```
#########################################################
#           REQUIRES MATPLOTLIB AND NUMPY               #
#########################################################
import random; random.seed(123)
import codecs
import string
from nltk.stem.porter import PorterStemmer
from nltk.probability import FreqDist
import gensim
import matplotlib
import matplotlib.pyplot as plt
import numpy as np


######### Task 1 #########
# Open the file
f = codecs.open("pg3300.txt", "r", "utf-8")

f = f.readlines()
text = ""
for line in f:
    text += line

# Split the text into paragraphs.
text = text.split("\n\r")

# Remove lines that contains Gutenberg
without_gutenberg = [par for par in text if "gutenberg" not in par.lower()]

# Remove all empty lines:
filtered_list = [par for par in without_gutenberg if len(par) > 0]

# Splits each paragraph into words
words = [par.split() for par in filtered_list]

# Initialize stemmer and freqDist
stemmer = PorterStemmer()
freqDist =  FreqDist()

# Stem words and remove punctuation. Also update freqDist.
for i, par in enumerate(words):
    for j, word in enumerate(par):
            words[i][j] = stemmer.stem(word.strip(string.punctuation + "\n\r\t").lower())
            freqDist[words[i][j]] += 1
    words[i] = list(filter(None, words[i])) # Filter out empty strings

# Filter out empty lists
words = list(filter(None, words))


# Start plot
freqDist = sorted(freqDist.items(), key=lambda x: x[1], reverse=True)[:15]

labels = [x[0] for x in freqDist]
values = [x[1] for x in freqDist]

# This code for plotting was taken from https://matplotlib.org/3.1.1/gallery/lines_bars_and_markers/barchart.html#sphx-glr-gallery-lines-bars-and-markers-barchart-py
# Wtih minor changes.
x = np.arange(len(labels))
width = 0.40

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, values, width, label='Freq')
ax.set_ylabel('Freq')
ax.set_title('Top 15 most frequent words')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

def autolabel(rects):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),
                    textcoords="offset points",
                    ha='center', va='bottom')

autolabel(rects1)
fig.tight_layout()

plt.show()
# End plot

#########################################################################
```

```
#                                                                       #
#                 TO MAKE THE FOLLOWING CODE RUN                        #
#                   CLOSE THE PLOT WINDOW!                              #
#                                                                       #
#########################################################################
########## Task 2
# Create dictionary
dictionary = gensim.corpora.Dictionary(words)

# Open stopwords and make a list on the form ["word1", "word2", ..."wordN"]
stopwords = open('stopwords.txt', 'r').read().split(",")

# Stem the stopwords
stopwords = [stemmer.stem(w) for w in stopwords]

# Create list with stopwords ids
stopword_ids = []
for w in stopwords:
    try:
        stopword_ids.append(dictionary.token2id[w])
    except:
        continue

# Filter out stopwords
dictionary.filter_tokens(stopword_ids)

# Create a bag of words
bag_of_words = list()
for par in words:
    bag_of_words.append(dictionary.doc2bow(par))

tfidf_model = gensim.models.TfidfModel(bag_of_words)

corpus_tfidf = tfidf_model[bag_of_words]

# Create matrix similarity
matrixsim = gensim.similarities.MatrixSimilarity(bag_of_words)

# Do the same for LSI:
lsi = gensim.models.LsiModel(corpus_tfidf, id2word=dictionary, num_topics=100)
lsi_corpus = lsi[corpus_tfidf]


# Print three first topics.
print('################################################')

print('Three first lsi topics')
print(lsi.show_topics(3))
print('################################################\n\n\n')

######## Task 4 ########
# Function for preprocessing of query
def preprocessing(q):
    return [stemmer.stem(w.strip(string.punctuation).lower()) for w in q.split()]

# Preprocess the query and convert to bag of words
q = preprocessing("What is the function of money?")
q = dictionary.doc2bow(q)


# Report weights
q_tfidf = tfidf_model[q]
index = gensim.similarities.MatrixSimilarity(corpus_tfidf)

print('################################################')
print('Reporting weights')
for pair in q_tfidf:
    weight = pair[1]
    word = dictionary.get(pair[0])
    print(f'Word:Â {word}, weight: {weight}')
print('################################################\n\n\n')

def report_relevant_paragraphs(q):
    docs2similarity = enumerate(index[q])
    sorted_docs = sorted(docs2similarity, key=lambda kv: -kv[1])[:3] #Sort the docs.
    relevant_paragraphs = list()
    for pair in sorted_docs:
        relevant_paragraphs.append(pair[0]) # Append the paragraph index to relevant paragraphs

    for par in relevant_paragraphs:
        print(f'[Paragraph: {par+1}]')
        lines = filtered_list[par].splitlines(6)[:6] # Split the paragraphs into lines and get the first 5
        print(''.join(lines) + '\n') # Print the lines.

print('################################################')
print('Relevant paragraphs for query "What is the function of money?')
report_relevant_paragraphs(q_tfidf) # Print the relevant docs (first 5 lines).
print('################################################\n\n\n')
```

```
print('##############################################')
print('Top 3 topics for "What is the function of money?')
lsi_corpus_q = lsi[q_tfidf]
sorted_lsi = (sorted(lsi_corpus_q, key=lambda kv: -abs(kv[1]))[:3] )
all_test_topics = (lsi.show_topics())
for i in sorted_lsi:
    print("[Topic ", i[0], "]")
    print(all_test_topics[i[0]][1])
print('##############################################')
```