1. **What are two reasons why you may decide to use a word2vec embedding model instead of count vectorization to convert your text documents into numeric vectors?**

    - To reduce the number of dimensions in your features (remember, word embeddings are usually 100, 200, 300 elements long, whereas count vectorization means you have as many features as the # of unique tokens in your dataset (often tens of thousands)
    - You want a better representation of the semantic meaning of the word. In count vectorization, each word is equally distant from any other word. So the word **dog** is as similar to the word **puppy** as it is similar to the word **bamboozle**.
    - You have very long documents, and you only want to consider words within a limited context window – count vectorization technique usually just count the frequency of the word in the document. But word embeddings usually apply a context window.
    - You want to leverage the work done by larger organizations and research labs, like Google, who have spent time/resources training embeddings on massive corpora of billions of documents. There is no "learning" in count vectorization – it is simply returning frequencies. In word embeddings, the "learning" is captured in the updates to the word and context matrices of the neural network.

2. **Which of the following code snippets is better suited to handle Big Data-sized text datasets? Explain why.**

Option C is the better option. By using file.readline(), you are reading in incrementally one line at a time from a dataset of unknown size. This allows you to process streams of data that may be too large to fit into memory on your computer. Option A uses pandas to read all of the data in at once, so it may run into memory issues. Option B is the same – it calls file.readlines(), which will read all of the data from the file into memory as well.

3. **You are working to build an NLP model. Your manager reviews your code and asks you to reduce the n-gram size in your model. What are two reasons why she may ask you to reduce the n value of your n-grams?**

    - Your model would begin to overfit, since you'll start to get very exact n-grams that are relevant to only a single document, like ("quickly ate at McDonalds before work at 6pm").
    - You get too many distinct features, from having too many combinations of tokens. Remember, unigram → the number of unique tokens. Bigram → the number of unique token → token combinations. Trigram → the number of unique token → token → token combinations, and so on.

4. **You notice a junior data scientist's regex pattern for finding all references to male children is re.findall(r'son'). Identify one issue w/ this regex pattern and**

propose an improved version that would reduce the number of false positives/negatives.

The regex pattern needs to have word boundaries, or otherwise it will match words like **sonar**, or **Jason**. You probably also want to have **sons?** To capture both **son** and **sons**.

5. According to Zipf's Law (assume an alpha $\alpha = 1$), the top 5 words in your corpus would comprise what percentage of your total word count?

The top 5 words would be 22.83% of all words.

- 1st: 10%
- 2nd: 5%
- 3rd: 3.33%
- 4th: 2.5%
- 5th: 2%

10% + 5% + 3.33% + 2.5% + 2% = 22.83%

6. **If you want to make sure your text search query model, when it is provided an input "baby care" returns all relevant products baby care products, would you optimize for recall or precision?**

   You should optimize for recall. Recall is the number of relevant products (True Positives) divided by the number of Actual Positives (TP + FN). You can minimize the number of false negatives in your prediction by just predicting positive a lot (although this has the tradeoff of reducing your precision).

7. **Would a bag of words model work better for representing as documents, product tags or New York Times articles? Why?**

   A product tags, since it is significantly shorter than a NYT article, and also because there is no clear sequential relationship between each tag (for the most part).

True/False

A. **True**
B. **True**

C. **False. It is ill-suited for high dimensions since comparing any two documents with high dimensions using Euclidean distance yields a relatively equal distance (the difference between the most distant and most similar documents in high dimensional space shrinks)**
D. **True**
E. **Latin 1 cannot represent the character, since it is a codepoint much higher than 256 (which is the limit of Latin 1).**