Introductory

- The prompt for this assignment states that you're a data scientist in healthcare managing tweets related to Coronavirus. You're given two datasets containing the columns shown here.
- We must import the necessary items before running any code.
- Next, we must import the datasets. I decided to use only the first 1000 rows of the Train dataset and the first 100 rows of the Test dataset to save time.
- I feel that it's good practice to show a portion of the dataset so we can see what we're working with, and this is something I do throughout first half the project with both sets of data. Here is the Train data.
- And here is the Test Data

Problem 1

- The first problem asks us to concatenate three columns in the data into a new column called Tweet Texts. I chose to remove all Nan values before doing this. Here is the code and the results.

Problem 2

- Problem 2 is about cleaning and preprocessing the data, and it contains several sub-steps. The first is about removing the dates and times in the tweets through regular expressions. I created four such expressions to remove the dates and times in as many different formats as possible, and the results are shown here.
- The next part of Problem 2 involves removing the hyperlinks using a regular expression. The code and outputs are displayed.
- Next comes removing the hashtags; again, the code and outputs are shown here.
- Then we remove the usernames.
- And this is followed by removing all one and two letter words.
- We get rid of all special characters.
- And we remove any rows with no text in the Tweet Texts column
- Next, we're tasked with performing part of speech tagging, which involves loading "en_core_web_sm" and applying it to each row so we can get information on each word in each tweet.
- We then visualize the dependency parser; I chose to do this for only the first 10 rows in each dataset, as applying it to them all created substantial lag on my computer. The code and outputs are here.
- We perform named entities recognition for the text, which entails looping through the tweets to find any named entities.
- The final component of Problem 2 involves highlighting named entities with specific labels in each tweet. It should be noted that there is no label "GEOLOCATION." I split it into "GPE" and "LOC." First, we must input the labels we're looking for and what colors to associate with each. Then we loop through the rows to perform this operation on each tweet.

Problem 3

- The third problem asks us to extract all the tokens from the Tweet Texts column. The code and outputs are shown here

Problem 4

- The next problem tasks us with replacing each word in Tweet Texts with the lemmas. A function that lemmatizes each word in each sentence and then joins them into new sentences must be defined. It's then run on the datasets and displayed here.

Problem 5

- We must now graphically summarize the sentiment and doing so on each dataset shows that they're both somewhat unbalanced.

Problem 6

- Next, we must graphically summarize the length of the text of the tweets using multiple different types of plots. Before doing this, it was necessary to get the original, uncleaned text from the Tweet Texts column. I did this by re-reading the datasets and saving them under the names "Original Train and Test Data." I then performed the same Nan value removal and concatenation that I did earlier. Next, I got the length of the text for each tweet in the original and cleaned versions of the Train and Test datasets and saved them in a new column. Before making the plots, we should display values for each dataset, such as the count, mean, standard deviation, and quartiles.
- Once we have all of this, we're able to create the necessary boxplots.
- Next comes the histograms.
- And last comes the density plots.

Problem 7

- The next problem is like Problem 6, though it instead asks for word counts. After getting the word counts for each tweet in each dataset, we save them in new columns and display values for each dataset like before.
- We then create the boxplots.
- Then we make histograms.
- And lastly, we create the density plots.

Problem 8

- Problem 8 instructs us to summarize the top 10 unigrams and bigrams of the tf-idf of the text of the Tweet Texts column. First, we define a function to get the top words in each dataset that has parameters such as "ngram_range" and "top_n."
- We then apply it to the original and cleaned Train data to get the top 10 unigrams and display them in the chart shown here.
- We then repeat the process to find and display the top 10 bigrams in the Train data.
- Next, we use our function to find and display the top 10 unigrams in the Test data.
- Lastly, we use it to find and display the top 10 bigrams in the Test data.

Problems 9-10

- Problems 9 and 10 ask for us to visualize the top 10 term frequencies and positive scores for the tweets while limiting ourselves to the "Positive" and "Extremely Positive" sentiments. I found that these problems can be done together to save space in the program. Also, it was necessary to create new data frames so that the two sentiments could be grouped together without affecting our existing data frames.
- First, we generate a corpus for the original Train text and one for the cleaned Train text. For each of these, we acquire the term frequency for the positive sentiment and the corresponding score for this sentiment, and then we rank them in decreasing order.
- Next, we visualize the top 10 frequencies of the tokens related to the positive tweets both before and after the text is cleaned in the Train data.
- Then we do the same with the top 10 scores of the tokens related to the positive tweets.
- We repeat the whole process on the Test dataset. We create the corpuses.
- We visualize the top 10 frequencies.
- And lastly, we visualize the top 10 scores.

Problem 11

- We are then tasked with converting the Tweet Texts column to a matrix of token counts. The code and the outputs are displayed here for both the Train and Test Data.

Problem 12

- Next, we perform tf-idf analysis on Tweet Texts, for which the code and outputs are similar to the last question and are shown here.

Problem 13

- We are now given the task of finding the cosine similarity between two of the tweets in the Train data. While the question says to use the 200th and 20000th tweets, this is not possible with the subset I used, so I chose to compare the 50th and 500th tweets instead. After fitting the tweets to the corpus and converting raw documents to a matrix of tf-idf features, we find the cosine similarity between the two based on the cosine distance.

Problem 14

- The next problem asks us to find the corpus vector that's the average of all the document vectors. The code and the output are shown here.

Problem 15

- Now we must build the first model based on the training dataset. Before we can do so, there are a few things that must be done, such as creating a tokenizer function as well as a customized Scikit-learn transformer for cleaning the texts. Although cleaning has already been performed, omitting this doesn't meaningfully improve accuracy in later problems. Also, failing to include this here has the potential to adversely affect later code.

- We also need to encode the labels to numerical values and then split the dataset.
- Now that that is done, we construct a pipeline to clean and vectorize the texts as well as perform classification using the Random Forest Classifier. It's good practice to look at the confusion matrix; here, we code the matrices for before and after normalization.
- The matrices are shown here, with the pre-normalized one on the left and the normalized one on the right.
- Lastly, we summarize the information using a classification report.

Problem 16

- The next question asks whether the model is a good one based on the evaluation metrics. We conclude that it is not a good model as most of the values are only between 0.10 and 0.50. It is important to note the accuracy provided by the report may be misleading since the dataset is unbalanced.

Problem 17

- The next model must be built using the Random Forest Classifier, Pipeline, and Grid Search CV. It entails performing a grid search with Grid Search CV on some hyperparameters of the estimators. The code and the ideal parameters are shown here.

Problem 18

- The next problem instructs us to tune the model, perform model diagnostics on the test dataset, and determine if it is a good model. First, we create the code for pre- and post-normalized confusion matrices as we did before.
- The results are shown here.
- A classification report shows that the model is poor, since as before, most values are only between 0.10 and 0.50. Also, the accuracy again has the potential to be misleading since the dataset is unbalanced.

Problem 19

- Next, we build a third model that uses Grid Search CV on multiple classifiers. In this case, we're using logistic regression, support vector machine, and random forest. We first define an estimator class that can handle different classifiers.
- Next, we define the pipeline and hyperparameters. It's important to remember that the cleaners and vectorizers only make use of one double underscore, but the classifiers have nested parameters which necessitate the use of two double underscores.
- We then display the optimal parameters.

Problem 20

- We must now repeat what we did with the second model. However, due to the varying parameters, we can't use the usual confusion matrix method and instead must use seaborn to plot the confusion matrices.

- The matrices are shown here.
- The classification report again shows that the model is poor, since the values have not improved much over the last two models. Most are now only between 0.30 and 0.60. The accuracy still has the potential to be misleading.

Problem 21

- The next problem instructs us to create a Latent Dirichlet (dee-rush-lay) Allocation (LDA) algorithm using the LDA method and Count Vectorizer. As we do this, we must set five topics.

Problem 22

- Now we need to plot the top 15 words for each of the five topics we defined in the last problem. We first display the words, and then we create a function to plot them.
- The plot of the 15 top words for each topic is shown here.

Problem 23

- We must now create a second LDA algorithm using the LDA method and Tf-idf Vectorizer. The code for this one has some slight differences to the first algorithm, but it’s mostly the same as before.

Problem 24

- Here, the words are shown and plotted as before.

Problem 25

- For the last problem, we must visualize the second model we just created using a dimension reduction method. I chose to use the default, which was Principal Coordinates Analysis, or PCOA. The circles in the output represent the different topics. Their positioning reflects their similarities, and their size shows their frequency in the corpus. The panel on the right shows the top 30 terms and will adjust to whichever topic is selected.

Questions?