Using NLP to Predict the Likelihood of Certain Types of Negative Events Occurring with Medical Devices

Problem Statement:

This capstone project aims to identify and study factors via unsupervised learning that influence certain types of negative events that occur with medical devices. These negative event types are recall, safety alert, field safety notice, and a combination of two or more of them. The primary clients this project targets are those that work in or deal with the medical field, particularly those that manufacture or utilize medical devices as well as the patients that receive them. Medical devices are known to save, extend, or made better millions of lives. However, according to the International Consortium of Investigative Journalists (ICIJ), more than 1.7 million injuries and nearly 83,000 deaths are suspected of being linked to medical devices over 10 years and reported to the U.S. alone. Additionally, the ICIJ noted that the U.S. had more than 26,700 device recalls while India — with more than a billion people — had just 14 from 2013 to 2017. It is important to consider though that this could be due to the U.S. having more rigorous testing and mandatory regulations and the possibility that medical devices were utilized more in the U.S. than in India. Patients often are the last to be informed about malfunctioning devices. Thus, this project can help medical practitioners make better decisions on which medical devices to use with the most minimal of risks as well as patients to be better informed on which medical devices they want to receive. Also, the results of this project can give indications to manufacturers on how to better design medical devices with reduction to injury and mortality.

The International Medical Devices Database will be used for analysis in this project. It is licensed under the Open Database License and its contents under Creative Commons Attribution-ShareAlike license. It is also available for download by the link: https://medicaldevices.icij.org/p/download.

Dataset Description:

It was decided that an unsupervised prediction model using NLP would be created to predict whether a medical device will issue a recall, a safety alert, a field safety notice, or a combination of two or more of them. For the purpose of data exploration and of the main analysis of this project, the df_final.csv file was merged and cleaned from three CSV files (devices-1562662526.csv, events-1562662544.csv, and manufacturers-1562662522.csv). This section describes the data wrangling and cleaning steps and methods used to create the final df_final.csv file.

Using IPython Notebook, devices-1562662526.csv, events-1562662544.csv, and manufacturers-1562662522.csv were read into dataframes called devices, events, and manufacturers respectfully. The devices dataframe had 15 columns and 104,066 rows, the manufacturers dataframe had 10 columns and 26,013 rows, and the events dataframe had 30 columns and 109,574 rows. The devices and manufacturers dataframes were merged (in an outer join fashion on manufacturer_id) to create the df dataframe which had 24 columns and 104,137 rows. Then, the df dataframe was merged (in an outer join fashion on device_id) with the events dataframe to create the df_final dataframe which had 53 columns and 109,645 rows.

Columns in the df_final dataframe were removed based on 60% missing percentage criteria as well as whether or not the columns would contribute to the data exploration and the main purpose of this project. By a boolean array, columns with a missing (null) percentage of 60% or more were removed. Then, it was decided that device_id (ID of the device), device_name (name of device), device_country (country where device was created), event_id (ID of event), action_classification (event risk class), event_country (country where the event took place), reason (textual reasons device is under investigation or reported), and type (event type) would be kept in the df_final dataframe while the others were dropped. This left the df_final dataframe with 8 columns and 109,645 rows.

The columns were then checked and reorganized based on inconsistent and missing values. The values of the action_classification (event risk class) were reorganized to make its string values more consistent. For example, "I," "Class I," and "Class 1" all denoted the category, Class 1, under the action_classification column, and thus "I" and "Class I" were changed into the string value of "Class 1" to indicate they were all of the same categorical value. Additionally, there were outlier values of "Unclassified Correction" and "Voluntary recall." As there were only four of them, the rows containing "Unclassified Correction" and "Voluntary recall" under the action_classification column were removed from the df_final dataframe since their removal won't affect the analysis results of this project. Finally, the rows containing the null values of the type and reason columns were removed from the df_final dataframe. This leaves the df_final dataframe with 8 columns and 61,799 rows.

Text cleaning was done on the reason column to prepare the df_final dataframe for the NLP analysis of the project. First, a function called, clean_text, was created to keep only alphabetical words, remove whitespaces, and convert the text to lowercase before it was applied on the reason column with the new column called, clean_reason, displaying the cleaned text. Next, a function called, stem_text, was written to help to stem the words of the text under the clean_reason column. Then, the remove_stopwords function was created to remove the stop words of the text under the clean_reason column. Finally, the rows containing the null values of the clean_reason column were removed from the df_final dataframe.

The final df_final dataframe was exported as a CSV file named, df_final.csv.

Initial Findings:

Table 1: Value Count of Event Type (type) Column in df_final

Event Type	Count
Recall	56955
Safety alert	3821
Field Safety Notice	992
Recall / Safety Alert	14

Table 2: Value Count of Event Risk Class (action_classification) Column in df_final

Event Risk Class	Count
Class 2	41373
Class 3	6896
Class 1	3996

Figure 3: Event Types Recorded by Country Medical Device Originated From

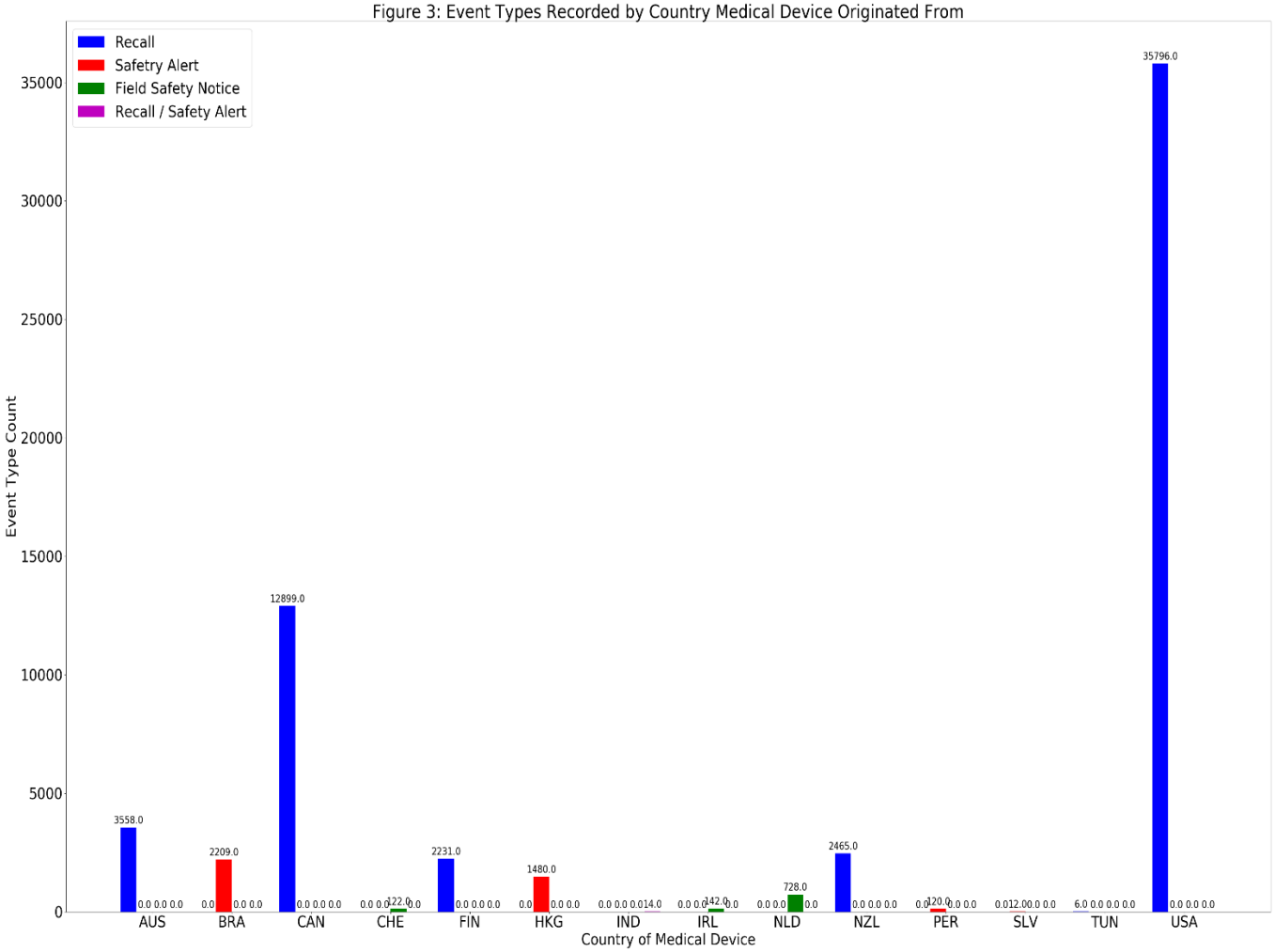
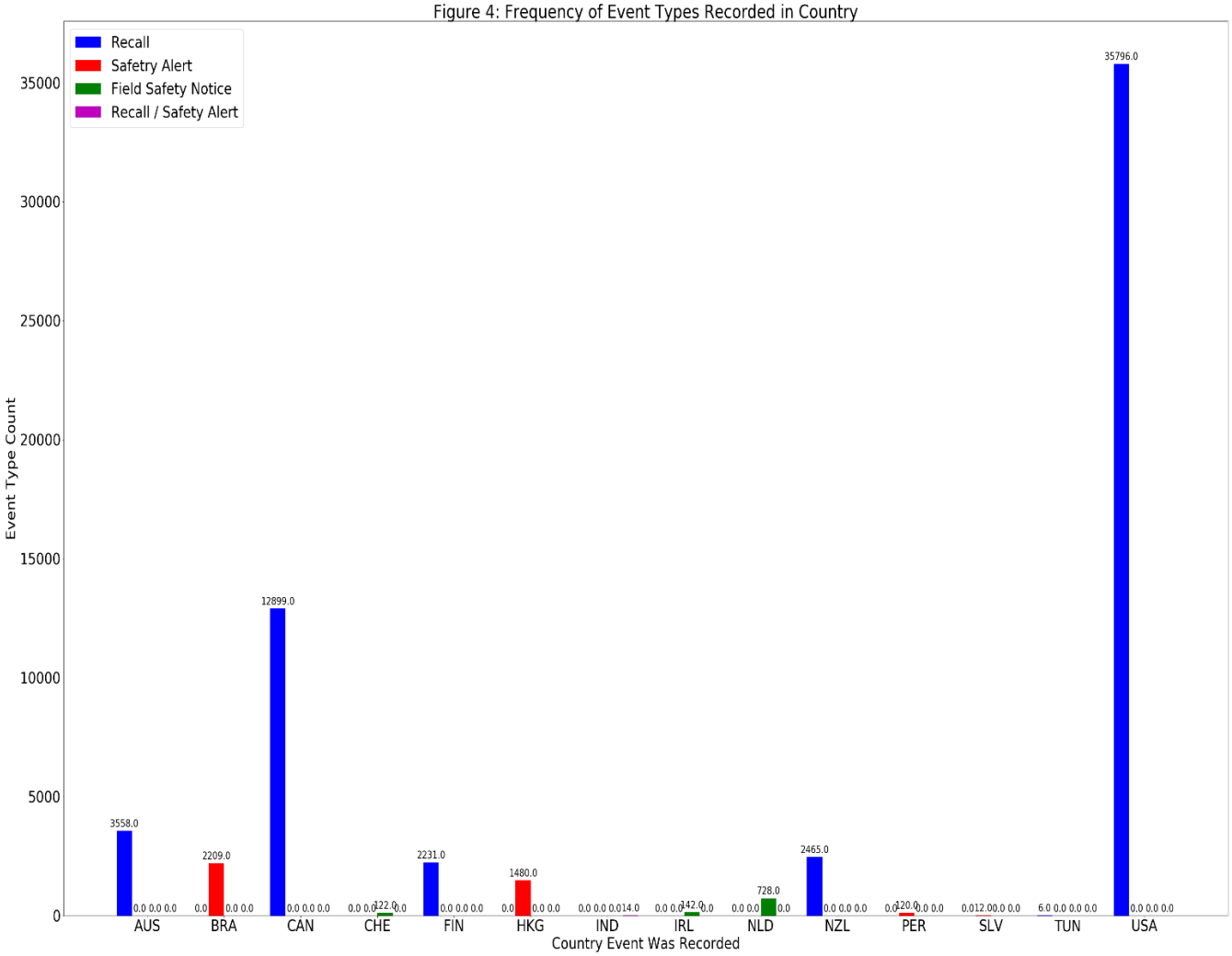


Figure 4: Frequency of Event Types Recorded in Country



Both Figure 3 and Figure 4 show the exact same numbers of each event type being recorded for each country. Perhaps, this is indicative that the devices were used in the same country that they were made from. In other words, they were likely never exported to other countries to use on their patients. Another interesting fact to take note of from Figure 3 and Figure 4 is that the USA has the largest number of recalls for its medical devices than any other country. This might be considered unusual for a developed country. However, it could be for this very reason that more recalls on medical devices were recorded as the USA could have more strict medical policies concerning health hazard and safety. Also supported by Table 1 and Table 2, "Recall" is the event type that is most recorded in the df_final dataset, yet Class 1 which is considered the event risk class with the highest severity in health risk is the least recorded. Thus, there are likely other factors to take into consideration for a medical device to be recalled other than health safety and health hazard. Perhaps the reason column in the df_final dataset could have those other factors to accurately predict what event type would likely occur for a medical device via NLP methods.

Figure 1: 100 Most Frequent Words of "reason" in df_final

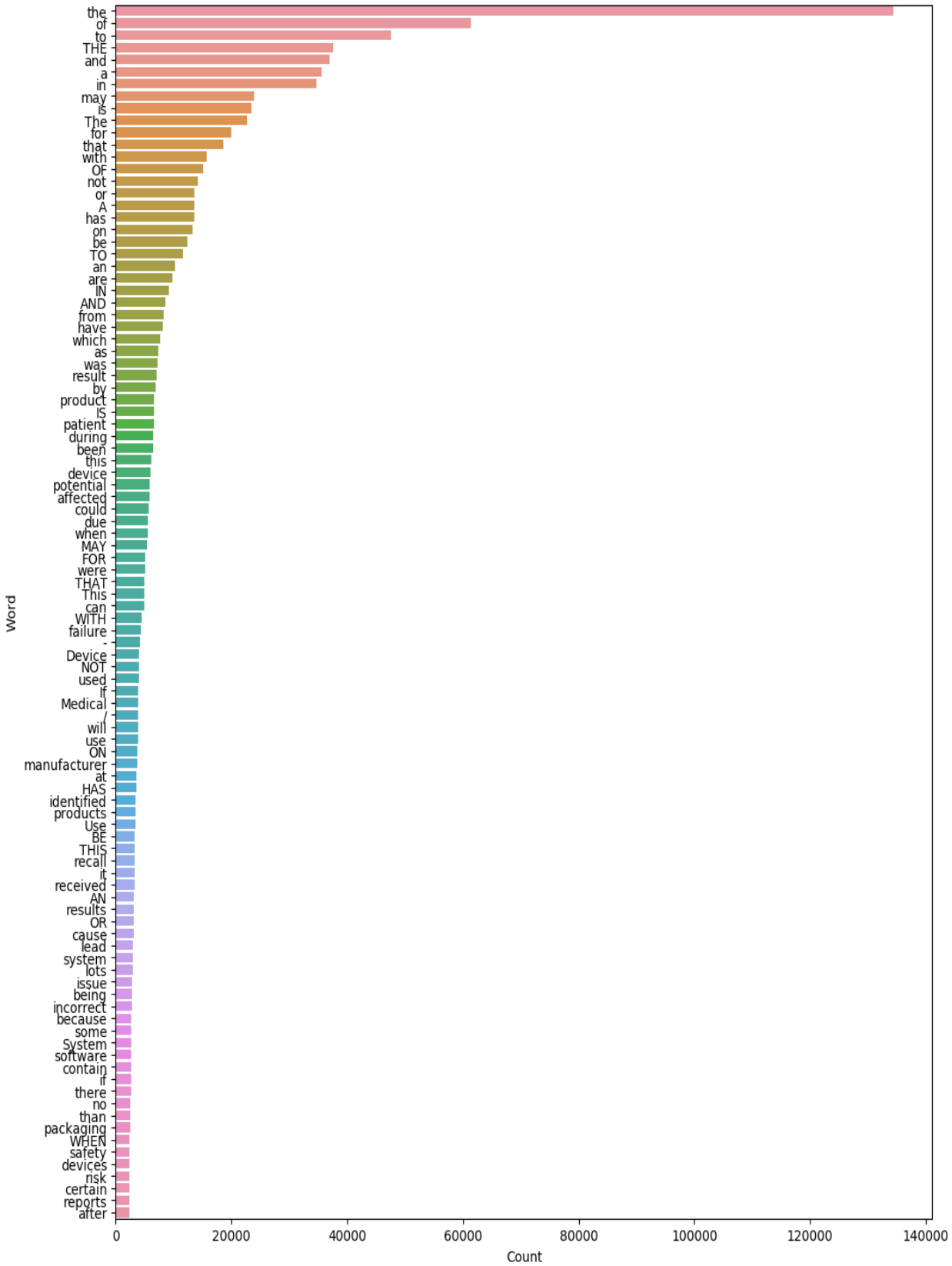
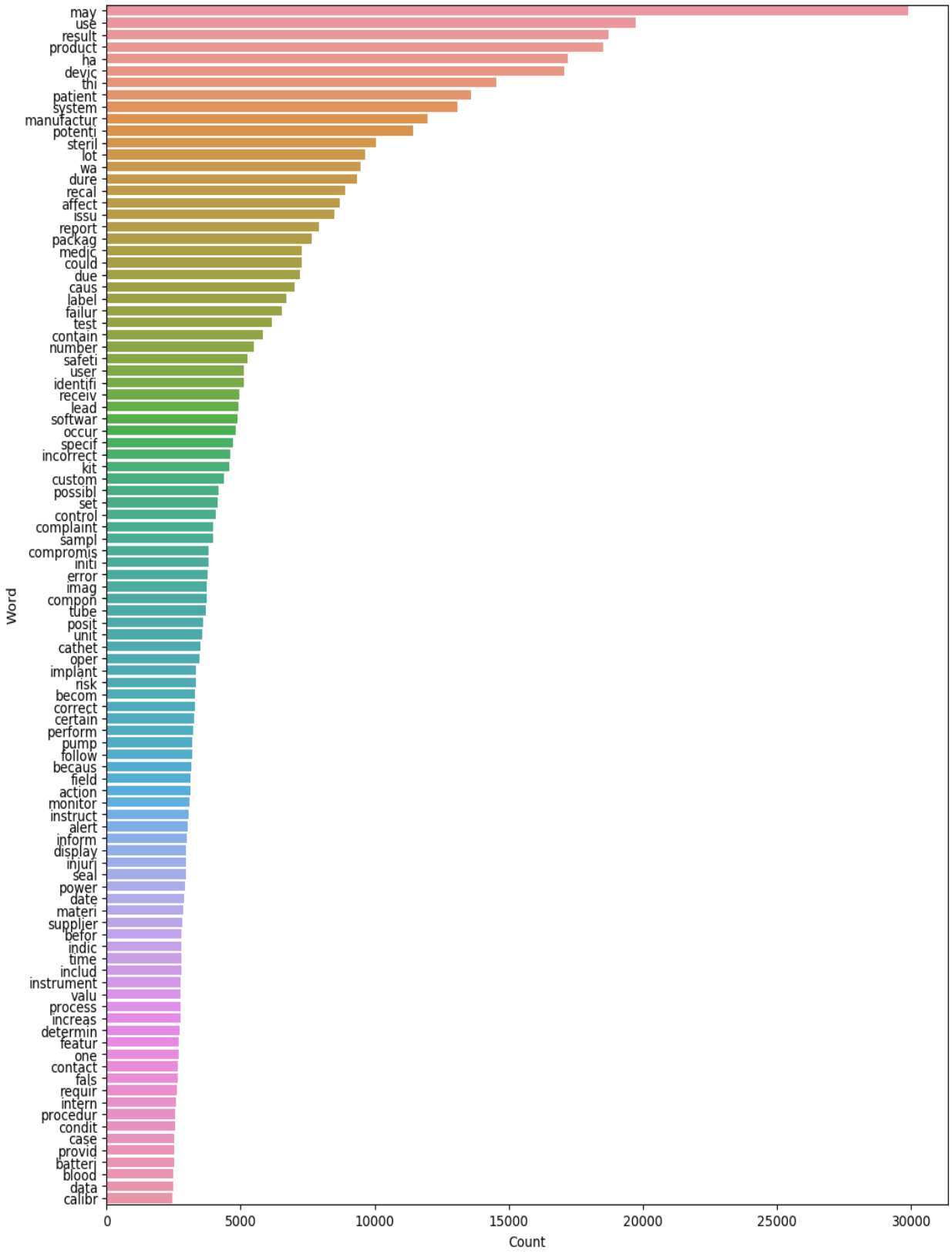


Figure 2: 100 Most Frequent Words of "clean_reason" in df_final



According to Figure 1 and 2, the word frequency plots of both the reason and clean_reason columns of the df_final dataframe show how well the text cleaning methods worked in removing the punctuations and stop words of the text as well as stemming them. Also, Figure 2 shows how much more meaningful words appear in the word frequency count of the text from the reason column after text cleaning was applied to it.

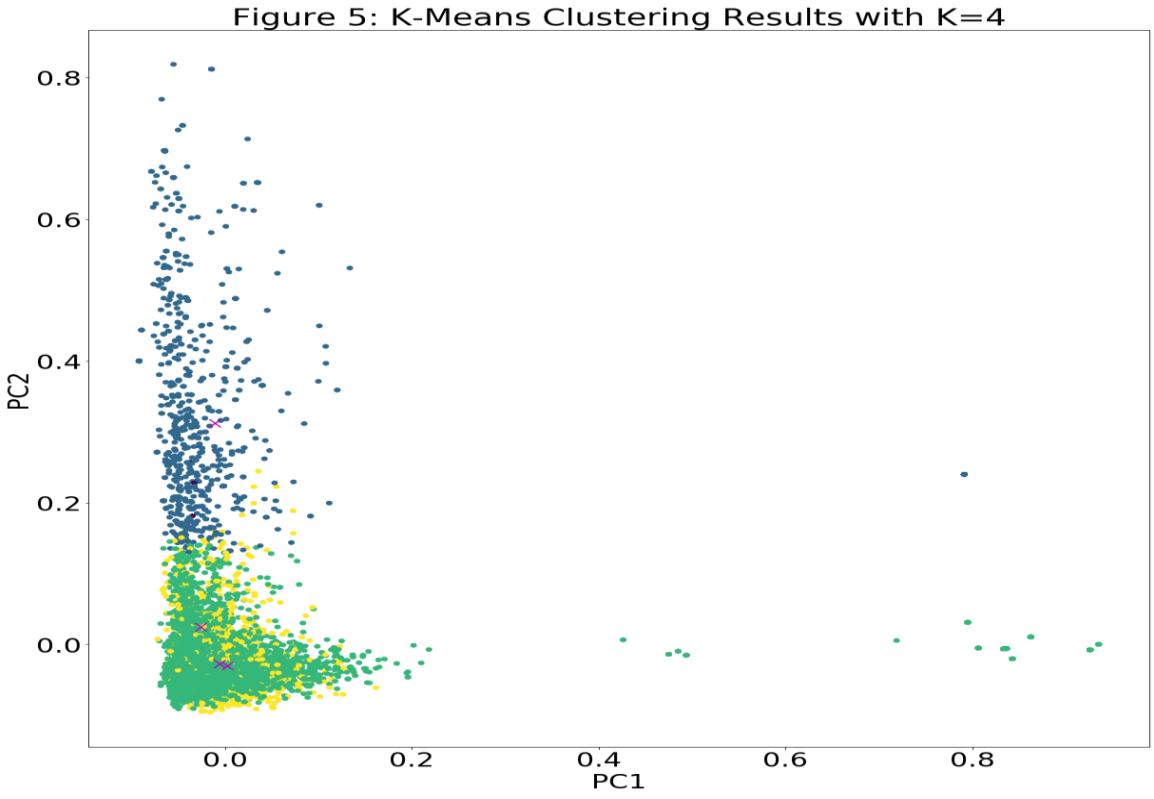
Two unsupervised learning algorithms were used for analyzing the text classification on the clean_reason column in the dataset of df_final.csv. These algorithms were K-Means clustering and Singular Value Decomposition (SVD). They were used since K-Means clustering can handle big data well due to its linear time complexity, and SVD is popular in the field of natural language processing (NLP) to create a representation of the large yet sparse word frequency matrices. Due to limitations of the RAM memory space of the computing computer, the df_final dataset was reduced from 9 columns with 61,799 rows to 9 columns with 14,512 rows by randomly dropping 83% of the rows that contain the "Recall" value under the type column as there are more "Recall" values under the type column than the other three categorical values of "Safety alert," "Field Safety Notice," and "Recall / Safety Alert" combined. Below documents how these inferential statistics techniques were used to analyze the data of df_final.csv and the inferences made based on the results.

K-Means clustering was first done on the cleaned text under the clean_reason column of the df_final dataframe. In order to do the analysis, TF-IDF (term frequency–inverse document frequency) values were computed to vectorize the text of the clean_reason column in order to create the features and X matrices. Then, the K-Means clustering algorithm was applied to the vectorized text. The number of clusters in the algorithm was set to 4 since there are 4 event types in the type column in the df_final dataset. The centroids and features of the K-Means clustering analysis were also calculated. Table 3 below shows which clusters the centroids, based on the words of the text in the clean_reason column, belong to.

Table 3: Centroids in the Four K-Means Clusters

Cluster	Centroids
0	may result use product patient system ha thi potenti dure
1	devic affect manufactur medic safeti product

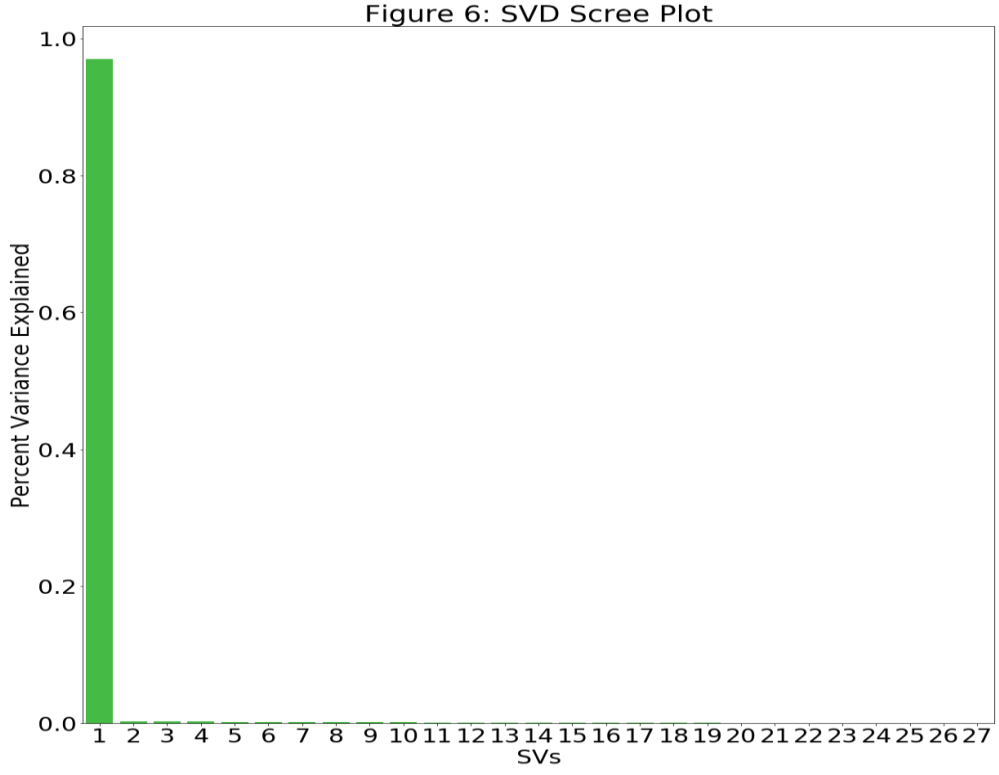
	alert supplier pleas ha
2	steril packag compromis product seal may integr pouch due barrier
3	featur befor failur devic use manufactur materi pack compon label



The K-Means clustering plot in Figure 5 was made by creating an instance of K-Means and then using the PCA function in Python 3 to reduce the calculated features and the cluster centers to 2D. According to Figure 5, the scatterplot shows the points overlap instead of forming into four distinct clusters. A homogeneity score was used to check how good the clustering model of the labelled text data of the clean_reason column is in Figure 5. The homogeneity score ranges between 0 and 1 where 1 stands for perfectly homogeneous labeling. The score was calculated to be about 0.129. As it is close to 0, this indicates that the labeling of the text data under the clean_reason column based on the discrete values of the type column is not very homogenous. Even if the Silhouette Coefficient is generally used for unlabeled data, it is also used to check the clustering model in Figure 5. The best value of the Silhouette Coefficient is 1 while the worst value is -1, and values near 0 indicate overlapping clusters. As the Silhouette Coefficient was calculated to be about 0.006, the clusters very much overlap rather than being distinct from each other. The calculations of K-Means so far indicate that the clean_reason column in the df_final dataset does not have the distinct factors to accurately predict what event type would likely occur for a medical device.

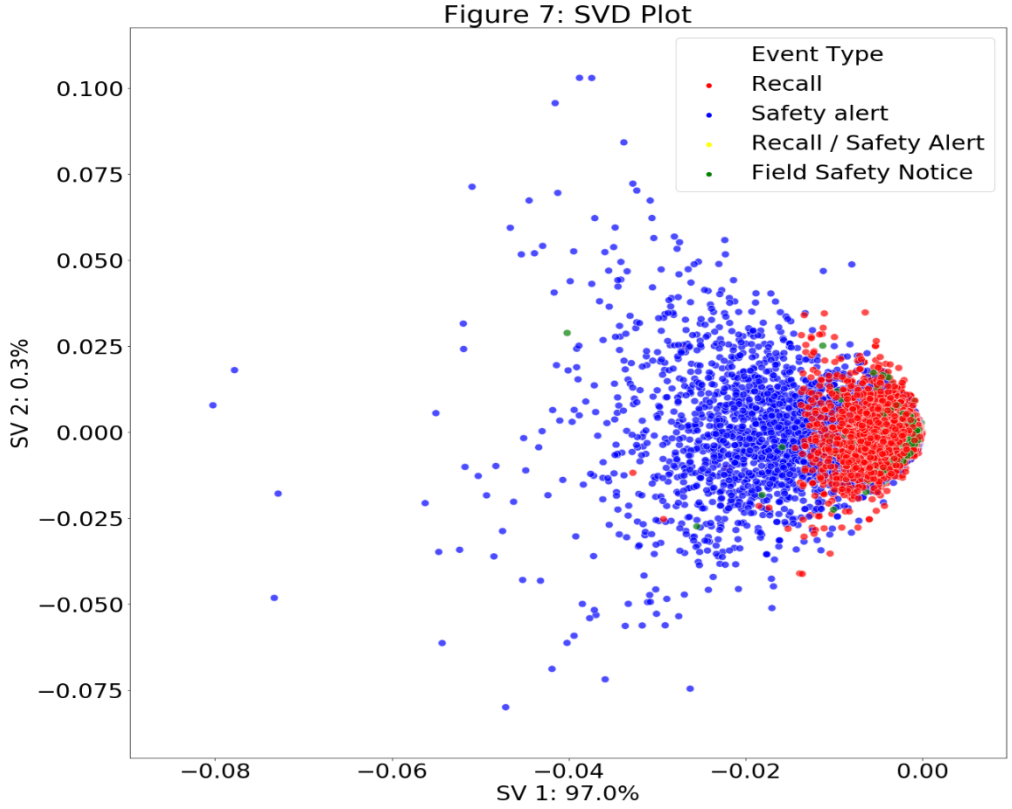
Single Value Decomposition (SVD) was also used to analyze the text under the clean_reason column in the df_final dataset. The bag-of-words approach was used to create a sparse word matrix of the clean_reason column to be analyzed by SVD. NumPy's linalg module's svd function was used to do the SVD with "full_matrices" set to True to get all singular vectors. SVD created three matrices denoted as u, s, and v. The matrices u and v contain singular vectors while s contains singular values. A scree plot was created to visualize the percent variance explained by each singular vector or PC (principal component) and to understand the structure of the text data in the clean_reason column in the df_final dataset.

Figure 6: SVD Scree Plot



The scree plot in Figure 6 shows the percentage of variance explained by each singular vector, and the first vector in Figure 6 explains most of the variation in the text data of the clean_reason column. To better visualize the SVD analysis, a scatter plot of the SVD was created. This was done by creating a data frame containing the first two singular vectors and the meta data from the type column in the df_final dataset before making a scatterplot of this dataframe.

Figure 7: SVD Plot



According to Figure 7, SVD also calculated the data of the text under the clean_reason column of the df_final dataframe to largely overlap and not form any distinctive clusters.
So far, K-Means clustering and SVD show that the text under the clean_reason column does not have any distinct classifications to properly predict the event type to likely occur for a medical device. However, there might be a few more statistical methods to properly categorize the text data of the clean_reason column to accurately predict the event type occurring for a device. Perhaps performing k-fold cross validation and a multi-label classification method would improve the classification of the text data in the clean_reason column in the df_final dataset to create a prediction model that accurately predicts the negative event type that would happen to a medical device.

In-Depth Analysis and Prediction Model:

For this unsupervised learning project, multi-label classification in NLP was used to create a prediction model to predict what event type would likely occur for a medical device based on the recorded reasons of why the medical device was originally put under investigation. K-fold cross validation was utilized to ensure the performance of the prediction model. This

section documents how in-depth analysis was used to create the prediction model via multi-label classification with NLP performed on the textual reason column from the dataset in the df_final.csv and with the type column from the same dataset being denoted as the target variable.

Before the prediction model was created, the data from the df_final dataset underwent the k-fold cross validation procedure as well as needed its text data and target variable converted into features and formats respectively that can be used in the prediction model. Basically, cross validation is a resampling technique used to evaluate machine learning models on a limited data set, and k-fold cross validation is the most common cross validation procedure. The k-fold cross validation procedure has a single parameter called k that represents the number of sub-groups that a given dataset is to be split into. In this project, the k-fold cross validation method was used over the simple train/test split due to the method generally having a less biased or less optimistic estimate of the model's skill. This method works by randomly dividing the set of the data's observations into approximately equal-sized k groups, or folds. The first fold is treated as the validation (test) set while the method is fitted on the remaining k − 1 folds (training set). To prepare the df_final dataset for the k-fold cross validation procedure, the clean_reason column which contains the cleaned text data of the reason column and the type column were copied from the df_final dataset to the sub-dataframe denoted as data. The clean_reason column and the type column in the data sub-dataframe were renamed "text" and "class" respectively. To have the sub-dataframe being randomized further during the k-fold cross validation procedure, the index of the data sub-dataset was randomly permutated.

```
#Preparing data for k-fold cross-validation
data = df_final[['clean_reason', 'type']].copy()
data = data.rename(columns={'clean_reason': 'text', 'type': 'class'}) #rename
the columns in data
data = data.reindex(np.random.permutation(data.index))
```

Next, the features of the text in the text column were extracted (reducing the mass of unstructured data into some uniform set of features that the algorithm can learn from). For text classification, word count frequencies were used on the corpus (the union collection of texts) in correspondence to each class of the event types. The CountVectorizer function of Python is perfect for the feature extraction as it converts a collection of text documents into a matrix of token counts. In order to further improve the results of the prediction model, more features were extracted from the text data in the text column of the data sub-dataset. This was done by utilizing n-gram counts instead of simply relying on word counts which uses bag-of-words features (All of the words are tossed into a "bag" and counted without regard to any contextual meaning that could be associated with the ordering of words). An n-gram is a sequence of ordered words of length n. For example, in the sentence, "Please, turn in your homework," a 2-gram (or bigram) sequence will be "please turn," "turn your," or "your homework" while a 3-gram (or trigram) sequence will be "please turn your" or "turn your homework". The CountVectorizer function can include any order of n-grams by giving it a range. For the data sub-dataset, bigrams seemed to give the k-fold cross validation algorithm the best boost in accuracy. Even if trigrams gave a bit more accuracy to the algorithm, it incurred a longer computation time which made it not worth the infinitesimal increase. With word count frequencies extracted as features, the next step would

be training a classifier and classifying the corpus. A naïve Bayes classifier was used to do so as it assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature, given the class variable. In other words, each feature (in this case word counts) is independent from every other feature, and each one contributes to the probability that an example belongs to a particular class. The MultinomialNB function in Python was used as the classifier and trained by passing in the feature vector and the target vector (the classes that each example belongs to). The indices of each vector must be aligned, but Pandas automatically does that. The Pipeline function from the Scikit Learn library was used to merge the feature extraction and classification into one operation. In other words, the Pipeline function pipelines a series of steps into one object which can be trained and then used to make predictions.

```
#Create a pipeline using a naïve Bayes classifier
pipeline = Pipeline([
  ('vectorizer',  CountVectorizer(ngram_range=(1, 2))), #extract more
features from the text by implementing bigram (2-grams) counts
  ('classifier',  MultinomialNB())])
```

With the features extracted from the data sub-dataset and classified, the k-fold cross validation was ready to be utilized. Scikit Learn's KFold was used to perform the k-fold cross validation by generating k pairs of bitmasks – vectors of booleans which were used to select out randomized portions of the dataset for the train and test sets. The shuffle option in the KFold function was set to true in order to shuffle the data before splitting it into k parts. After the data sub-dataset was shuffled, it was split into k parts. Then, one of the parts was held out while the others were combined. Finally, the k-fold cross validation trained on the combined parts and then validated against the held-out portion. The process was repeated k times (number of folds), holding out a different portion each time. In this project, the value of k was fixed to 10 as it is a value that has been found through experimentation to generally result in a model skill estimate with low bias and a modest variance.

```
#Performing k-fold cross-validation
k_fold = KFold(n_splits=10, shuffle=False, random_state=None) #set 10 folds
to the cross-validation
scores = []
for train_indices, test_indices in k_fold.split(data):
  train_text = np.asarray(data.iloc[train_indices]['text'])
  train_y    = np.asarray(data.iloc[train_indices]['class'])

  test_text  = np.asarray(data.iloc[test_indices]['text'])
  test_y     = np.asarray(data.iloc[test_indices]['class'])

  pipeline.fit(train_text, train_y)
  score = pipeline.score(test_text, test_y)
  scores.append(score)
```

```
score = sum(scores) / len(scores)
```

Scikit Learn models provide a score method that calculates the mean accuracies of each fold which are then averaged together for a mean accuracy of the entire k-fold cross validation set.

```
#Check the mean accuracy score of the k-fold cross-validation
score
```

```
0.94851247432078
```

The mean accuracy of the k-fold cross validation in this project was calculated to be about 95%. This means that the train and test sets created from the k-fold cross validation are about 95% accurate overall. This is a pretty high accuracy which is needed for the NLP prediction model to be created in this project. The train and test sets of the text data were converted into TF-IDF (Term Frequency — Inverse Document Frequency) features.

```
#Using TF-IDF to extract features from the cleaned version of the reason data
tfidf_vectorizer = TfidfVectorizer(max_df=0.8, max_features=10000) # set 10,0
00 most frequent words in the data as features

#Create TF-IDF features
xtrain_tfidf = tfidf_vectorizer.fit_transform(train_text)
xval_tfidf = tfidf_vectorizer.transform(test_text)
```

This means that the words in the text were quantified based on the computed weight of each word which signifies the importance of the word in the corpus. The train and test sets of the target variable were also formatted to be one-hot encoded by sklearn's MultiLabelBinarizer function which transforms a list of sets or tuples into the supported multilabel format - a (samples x classes) binary matrix indicating the presence of a class label. This was done by converting the train and test sets of the target variable into dataframes before splitting their values into separate lists.

```
#Convert train_y and test_y into datasets
df_train_y=pd.DataFrame(train_y, columns=['new_type'])
df_test_y=pd.DataFrame(test_y, columns=['new_type'])

#Converting values in df_train_y and df_test_y into lists with ',' replacing
'/'
df_train_y['new_type'] = df_train_y['new_type'].str.split(' / ')
df_test_y['new_type'] = df_test_y['new_type'].str.split(' / ')
```

Then, they were fitted and transformed by the MultiLabelBinarizer function into the binary matrix format that can be used in the prediction model for classifying the text data.

```
#One hot encode the target variable, i.e., type by using sklearn's MultiLabel
Binarizer( )
multilabel_binarizer = MultiLabelBinarizer()
multilabel_binarizer.fit(df_train_y['new_type'])
multilabel_binarizer.fit(df_test_y['new_type'])
```

```
MultiLabelBinarizer(classes=None, sparse_output=False)

# transform target variable
ytrain = multilabel_binarizer.transform(df_train_y['new_type'])
yval = multilabel_binarizer.transform(df_test_y['new_type'])
```

With the features of the text data (clean_reason column from the df_final dataset) extracted, and the target variable (type column from the df_final dataset) formatted, the NLP prediction model is ready to be created. Building a model for every one-hot encoded target variable would take a considerable amount of time. Thus, a Logistic Regression model would be built as it is quick to train on limited computational power with sklearn's OneVsRestClassifier class used to solve the model's problem as a Binary Relevance or a one-vs-all problem.

```
#Use sk-learn's OneVsRestClassifier class to solve this problem as a Binary
Relevance or one-vs-all problem
lr = LogisticRegression()
clf = OneVsRestClassifier(lr)
```

The model was fitted on the train sets of the TF-IDF features and the formatted target variable.

```
#Fit model on train data
clf.fit(xtrain_tfidf, ytrain)
```

After the model was trained, it was used to make predictions with the validation (or test) TF-IDF features set.

```
#Make predictions for validation set
y_pred = clf.predict(xval_tfidf)
```

Displaying a sample of the predictions only shows it as a binary one-dimensional array (or the one-hot encoded form of the event types tags).

```
#Check out a sample from these predictions
y_pred[3]
```

```
array([0, 1, 0])
```

Fortunately, sklearns' inverse_transform function and the MultiLabelBinarizer object helped to convert the predicted arrays into event type tags.

```
#Convert the predicted arrays into event type tags
multilabel_binarizer.inverse_transform(y_pred)[3]
```

```
('Recall',)
```

To evaluate the model's overall performance, all of the predictions and the entire target variable of the validation set would need to be taken into consideration. The F1 score, also known as balanced F-score or F-measure, was used to evaluate the overall performance of the prediction model. The F1 score is interpreted as a weighted average of the precision and recall where 1 is its best score, and 0 is its worst score. For the F1 score, the relative contribution of precision and recall are equal. The formula for the F1 score is:

F1 = 2 x (precision x recall) / (precision + recall)

In the multi-class and the multi-label case, the F1 score is the average of each class with weighting depending on the average parameter.

```
# evaluate performance
f1_score(yval, y_pred, average="micro")
```

```
0.9596819988642816
```

The F1 score was calculated to be about 0.96. This means that the prediction model that was created in this project has nearly perfect precision and recall. Since the prediction model has a nearly perfect F1 score, an inference function was created to take any text describing the reasons for a medical device being put under investigation as input and to give the predicted outcome of the negative event type(s) (Field Safety Notice, Recall, and/or Safety Alert) that would take place for the medical device. First, the type column in the df_final dataset was split to convert the values in the column into individual lists which were stored into a new column called new_type.

```
#Create new_type column in df_final dataset
df_final['new_type'] = df_final['type'].str.split(' / ')
```

Then the text cleaning functions that were created in Capstone Project 1 Dataset - Final-Edited.ipynb were loaded.

```
# function for text cleaning
def clean_text(text):
    # remove everything except alphabets
    text = re.sub("[^a-zA-Z]"," ",text)
    # remove whitespaces
    text = ' '.join(text.split())
    # convert text to lowercase
    text = text.lower()

    return text

# function for stemming words
def stem_text(text):
    ps = PorterStemmer()
    token_words=word_tokenize(str(text))
    token_words
    stem_text=[]
    for word in token_words:
        stem_text.append(ps.stem(word))
        stem_text.append(" ")
    return "".join(stem_text)

# function to remove stopwords
```

```
def remove_stopwords(text):
    stop_words = set(stopwords.words('english'))
    no_stopword_text = [w for w in str(text).split() if not w in stop_words]
    return ' '.join(no_stopword_text)
```

With the formatted type (new_type) column from the df_final dataset, the loaded text cleaning functions, and the prediction model of this project, the infer_event function was created.

```
#Create inference function
def infer_event(q):
    q = clean_text(q)
    q = stem_text(q)
    q = remove_stopwords(q)
    q_vec = tfidf_vectorizer.transform([q])
    q_pred = clf.predict(q_vec)
    return multilabel_binarizer.inverse_transform(q_pred)
```

The infer_event function first cleans the text by removing non-alphabetical texts and white spaces, converting the text to lowercase, stemming the text, and removing stop words before extracting features from the text via the TfidfVectorizer function in Python to make predictions which are returned as negative medical event type tags. The Python code below displays the results of the infer_event function worked on ten samples from the corpus under the reason column in the df_final dataset.

```
for i in range(10):
  k = df_final['reason'].sample(1).index[0]
  print("Medical Device: ", df_final['device_name'][k], "\nPredicted Event
Type: ", infer_event(df_final['reason'][k])), print("Actual Event Type:
",df_final['new_type'][k], "\n")
```

```
Medical Device:  ARCHITECT SYSTEM - PROGESTERONE ASSAY
Predicted Event Type:  [('Recall',)]
Actual Event Type:  ['Recall']

Medical Device:  Single Shot Epidural Anesthesia Kit, Internal Jugular Pun
cture Kit with Blue FlexTip(R) Catheter, Pediatric Jugular Puncture Kit, A
rterial Line Kit, Vessel Catheterization kit, Central Venous Catheterizati
on kit, Jugular Puncture Ks. jne. ks.ilm.,
Predicted Event Type:  [('Recall',)]
Actual Event Type:  ['Recall']

Medical Device:  Device Recall  Cytosponge Cell Collection Device
Predicted Event Type:  [('Recall',)]
Actual Event Type:  ['Recall']

Medical Device:  Device Recall  Howell D.A.S.H. Extraction Balloon
Predicted Event Type:  [('Recall',)]
Actual Event Type:  ['Recall']

Medical Device:  CHECKCELLS (POOLED CELLS)
```

```
Predicted Event Type:  [('Recall',)]
Actual Event Type:  ['Recall']

Medical Device:  RESONATE, VIGILANT X4, PERCIVA, and MOMENTUM CRT-D/ICD
Predicted Event Type:  [('Safety Alert',)]
Actual Event Type:  ['Safety Alert']

Medical Device:  LOW PROFILE ABUTMENTS  INTERNAL AND EXTERNAL CONNECTION
Predicted Event Type:  [('Recall',)]
Actual Event Type:  ['Recall']

Medical Device:  AVANTA FLUID INJECTION SYSTEM - MAIN UNIT WITH PEDESTAL
Predicted Event Type:  [('Recall',)]
Actual Event Type:  ['Recall']

Medical Device:  WASHER AND AUTOMATIC ENDOSCOPE REPAIRER AER - registered
in Anvisa under the number 80145900728 - Code 20301 - Serial numbers / Lot
s: EP1151741; EP1151742; EP1151744; EP1151747; EP1151715; EP1151568; EP115
1390; 4024730; 3024357; EP1151459; EP115119; EP115910; EP1151116.
Predicted Event Type:  [('Recall',)]
Actual Event Type:  ['Safety Alert']

Medical Device:  Device Recall  Giraffe Infant Warmers
Predicted Event Type:  [('Recall',)]
Actual Event Type:  ['Recall']
```

The NLP prediction model especially in the infer_event function looks very serviceable. However, this model was only worked on English medical text. Thus, it would likely have limitations on medical text written in different languages as different languages have different grammatical structures. Plus, as countries have different medical policies and follow different medical systems, there might be different medical terminology and lingo between individual countries. Thus, the prediction model created in this project would most successfully work on English-language medical text that follow the same medical policies and systems for negative event type classifications concerning medical devices. However, might it be possible to train this model to incorporate the different languages and medical terminology of different countries to classify the negative event types that might occur for the medical devices?