A-Za-z0-9\\.\\,\\-\[\]\']+', ' ',i))

[ ]  new_para_df = pd.DataFrame(new_para)
     new_para_df['para_cleaned'] = pd.DataFrame(new_para)

[ ]  new_para_df.para_cleaned

     0      Front Page Radiation Oncology RTOG Trials Rand...
     1      B uturi Beverages[edit edit source] Tea Ceai M...
     2      Karrigell is an open Source Python web framewo...
     3      setupUnitPanel[edit edit source] Helper functi...
     4      Contents 1 The Concept 2 The System 3 The Data...
                                  ...
     95     Contents 1 Pearl Harbor 2 Battle of Midway 3 B...
     96     Special search illness prefix English-Hanzi Il...
```

Figure 7: Cleansing textual data



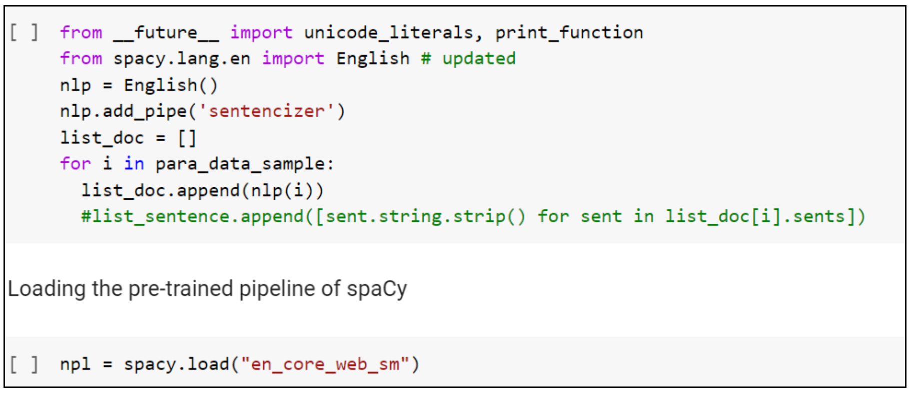
Data preparation

Next the data has been prepared "body text" column to use as per spacy. The paragraphs are trimmed to 10000 words to have a definite length. Too short paragraphs do not have much information in it.

```
[ ]  para_data = []
     for i in new_para_df.para_cleaned:
      para_data.append(i[:10000])
     para_data_sample = para_data.copy()
```

Figure 8: Data preparation

To implement the NLP pipeline the paragraphs are broken into sentences



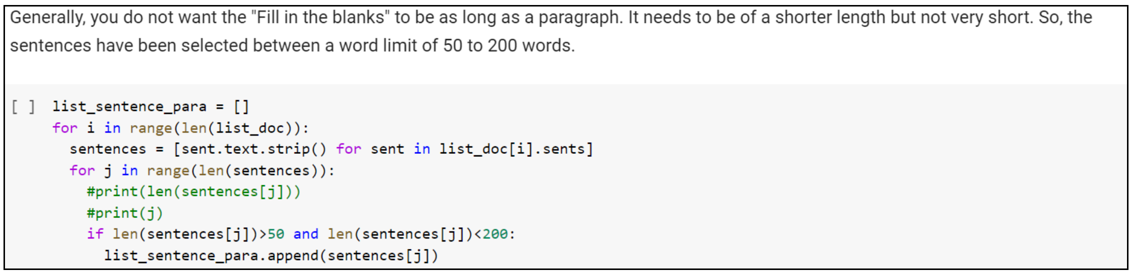
```
[ ]  from __future__ import unicode_literals, print_function
     from spacy.lang.en import English # updated
     nlp = English()
     nlp.add_pipe('sentencizer')
     list_doc = []
     for i in para_data_sample:
       list_doc.append(nlp(i))
       #list_sentence.append([sent.string.strip() for sent in list_doc[i].sents])
```

Loading the pre-trained pipeline of spaCy

```
[ ]  npl = spacy.load("en_core_web_sm")
```

Figure 8: Breaking down the paragraphs into list of sentences

Deciding the length of output so as to not have a very lengthy question


Generally, you do not want the "Fill in the blanks" to be as long as a paragraph. It needs to be of a shorter length but not very short. So, the sentences have been selected between a word limit of 50 to 200 words.

```
[ ]  list_sentence_para = []
     for i in range(len(list_doc)):
       sentences = [sent.text.strip() for sent in list_doc[i].sents]
       for j in range(len(sentences)):
         #print(len(sentences[j]))
         #print(j)
         if len(sentences[j])>50 and len(sentences[j])<200:
           list_sentence_para.append(sentences[j])
```

Figure 8: Breaking down the paragraphs into list of sentences

5.1 Fill in the blank type of question

5.1.1. Cleaning with respect to specific question types

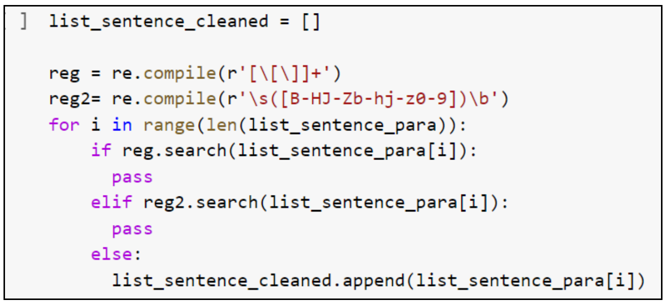
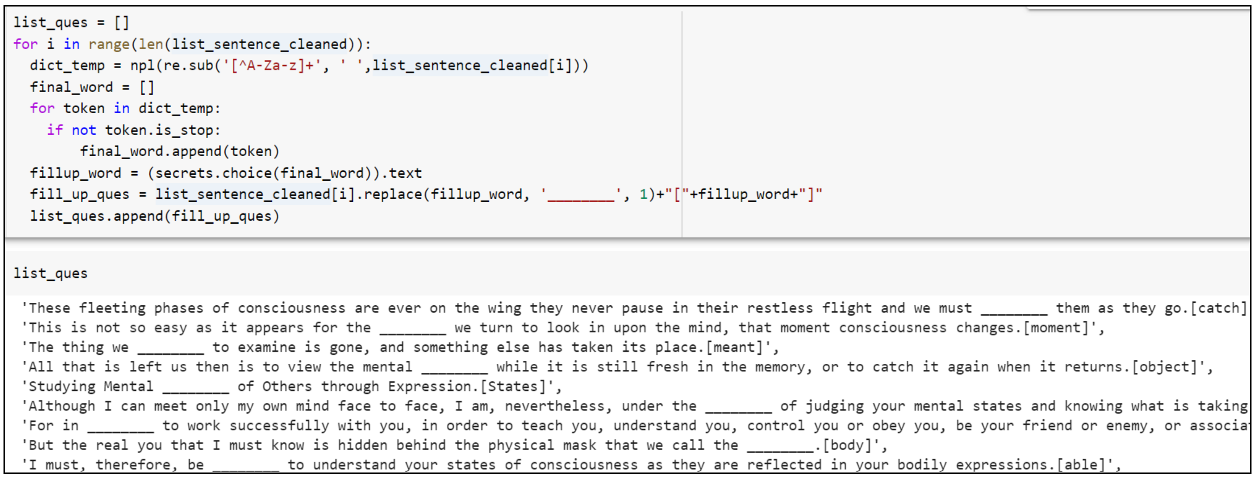

```
 ]  list_sentence_cleaned = []

    reg = re.compile(r'[\[\]]+')
    reg2= re.compile(r'\s([B-HJ-Zb-hj-z0-9])\b')
    for i in range(len(list_sentence_para)):
        if reg.search(list_sentence_para[i]):
          pass
        elif reg2.search(list_sentence_para[i]):
          pass
        else:
          list_sentence_cleaned.append(list_sentence_para[i])
```

Figure 9: Cleaning specific to generating fill in the blank question

5.1.2 Looking for pivotal answers from the sentences and then replacing it with a dash to generate output for fill in the blank question (Tokenization) .

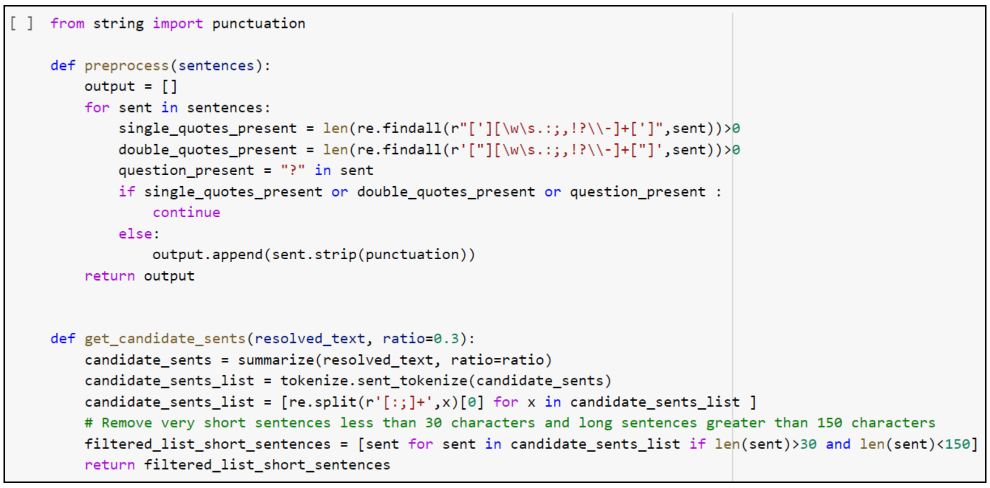

```
list_ques = []
for i in range(len(list_sentence_cleaned)):
  dict_temp = npl(re.sub('[^A-Za-z]+', ' ',list_sentence_cleaned[i]))
  final_word = []
  for token in dict_temp:
    if not token.is_stop:
        final_word.append(token)
  fillup_word = (secrets.choice(final_word)).text
  fill_up_ques = list_sentence_cleaned[i].replace(fillup_word, '________', 1)+"["+fillup_word+"]"
  list_ques.append(fill_up_ques)
```

```
list_ques
```

```
 'These fleeting phases of consciousness are ever on the wing they never pause in their restless flight and we must ________ them as they go.[catch]
 'This is not so easy as it appears for the ________ we turn to look in upon the mind, that moment consciousness changes.[moment]',
 'The thing we ________ to examine is gone, and something else has taken its place.[meant]',
 'All that is left us then is to view the mental ________ while it is still fresh in the memory, or to catch it again when it returns.[object]',
 'Studying Mental ________ of Others through Expression.[States]',
 'Although I can meet only my own mind face to face, I am, nevertheless, under the ________ of judging your mental states and knowing what is taking
 'For in ________ to work successfully with you, in order to teach you, understand you, control you or obey you, be your friend or enemy, or associa
 'But the real you that I must know is hidden behind the physical mask that we call the ________.[body]',
 'I must, therefore, be ________ to understand your states of consciousness as they are reflected in your bodily expressions.[able]',
```

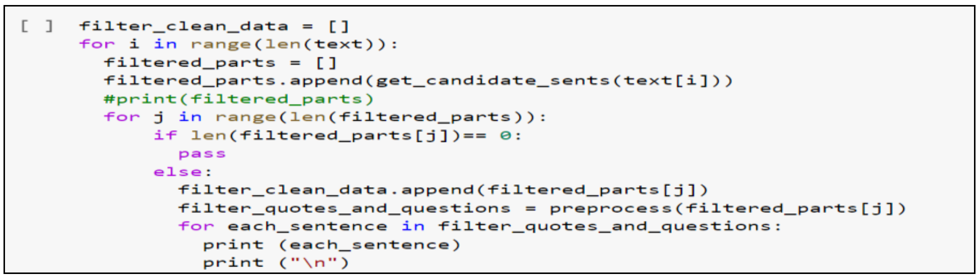
Figure 10: Searching for pivotal answer and then replacing it with a dash

5.2 True or False type question



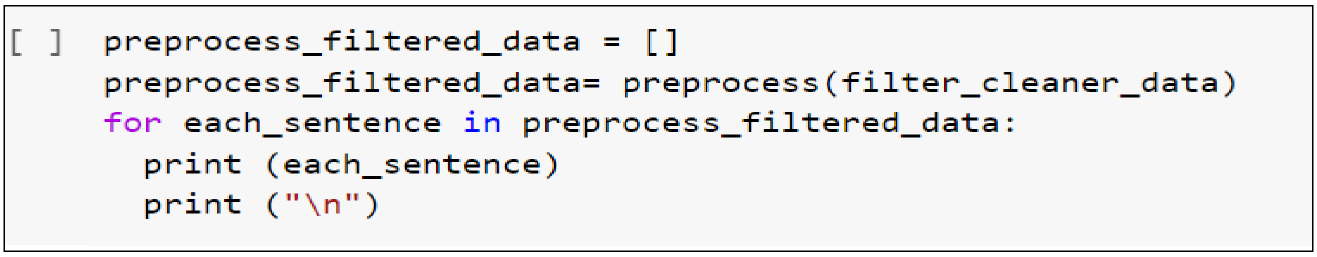
```
[ ]  from string import punctuation

     def preprocess(sentences):
         output = []
         for sent in sentences:
             single_quotes_present = len(re.findall(r"['][\w\s.:;,!?\\-]+[']",sent))>0
             double_quotes_present = len(re.findall(r'["][\w\s.:;,!?\\-]+["]',sent))>0
             question_present = "?" in sent
             if single_quotes_present or double_quotes_present or question_present :
                 continue
             else:
                 output.append(sent.strip(punctuation))
         return output


     def get_candidate_sents(resolved_text, ratio=0.3):
         candidate_sents = summarize(resolved_text, ratio=ratio)
         candidate_sents_list = tokenize.sent_tokenize(candidate_sents)
         candidate_sents_list = [re.split(r'[:;]+',x)[0] for x in candidate_sents_list ]
         # Remove very short sentences less than 30 characters and long sentences greater than 150 characters
         filtered_list_short_sentences = [sent for sent in candidate_sents_list if len(sent)>30 and len(sent)<150]
         return filtered_list_short_sentences
```

Figure 11: Removing quotations and question marks for true or false question



```
[ ]  filter_clean_data = []
     for i in range(len(text)):
       filtered_parts = []
       filtered_parts.append(get_candidate_sents(text[i]))
       #print(filtered_parts)
       for j in range(len(filtered_parts)):
           if len(filtered_parts[j])== 0:
             pass
           else:
             filter_clean_data.append(filtered_parts[j])
             filter_quotes_and_questions = preprocess(filtered_parts[j])
             for each_sentence in filter_quotes_and_questions:
               print (each_sentence)
               print ("\n")
```

Figure 12: Data filtering based on quotes and question marks

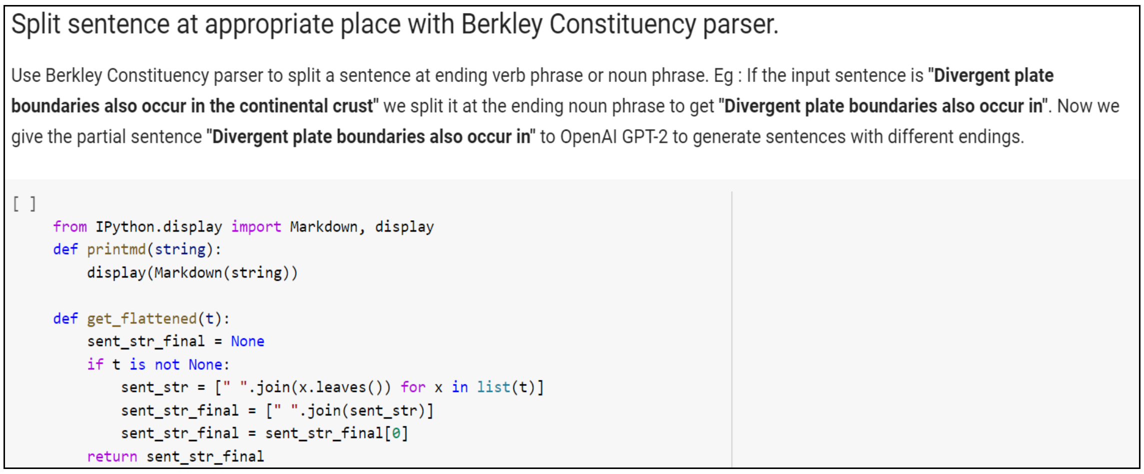


```
[ ]  preprocess_filtered_data = []
     preprocess_filtered_data= preprocess(filter_cleaner_data)
     for each_sentence in preprocess_filtered_data:
       print (each_sentence)
       print ("\n")
```

Figure 13: Displaying the cleaned data

5.2.1 Parsing

Parsing the sentence to split the sentences at the appropriate phrase to generate a different output than the original one using OpenAI GPT-2.

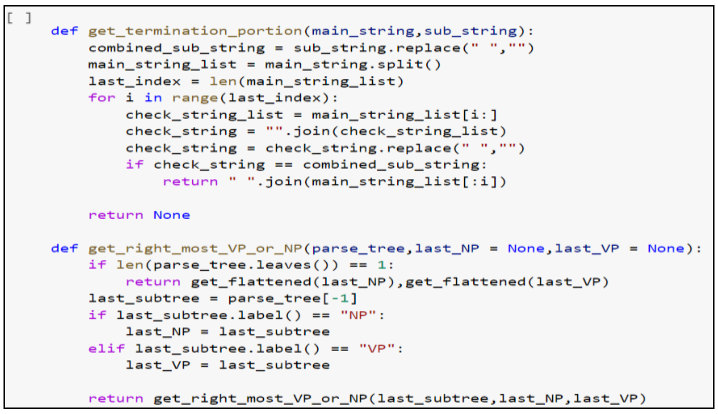


Split sentence at appropriate place with Berkley Constituency parser.

Use Berkley Constituency parser to split a sentence at ending verb phrase or noun phrase. Eg : If the input sentence is "**Divergent plate boundaries also occur in the continental crust**" we split it at the ending noun phrase to get "**Divergent plate boundaries also occur in**". Now we give the partial sentence "**Divergent plate boundaries also occur in**" to OpenAI GPT-2 to generate sentences with different endings.

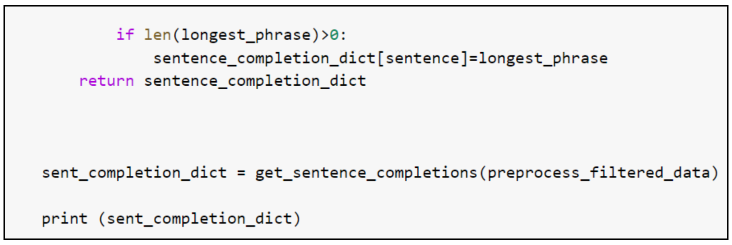
```
[ ]
from IPython.display import Markdown, display
def printmd(string):
    display(Markdown(string))

def get_flattened(t):
    sent_str_final = None
    if t is not None:
        sent_str = [" ".join(x.leaves()) for x in list(t)]
        sent_str_final = [" ".join(sent_str)]
        sent_str_final = sent_str_final[0]
    return sent_str_final
```

Figure 14: Parsing



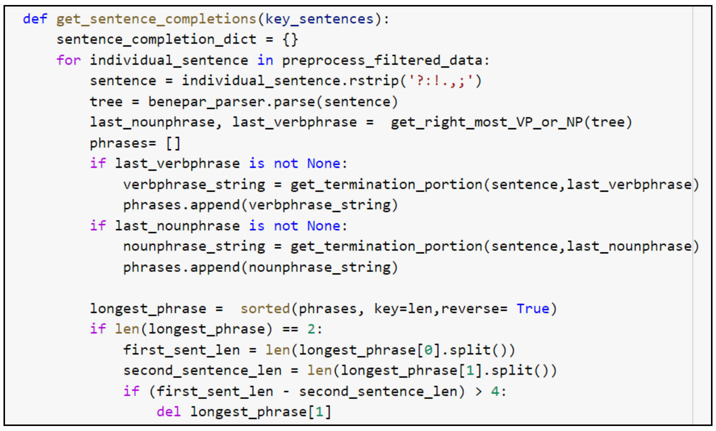
```
[ ]
def get_termination_portion(main_string,sub_string):
    combined_sub_string = sub_string.replace(" ","")
    main_string_list = main_string.split()
    last_index = len(main_string_list)
    for i in range(last_index):
        check_string_list = main_string_list[i:]
        check_string = "".join(check_string_list)
        check_string = check_string.replace(" ","")
        if check_string == combined_sub_string:
            return " ".join(main_string_list[:i])

    return None

def get_right_most_VP_or_NP(parse_tree,last_NP = None,last_VP = None):
    if len(parse_tree.leaves()) == 1:
        return get_flattened(last_NP),get_flattened(last_VP)
    last_subtree = parse_tree[-1]
    if last_subtree.label() == "NP":
        last_NP = last_subtree
    elif last_subtree.label() == "VP":
        last_VP = last_subtree

    return get_right_most_VP_or_NP(last_subtree,last_NP,last_VP)
```

Figure 15: Parsing cont.



```
        if len(longest_phrase)>0:
            sentence_completion_dict[sentence]=longest_phrase
    return sentence_completion_dict



sent_completion_dict = get_sentence_completions(preprocess_filtered_data)

print (sent_completion_dict)
```

Figure 16: Parsing cont. (function to create the dictionary of the sentences)

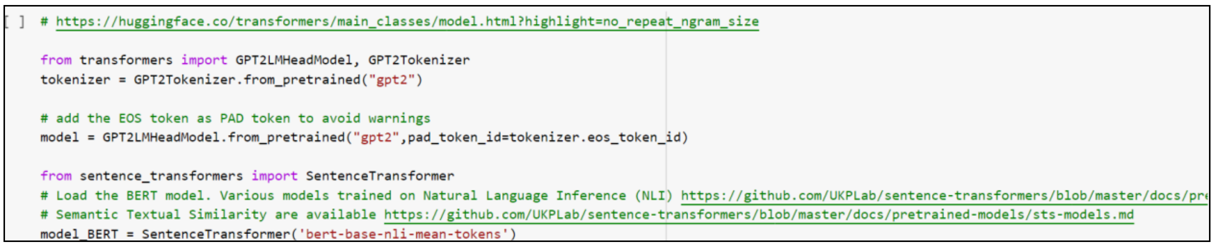


```
def get_sentence_completions(key_sentences):
    sentence_completion_dict = {}
    for individual_sentence in preprocess_filtered_data:
        sentence = individual_sentence.rstrip('?:!.,;')
        tree = benepar_parser.parse(sentence)
        last_nounphrase, last_verbphrase =  get_right_most_VP_or_NP(tree)
        phrases= []
        if last_verbphrase is not None:
            verbphrase_string = get_termination_portion(sentence,last_verbphrase)
            phrases.append(verbphrase_string)
        if last_nounphrase is not None:
            nounphrase_string = get_termination_portion(sentence,last_nounphrase)
            phrases.append(nounphrase_string)

        longest_phrase =  sorted(phrases, key=len,reverse= True)
        if len(longest_phrase) == 2:
            first_sent_len = len(longest_phrase[0].split())
            second_sentence_len = len(longest_phrase[1].split())
            if (first_sent_len - second_sentence_len) > 4:
                del longest_phrase[1]
```

Figure 17: Function or logic on how to treat the sentence upon splitting

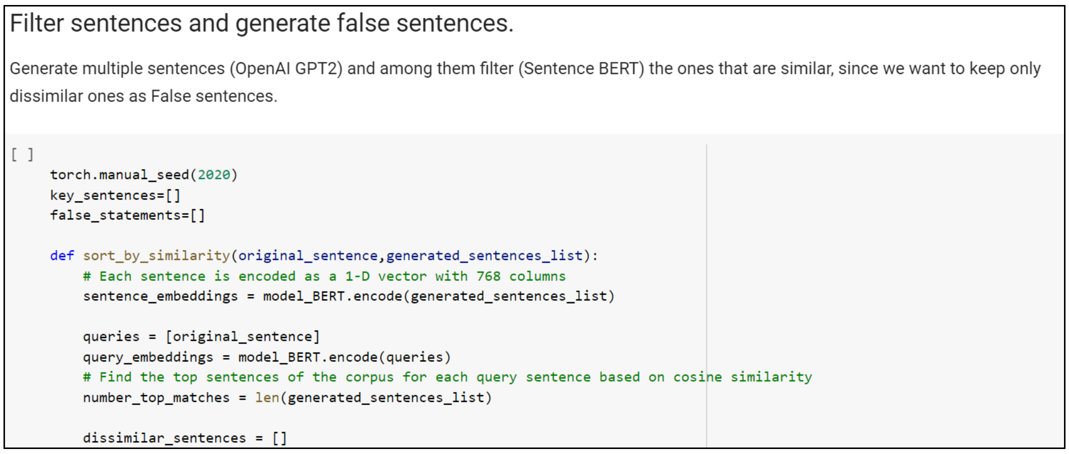
6 Logic or model implementation

Pre-trained BERT model is deployed to generate the true or false questions.



```
[ ] # https://huggingface.co/transformers/main_classes/model.html?highlight=no_repeat_ngram_size

    from transformers import GPT2LMHeadModel, GPT2Tokenizer
    tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

    # add the EOS token as PAD token to avoid warnings
    model = GPT2LMHeadModel.from_pretrained("gpt2",pad_token_id=tokenizer.eos_token_id)

    from sentence_transformers import SentenceTransformer
    # Load the BERT model. Various models trained on Natural Language Inference (NLI) https://github.com/UKPLab/sentence-transformers/blob/master/docs/pre
    # Semantic Textual Similarity are available https://github.com/UKPLab/sentence-transformers/blob/master/docs/pretrained-models/sts-models.md
    model_BERT = SentenceTransformer('bert-base-nli-mean-tokens')
```

Figure 18: Model implementation

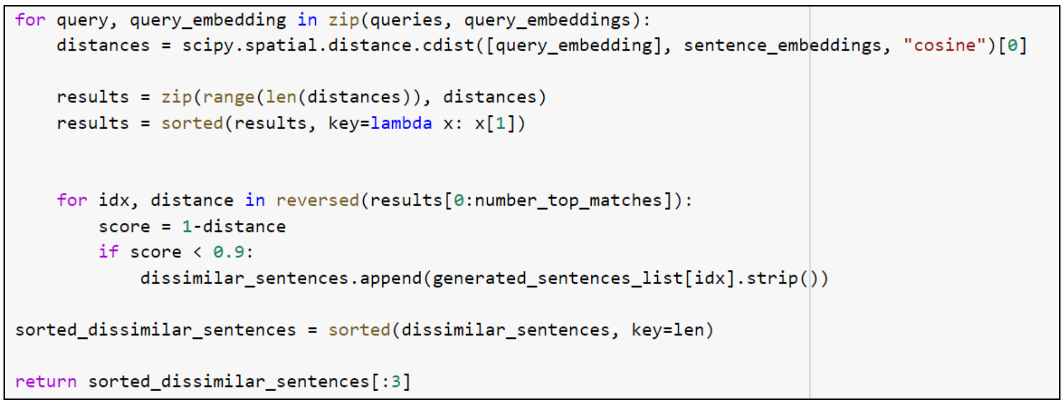


Filter sentences and generate false sentences.

Generate multiple sentences (OpenAI GPT2) and among them filter (Sentence BERT) the ones that are similar, since we want to keep only dissimilar ones as False sentences.

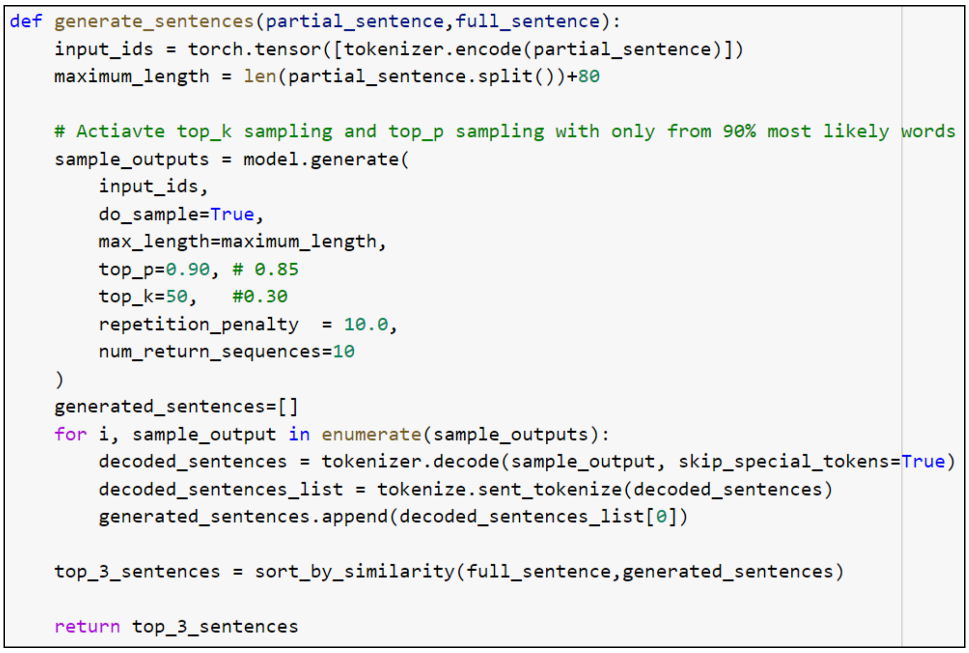
```
[ ]
    torch.manual_seed(2020)
    key_sentences=[]
    false_statements=[]

    def sort_by_similarity(original_sentence,generated_sentences_list):
        # Each sentence is encoded as a 1-D vector with 768 columns
        sentence_embeddings = model_BERT.encode(generated_sentences_list)

        queries = [original_sentence]
        query_embeddings = model_BERT.encode(queries)
        # Find the top sentences of the corpus for each query sentence based on cosine similarity
        number_top_matches = len(generated_sentences_list)

        dissimilar_sentences = []
```

Figure 19: Definition or function to find out sentences different from the original sentences



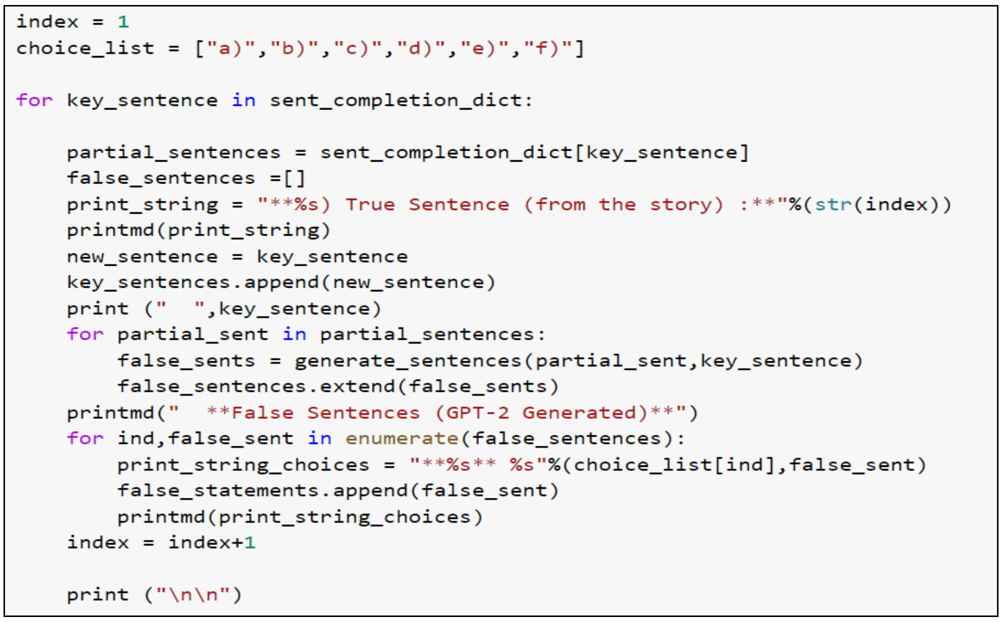
```python
for query, query_embedding in zip(queries, query_embeddings):
    distances = scipy.spatial.distance.cdist([query_embedding], sentence_embeddings, "cosine")[0]

    results = zip(range(len(distances)), distances)
    results = sorted(results, key=lambda x: x[1])


    for idx, distance in reversed(results[0:number_top_matches]):
        score = 1-distance
        if score < 0.9:
            dissimilar_sentences.append(generated_sentences_list[idx].strip())

sorted_dissimilar_sentences = sorted(dissimilar_sentences, key=len)

return sorted_dissimilar_sentences[:3]
```

Figure 20: The comparison to segregate the dissimilar sentences is done by cosine similarity test



```python
def generate_sentences(partial_sentence,full_sentence):
    input_ids = torch.tensor([tokenizer.encode(partial_sentence)])
    maximum_length = len(partial_sentence.split())+80

    # Actiavte top_k sampling and top_p sampling with only from 90% most likely words
    sample_outputs = model.generate(
        input_ids,
        do_sample=True,
        max_length=maximum_length,
        top_p=0.90, # 0.85
        top_k=50,   #0.30
        repetition_penalty  = 10.0,
        num_return_sequences=10
    )
    generated_sentences=[]
    for i, sample_output in enumerate(sample_outputs):
        decoded_sentences = tokenizer.decode(sample_output, skip_special_tokens=True)
        decoded_sentences_list = tokenize.sent_tokenize(decoded_sentences)
        generated_sentences.append(decoded_sentences_list[0])

    top_3_sentences = sort_by_similarity(full_sentence,generated_sentences)

    return top_3_sentences
```

Figure 21: Sampling of the top_k sentences



```python
index = 1
choice_list = ["a)","b)","c)","d)","e)","f)"]

for key_sentence in sent_completion_dict:

    partial_sentences = sent_completion_dict[key_sentence]
    false_sentences =[]
    print_string = "**%s) True Sentence (from the story) :**"%(str(index))
    printmd(print_string)
    new_sentence = key_sentence
    key_sentences.append(new_sentence)
    print ("  ",key_sentence)
    for partial_sent in partial_sentences:
        false_sents = generate_sentences(partial_sent,key_sentence)
        false_sentences.extend(false_sents)
    printmd("  **False Sentences (GPT-2 Generated)**")
    for ind,false_sent in enumerate(false_sentences):
        print_string_choices = "**%s** %s"%(choice_list[ind],false_sent)
        false_statements.append(false_sent)
        printmd(print_string_choices)
    index = index+1

    print ("\n\n")
```

Figure 22: Generating true and false sentences

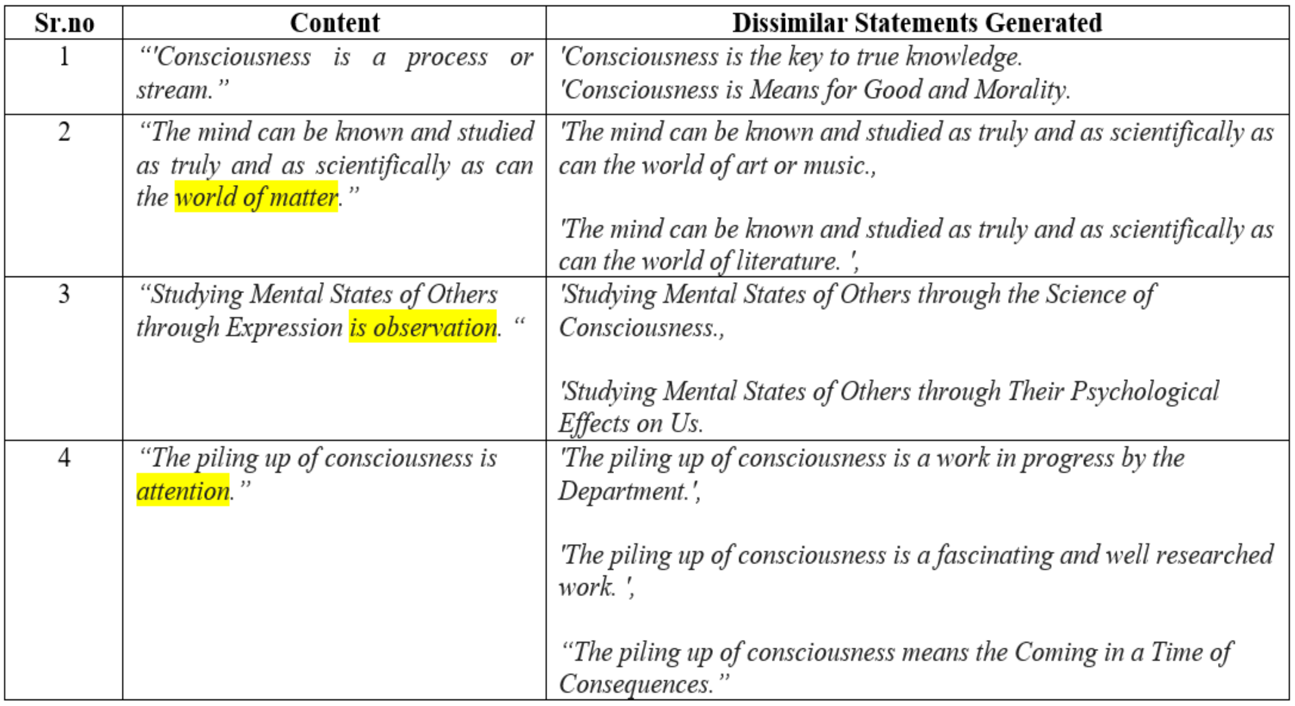
7 Evaluation Results

For evaluation different data can be considered and the generated output can be evaluated.

Experiment 1 fill in the blank output

Sr.no	Content	Question Generated
1	*"They must live their own lives, think their own thoughts, and arrive _________ at their own destiny."*	*They must live their own lives, think their own thoughts, and _________ at their own destiny.*
2	*"In the language of the psychologist, we must introspect."*	*In the language of the psychologist, we must _________.*
3	*'But how are we to discover the nature of the mind or come to know the processes by which consciousness works for mind is intangible.*	*But how are we to discover the nature of the mind or come to know the processes by which _________ works for mind is intangible.*
4	*'Mind belongs not to the realm of matter, which is known to the senses, but to the realm of spirit, which the senses can never grasp.*	*'Mind belongs not to the realm of matter, which is known to the senses, but to the realm of _________, which the senses can never grasp.*
5	*'You and I may look into each other's face and there guess the meaning that lies back of the smile or frown or flash of the eye, and so read something of the mind's activity.'*	*'You and I may look into each other's face and there guess the meaning that lies back of the smile or _________ or flash of the eye, and so read something of the mind's activity.'*
6	*'For one can never come to understand the nature of mind and its laws of working by listening to lectures or reading textbooks alone. '*	*'For one can never come to understand the nature of _________ and its laws of working by listening to lectures or reading textbooks alone.*
7	*'The thing we meant to examine is gone, and something else has taken its place.'*	*'The _________ we meant to examine is gone, and something else has taken its place.'*
8	*'The only way to know what mind is, is to look in upon our own consciousness and observe what is transpiring there.'*	*'The only way to know what mind is, is to look in upon our own _________ and observe what is transpiring there. ',*

Experiment 1 True or false output



Sr.no	Content	Dissimilar Statements Generated
1	*“'Consciousness is a process or stream.”*	*'Consciousness is the key to true knowledge.* *'Consciousness is Means for Good and Morality.*
2	*“The mind can be known and studied as truly and as scientifically as can the world of matter.”*	*'The mind can be known and studied as truly and as scientifically as can the world of art or music.,* *'The mind can be known and studied as truly and as scientifically as can the world of literature. ',*
3	*“Studying Mental States of Others through Expression is observation. “*	*'Studying Mental States of Others through the Science of Consciousness.,* *'Studying Mental States of Others through Their Psychological Effects on Us.*
4	*“The piling up of consciousness is attention.”*	*'The piling up of consciousness is a work in progress by the Department.',* *'The piling up of consciousness is a fascinating and well researched work. ',* *“The piling up of consciousness means the Coming in a Time of Consequences.”*

References

1. https://www.kaggle.com/dhruvildave/wikibooks-dataset