| QUESTION 1|

Code used is marked in the python below

ID: 296, Word: human ID: 735, Word: humanitarian ID: 953, Word: humans ID: 2910, Word: humanity ID: 7356, Word: humanism ID: 9270, Word: humankind ID: 16429, Word: humanities ID: 24754, Word: humanistic ID: 26705, Word: humanist ID: 30593, Word: humanoid ID: 32096, Word: humane

```
########## QUESTION 1 CODE ###########
# Iterate through all items in id2word_wiki and filter for words
starting with "human"
humanWords = {id: word for id, word in id2word_wiki.items() if
word.startswith("human")}

# Print the results
for wordID, wrd in humanWords.items():
    print(f"Word ID: {wordID}, Actual Word: {wrd}")
########## QUESTION 1 CODE ###########
```

| QUESTION 2|

Code used is marked in the python below

[('normally', 0.3314205262599425), ('blood', 0.5989905558961313), ('produced', 0.23752276286313315), ('cell', 0.6891688369642741)]

```
########## QUESTION 2 CODE ###########
bv = id2word_wiki.doc2bow(tokenize(text))

# Transform the bag-of-words vector into TF-IDF space
vTFIDF = tfidf_model[bv]

# Print the transformed vector in TF-IDF space
print([(id2word_wiki[id], value) for id, value in vTFIDF])
########## QUESTION 2 CODE ###########
```

| QUESTION 3|

The difference in output between the two notebooks is due to how the data is stored in each and the way in which it is iterated through in order to create 'top_words'. Radim's notebook prints out the actual top words, while the fixed-up notebook prints out decimal numbers. To fix this is the fixed-up notebook the only thing that has to be changed is fliiping the order of *', words' to 'words,* '. This means the final code snippet should look like this:

top_words = [[word for word,_ in lda_model.show_topic(topicno, topn=50)] for topicno in range(lda_model.num_topics)] print(top_words)

| QUESTION 4|

The misplaced word technique involves artificially replacing one word within each topic with an unrelated word then going and observing how good a job the algorithm does of identifying that misplaced word.

Code from the notebook that I believe does this:

replace_index = np.random.randint(0, 10, lda_model.num_topics)

replacements = [] for topicno, words in enumerate(top_words): other_words = all_words.difference(words) replacement = np.random.choice(list(other_words)) replacements.append((words[replace_index[topicno]], replacement)) words[replace_index[topicno]] = replacement print (topicno, ' '.join([str(w) for w in words[:10]])) # print("%i: %s" % (topicno, ' '.join(words[:10])))

print("Actual replacements were:") print(list(enumerate(replacements)))

| QUESTION 5|

The goal in the Half & Half technique is to split each document into two halves, evaluate the topic assignments for each half, and measure both the Intra-document similarity (the topics of the two halves of the same document should be similar) and the Inter-document similarity (the topics of halves from different documents should be dissimilar).

Code from the notebook that I believe does this:

doc_stream = (tokens for _, tokens in iter_wiki(wiki_file)) # generator test_docs = list(itertools.islice(doc_stream, 8000, 9000))

def intra_inter(model, test_docs, num_pairs=10000): # split each test document into two halves and compute topics for each half half = int(len(test_docs)/2) part1 = [model[id2word_wiki.doc2bow(tokens[: half])] for tokens in test_docs] part2 = [model[id2word_wiki.doc2bow(tokens[half :])] for tokens in test_docs]

```
# print computed similarities (uses cossim)
print("average cosine similarity between corresponding parts (higher
is better):")
print(np.mean([gensim.matutils.cossim(p1, p2) for p1, p2 in zip(part1,
part2)]))

random_pairs = np.random.randint(0, len(test_docs), size=(num_pairs,
2))
print("average cosine similarity between 10,000 random parts (lower is
better):")
print(np.mean([gensim.matutils.cossim(part1[i[0]], part2[i[1]]) for i
in random_pairs]))
```

print("LDA results:") intra_inter(lda_model, test_docs)

print("LSI results:") intra_inter(lsi_model, test_docs)

| QUESTION 6|

Using the Half & Half Technique mentioned in the previous question we can get the data from intra_inter to best answer this question.

Analysis of the LDA Results: Intra-document similarity (0.5139) - Indicates that the two halves of the same document have moderately similar topic distributions. - This is expected for LDA, as it is designed to assign specific topics to parts of a document probabilistically. Inter-document similarity (0.4645) - Shows that topic distributions between random parts of different documents are somewhat dissimilar, but not highly so. - This indicates some overlap in topic assignments across documents, possibly due to shared vocabulary or themes in the dataset.

Analysis of LSI Results: Intra-document similarity (0.0645) - The very low value suggests that LSI struggles to capture specific topics within a single document. - LSI is more focused on capturing general patterns, which may not align well with splitting documents into halves. Inter-document similarity (0.0080) - The extremely low value indicates that random parts of different documents are highly dissimilar in topic space. - This is expected because LSI emphasizes broad semantic structures rather than document-specific topics.

Based on all of this I would say the LDA is the better technique for this data because it not only captures more coherent topics within documents, but it also provides interpretable topics that align with the dataset's structure. While LSI does have better inter-document distinction, LDA has better intra-document distinction and a better ability to interpret topics.