NLP CAPSTONE PROJECT – Interim Report

1) *Summary*

Given a dataset with records of accidents from 12 different plants in 03 different countries describing different accidents. We have developed a chatbot which can accurately predict the accident level using the description the user provides. We have tried different vectorizers and different classification models to predict the accident level. This chatbot can be used to effectively predict the accident level and provide required assistance as per the incident description.

GitHub Link:- *https://github.com/SaiCharan99/GL-Capstone-NLP2*

Data cleaning

Shape: There are 425 records and 11 attributes/columns

```
: (425, 11)
```

Missing values: There are no missing values in the data

```
: Date                        0
  Country                     0
  Local                       0
  Industry Sector             0
  Accident Level              0
  Potential Accident Level    0
  Gender                      0
  Employee type               0
  Critical Risk               0
  Description                 0
  dtype: int64
```

Removing duplicate values and unwanted column: There are 7 duplicate rows and 1 unwanted column, after removing those we have final dataset of 418 rows and 10 attributes/columns now.

```
data.duplicated().sum()
```

```
: 7
```

```
data.drop_duplicates(inplace= True )
#droping column Unnamed
data.drop(["Unnamed: 0"], axis=1, inplace=True)
data.shape
```

```
: (418, 10)
```

Type of attributes: All columns are object type. There is no numerical attribute present.

```
: Date                        object
  Country                     object
  Local                       object
  Industry Sector             object
  Accident Level              object
  Potential Accident Level    object
  Gender                      object
  Employee type               object
  Critical Risk               object
  Description                 object
  dtype: object
```

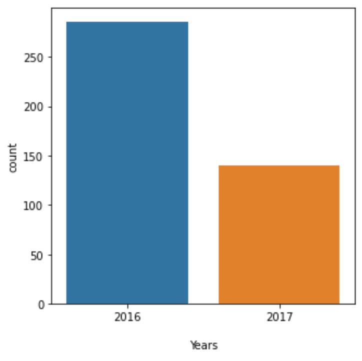
2) Exploratory Data Analysis

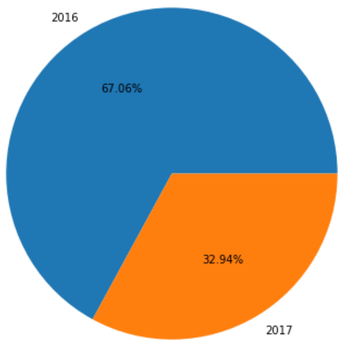
Univariate Analysis: In this, each attribute is individually analysed.

DATE:

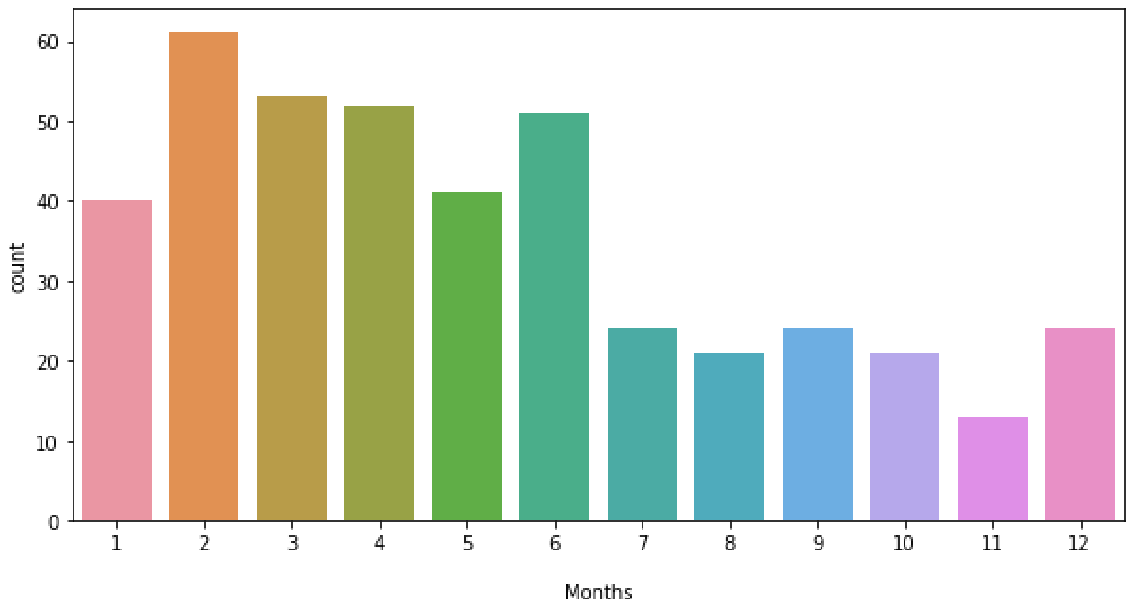
Here, the column name is renamed from Data to Date. Also, the year, month and day variables are extracted from the date column to find if there are any seasonality co-occurrences of accident.

Year-

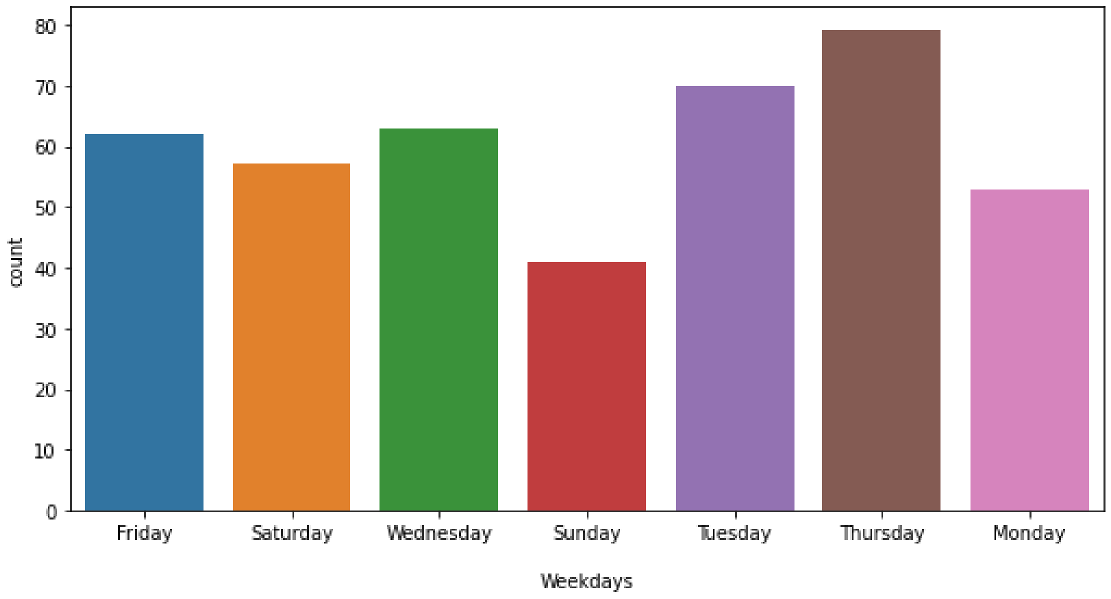




Month-



Day-

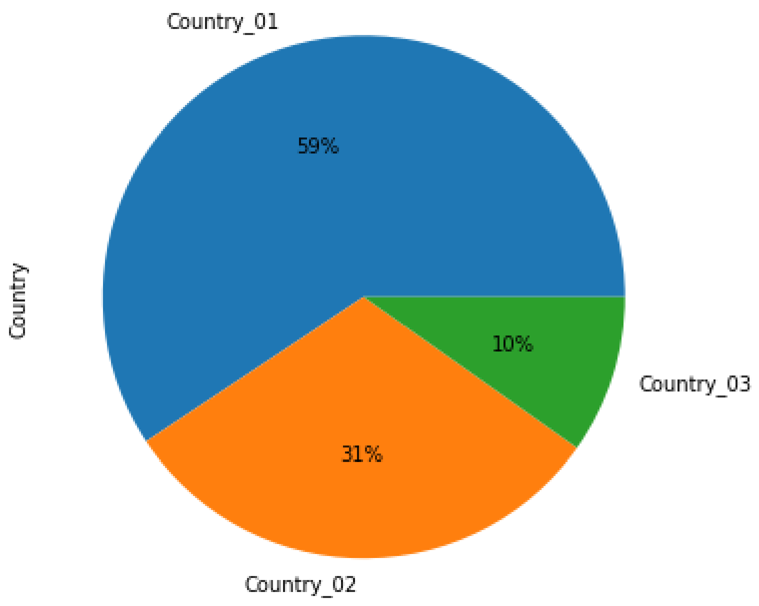


<u>Observations:</u>

- There are lots of records for the year 2016, contributing to approximately 67%, whereas only approximately 33% for the year 2017.
- Most of the incidents are recorded during the first 6 months than the last six months of a year
- Thursday has recorded the highest incidents

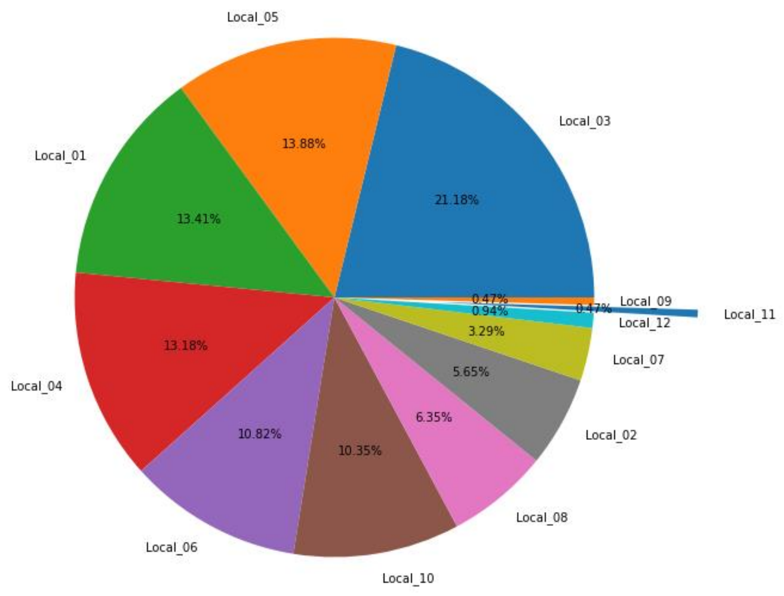
COUNTRY:

Here, the column name is renamed from Countries to Country



Observation: The most affected country from the above dataset is Country_01 with around 59% of the accidents, whereas Country_02 and Country_03 counts for 31% and 10% respectively.

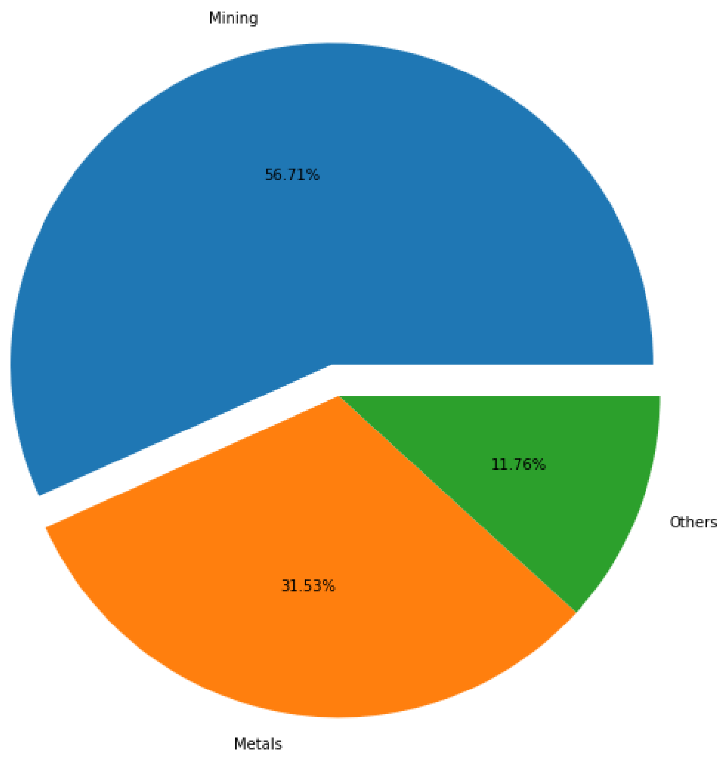
LOCAL:



Observations:

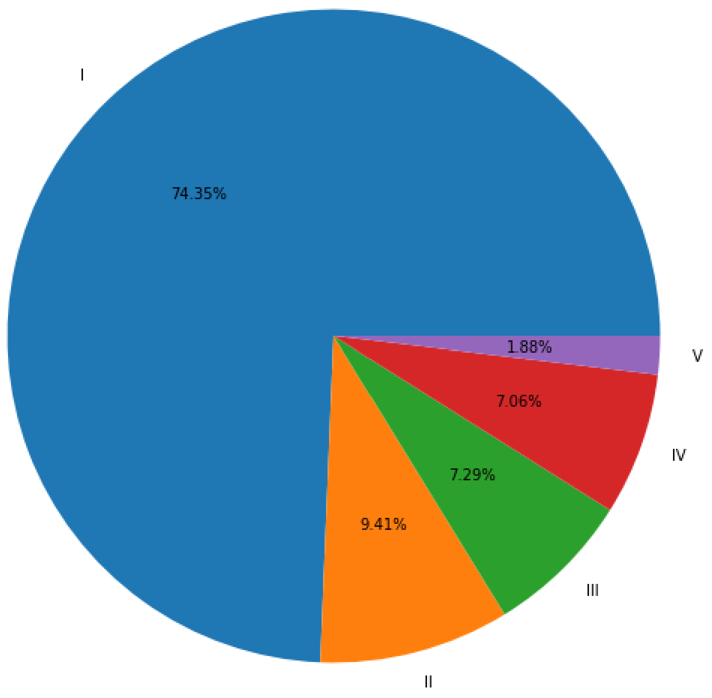
- Highest manufacturing plants are located in Local_03 city.
- Lowest manufacturing plants are located in Local_09 and Local_11 city.

INDUSTRY SECTOR:



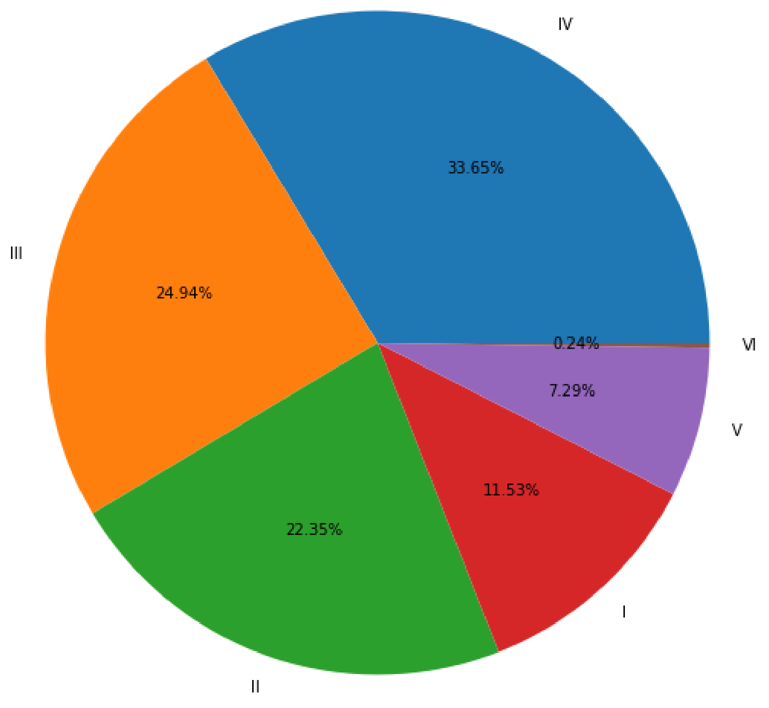
Observation: Most of the manufacturing plants belongs to Mining sector.

ACCIDENT LEVEL:



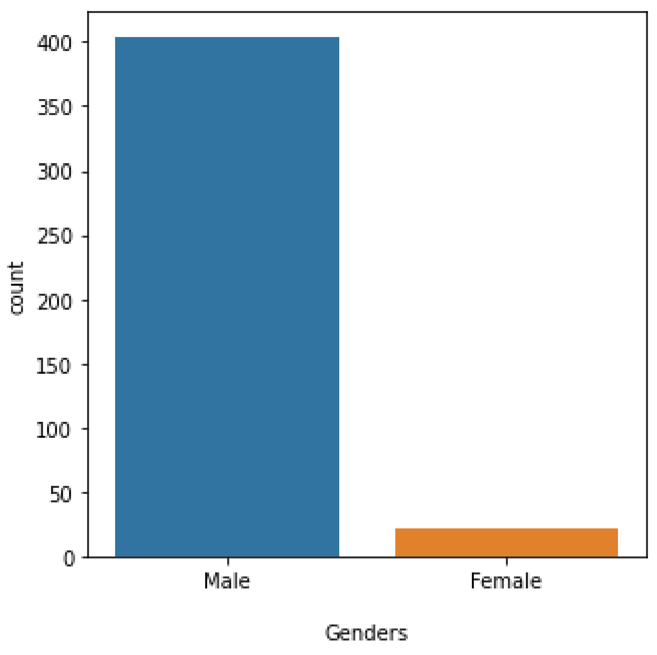
Observation: Most accidents belongs to "Accident Level" I .Its count is 316 which is equivalent to 74.35%% of total accidents

POTENTIAL ACCIDENT LEVEL:



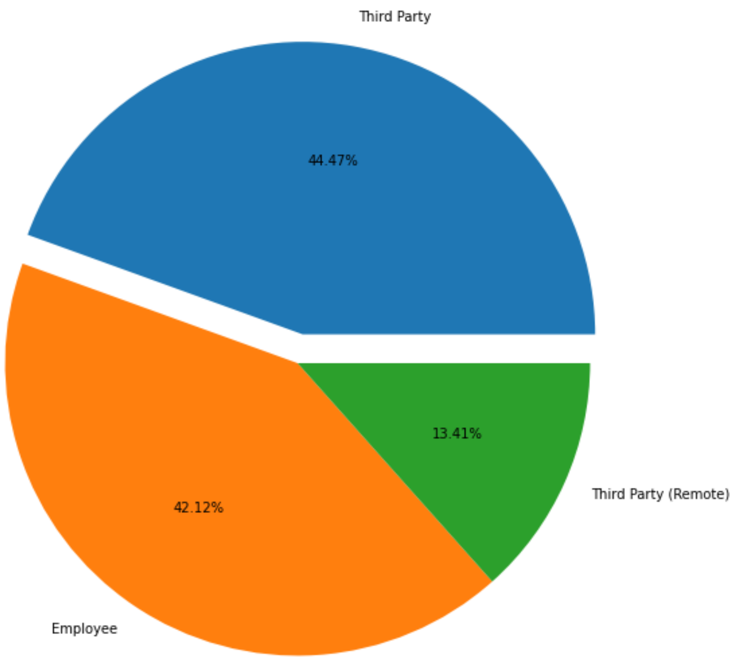
Observation: Most "Potential Accident Level" belongs to level IV .Its count is 143 which is equivalent to 33.65% of total potential accidents.

GENDER:



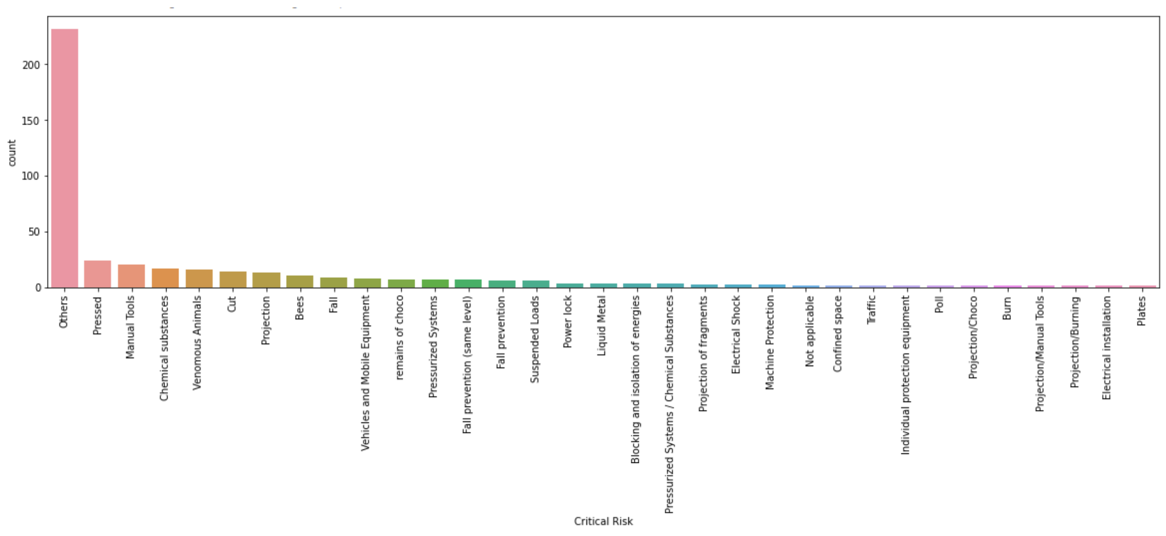
Observation**:** There are more men working in this industry as compared to women.

EMPLOYEE TYPE:



Observation**:** Around 44% Third party employees, 43% own employees and 13% Third party(Remote) employees working in this industry.

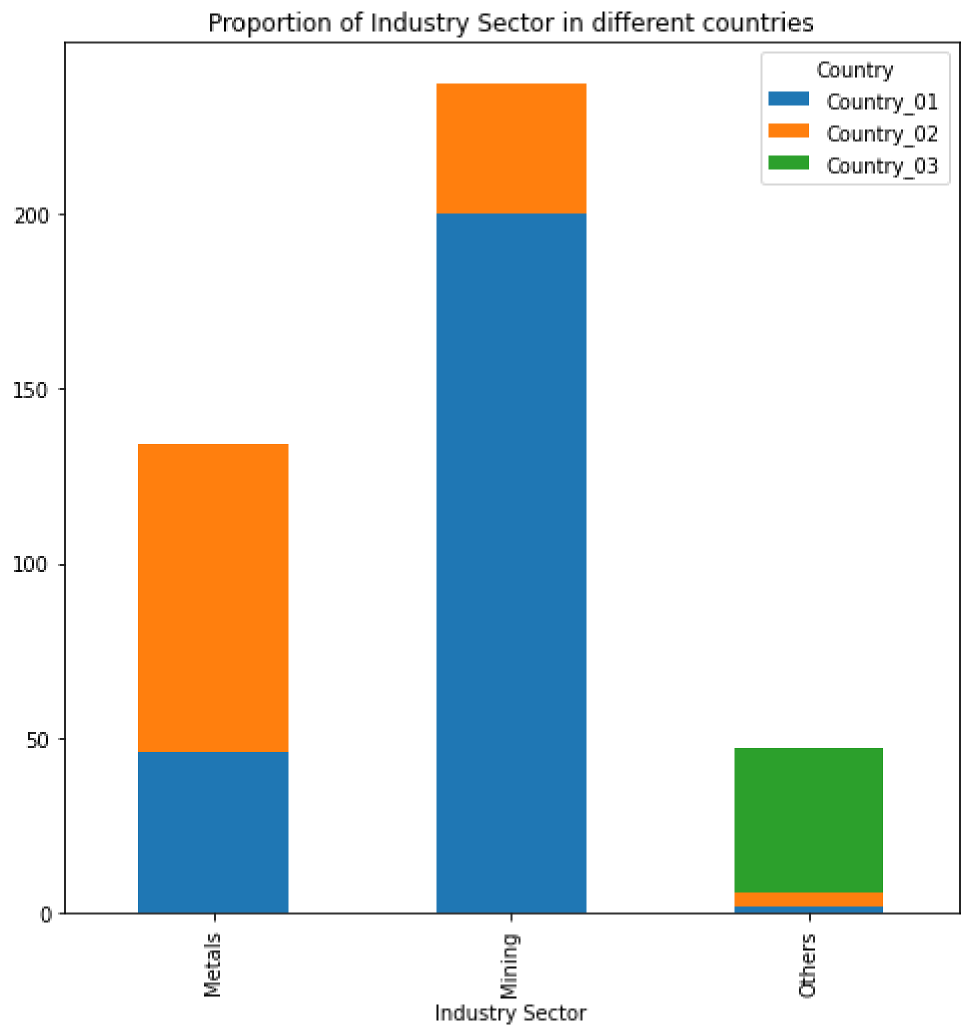
CRITICAL RISK:



Observation: Most of the incidents are registered as 'Others', it takes lot of time to analyse risks and reasons why the accidents occur.

Bivariate Analysis:

Country and Industry Sector:



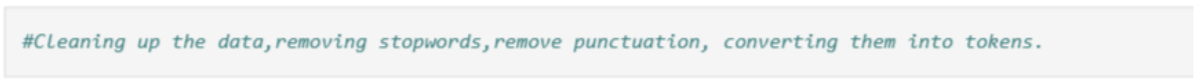
Observations:

- Metals and Mining industry sector plants are not available in Country_03.
- Distribution of industry sector differ significantly in each country.

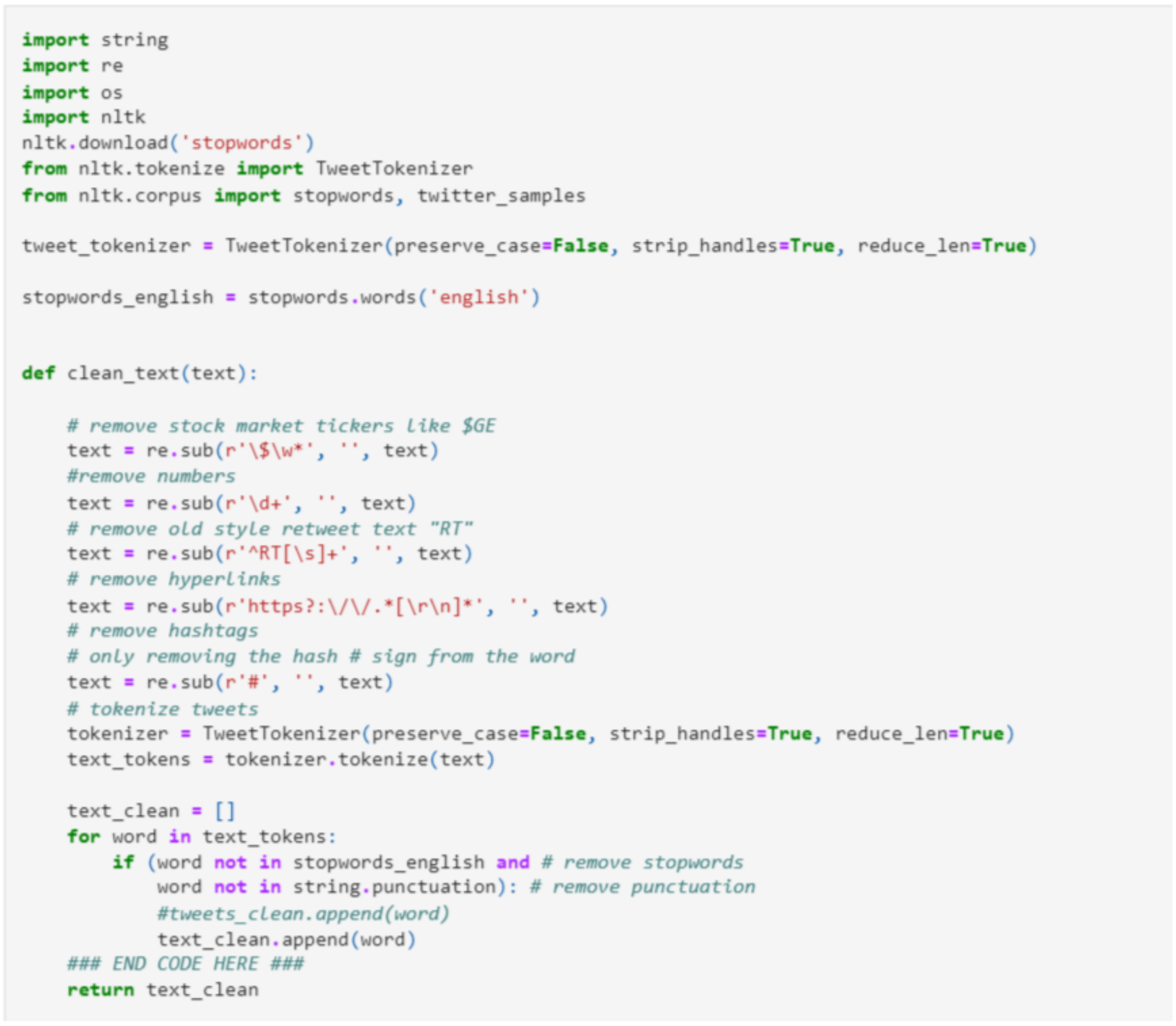
NLP Pre-processing

Here, the main task is to process a language given in Description column in a right way to get the best results.

- Converting to lowercase for all words
- removing stock market tickers like $GE
- removing numbers
- removing old style retweet text "RT"
- removing hyperlinks
- removing hashtags
- only removing the hash # sign from the word
- removing stopwords
- removing punctuation



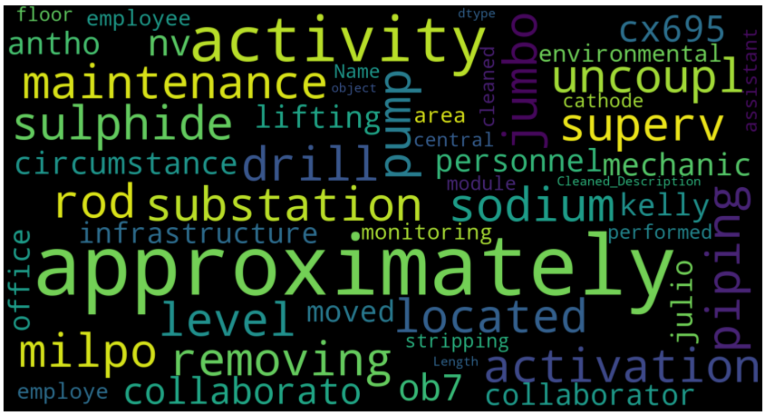
```
#Cleaning up the data,removing stopwords,remove punctuation, converting them into tokens.
```



```
import string
import re
import os
import nltk
nltk.download('stopwords')
from nltk.tokenize import TweetTokenizer
from nltk.corpus import stopwords, twitter_samples

tweet_tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True, reduce_len=True)

stopwords_english = stopwords.words('english')


def clean_text(text):

    # remove stock market tickers like $GE
    text = re.sub(r'\$\w*', '', text)
    #remove numbers
    text = re.sub(r'\d+', '', text)
    # remove old style retweet text "RT"
    text = re.sub(r'^RT[\s]+', '', text)
    # remove hyperlinks
    text = re.sub(r'https?:\/\/.*[\r\n]*', '', text)
    # remove hashtags
    # only removing the hash # sign from the word
    text = re.sub(r'#', '', text)
    # tokenize tweets
    tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True, reduce_len=True)
    text_tokens = tokenizer.tokenize(text)

    text_clean = []
    for word in text_tokens:
        if (word not in stopwords_english and # remove stopwords
            word not in string.punctuation): # remove punctuation
            #tweets_clean.append(word)
            text_clean.append(word)
    ### END CODE HERE ###
    return text_clean
```

WordCloud Analysis

Wordcloud for cleaned description:



<u>Observation:</u> Most words are related to maintenance, accident, employee, equipment, infrastructure.

3) Data Modelling

Long-Short Term Memory (LSTM) with Glove embedding:

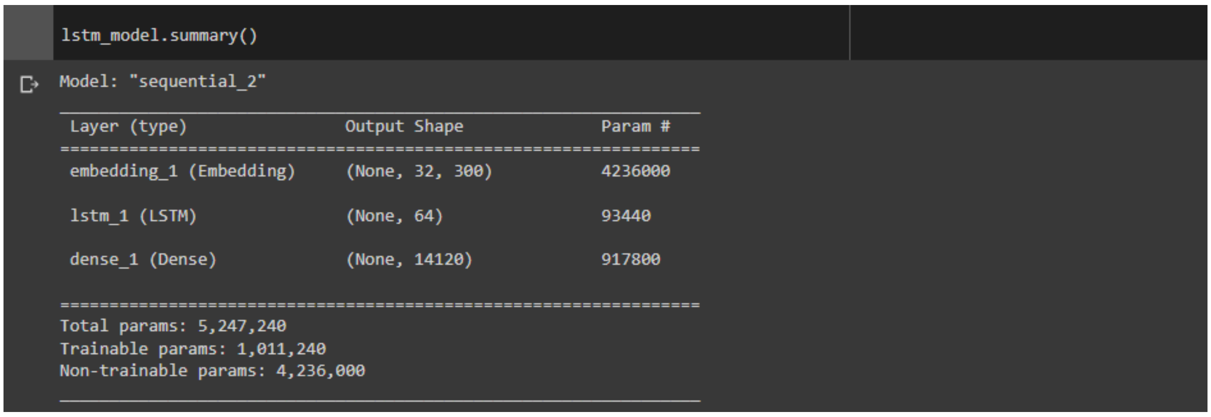
LSTM is a type of Recurrent Neural Network in Deep Learning that has been specifically developed for the use of handling sequential prediction problems.
For example: Weather Forecasting, Stock Market Prediction, Product Recommendation, Text/Image/Handwriting Generation, Text Translation

<u>GLOVE Embedding:</u>

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

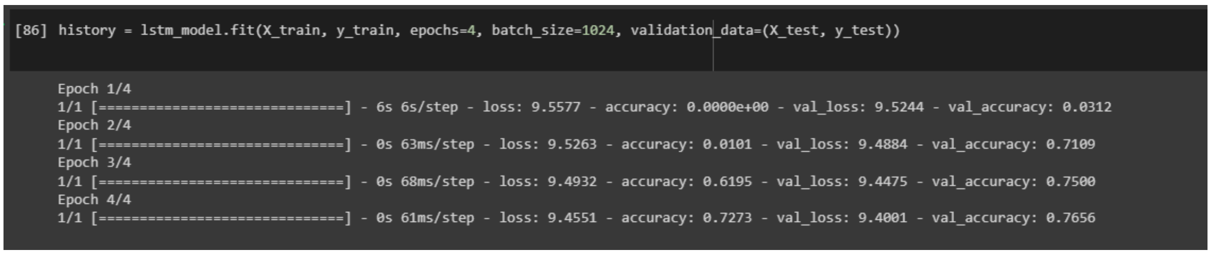
Glove file Used:**glove.6B.300d.txt**

Model Summary:



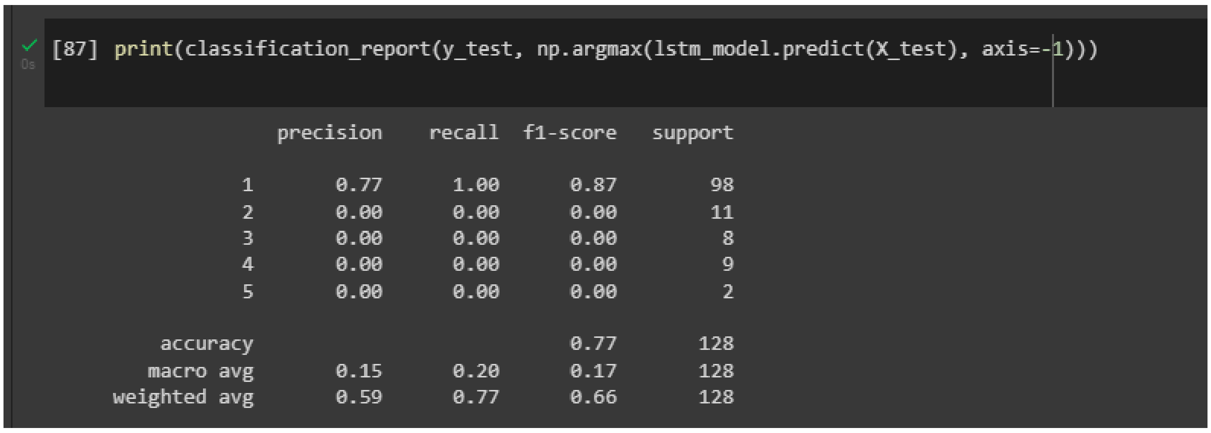
```
lstm_model.summary()

Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 32, 300)           4236000

 lstm_1 (LSTM)               (None, 64)                93440

 dense_1 (Dense)             (None, 14120)             917800

=================================================================
Total params: 5,247,240
Trainable params: 1,011,240
Non-trainable params: 4,236,000
_________________________________________________________________
```

Model Performance:



```
[86] history = lstm_model.fit(X_train, y_train, epochs=4, batch_size=1024, validation_data=(X_test, y_test))

     Epoch 1/4
     1/1 [==============================] - 6s 6s/step - loss: 9.5577 - accuracy: 0.0000e+00 - val_loss: 9.5244 - val_accuracy: 0.0312
     Epoch 2/4
     1/1 [==============================] - 0s 63ms/step - loss: 9.5263 - accuracy: 0.0101 - val_loss: 9.4884 - val_accuracy: 0.7109
     Epoch 3/4
     1/1 [==============================] - 0s 68ms/step - loss: 9.4932 - accuracy: 0.6195 - val_loss: 9.4475 - val_accuracy: 0.7500
     Epoch 4/4
     1/1 [==============================] - 0s 61ms/step - loss: 9.4551 - accuracy: 0.7273 - val_loss: 9.4001 - val_accuracy: 0.7656
```

Classification Report:



```
[87] print(classification_report(y_test, np.argmax(lstm_model.predict(X_test), axis=-1)))

                   precision    recall  f1-score   support

              1       0.77      1.00      0.87        98
              2       0.00      0.00      0.00        11
              3       0.00      0.00      0.00         8
              4       0.00      0.00      0.00         9
              5       0.00      0.00      0.00         2

       accuracy                           0.77       128
      macro avg       0.15      0.20      0.17       128
   weighted avg       0.59      0.77      0.66       128
```

BILSTM with glove Embeddings:

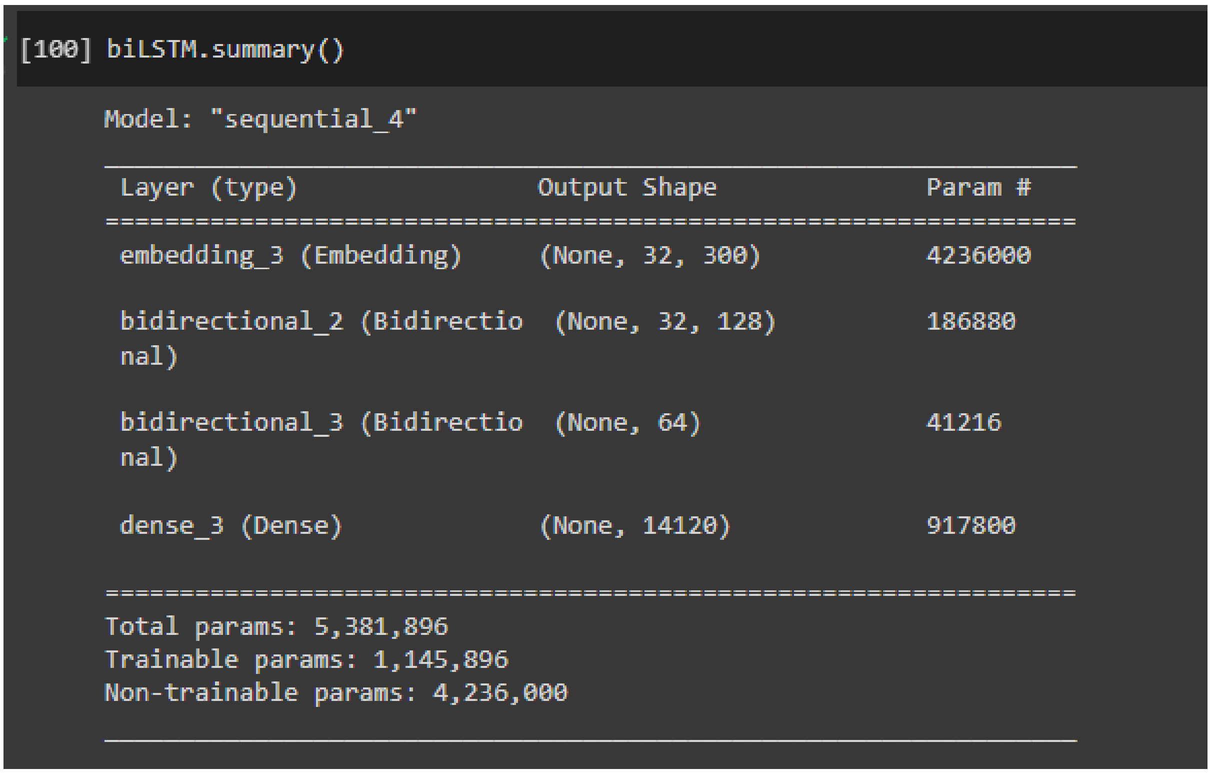
Bidirectional long-short term memory(bi-lstm) is the process of making any neural network o have the sequence information in both directions backwards (future to past) or forward(past to future).

In bidirectional, our input flows in two directions, making a bi-lstm different from the regular LSTM. With the regular LSTM, we can make input flow in one direction, either backwards or forward. However, in bi-directional, we can make the input flow in both directions to preserve the future and the past information

BiLSTM Summary:

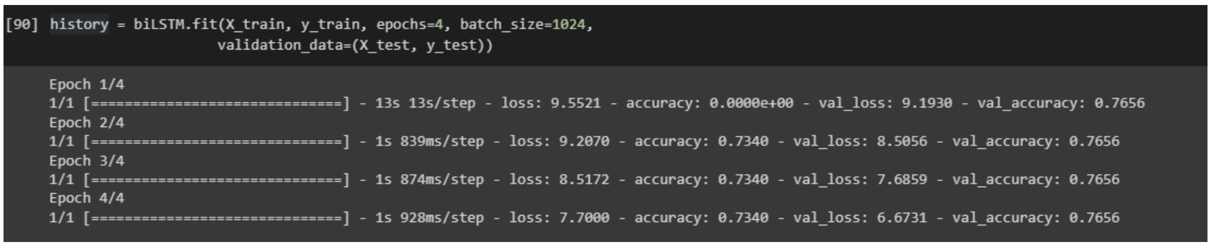
Glove file Used:**glove.6B.300d.txt**

Model Summary:



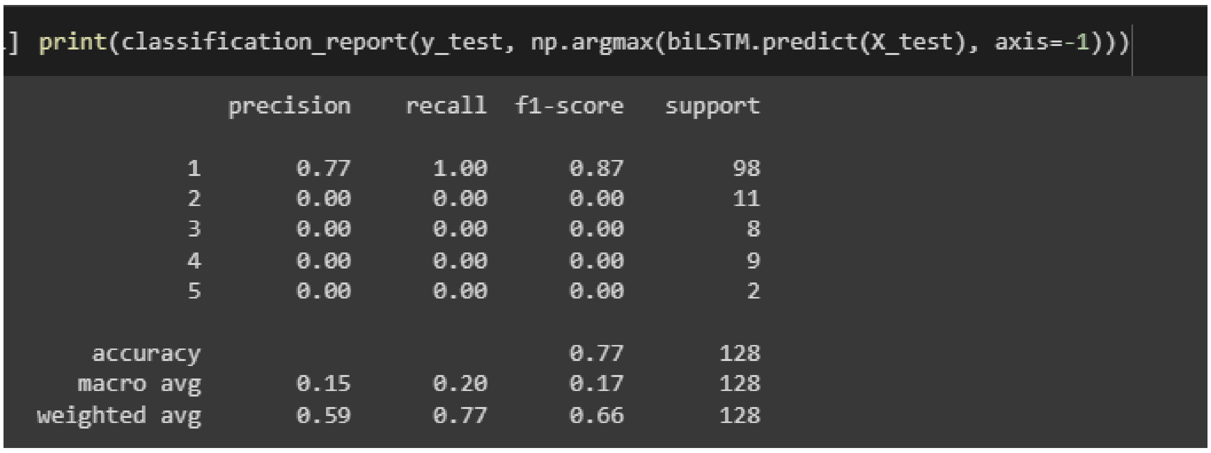
```
[100] biLSTM.summary()

      Model: "sequential_4"
      _________________________________________________________________
       Layer (type)                Output Shape              Param #
      =================================================================
       embedding_3 (Embedding)     (None, 32, 300)           4236000

       bidirectional_2 (Bidirectio  (None, 32, 128)          186880
       nal)

       bidirectional_3 (Bidirectio  (None, 64)               41216
       nal)

       dense_3 (Dense)             (None, 14120)             917800

      =================================================================
      Total params: 5,381,896
      Trainable params: 1,145,896
      Non-trainable params: 4,236,000
      _________________________________________________________________
```

Model Performance:



```
[90] history = biLSTM.fit(X_train, y_train, epochs=4, batch_size=1024,
                     validation_data=(X_test, y_test))

     Epoch 1/4
     1/1 [==============================] - 13s 13s/step - loss: 9.5521 - accuracy: 0.0000e+00 - val_loss: 9.1930 - val_accuracy: 0.7656
     Epoch 2/4
     1/1 [==============================] - 1s 839ms/step - loss: 9.2070 - accuracy: 0.7340 - val_loss: 8.5056 - val_accuracy: 0.7656
     Epoch 3/4
     1/1 [==============================] - 1s 874ms/step - loss: 8.5172 - accuracy: 0.7340 - val_loss: 7.6859 - val_accuracy: 0.7656
     Epoch 4/4
     1/1 [==============================] - 1s 928ms/step - loss: 7.7000 - accuracy: 0.7340 - val_loss: 6.6731 - val_accuracy: 0.7656
```

Classification Report:



```
] print(classification_report(y_test, np.argmax(biLSTM.predict(X_test), axis=-1)))

                  precision    recall  f1-score   support

               1       0.77      1.00      0.87        98
               2       0.00      0.00      0.00        11
               3       0.00      0.00      0.00         8
               4       0.00      0.00      0.00         9
               5       0.00      0.00      0.00         2

        accuracy                           0.77       128
       macro avg       0.15      0.20      0.17       128
    weighted avg       0.59      0.77      0.66       128
```

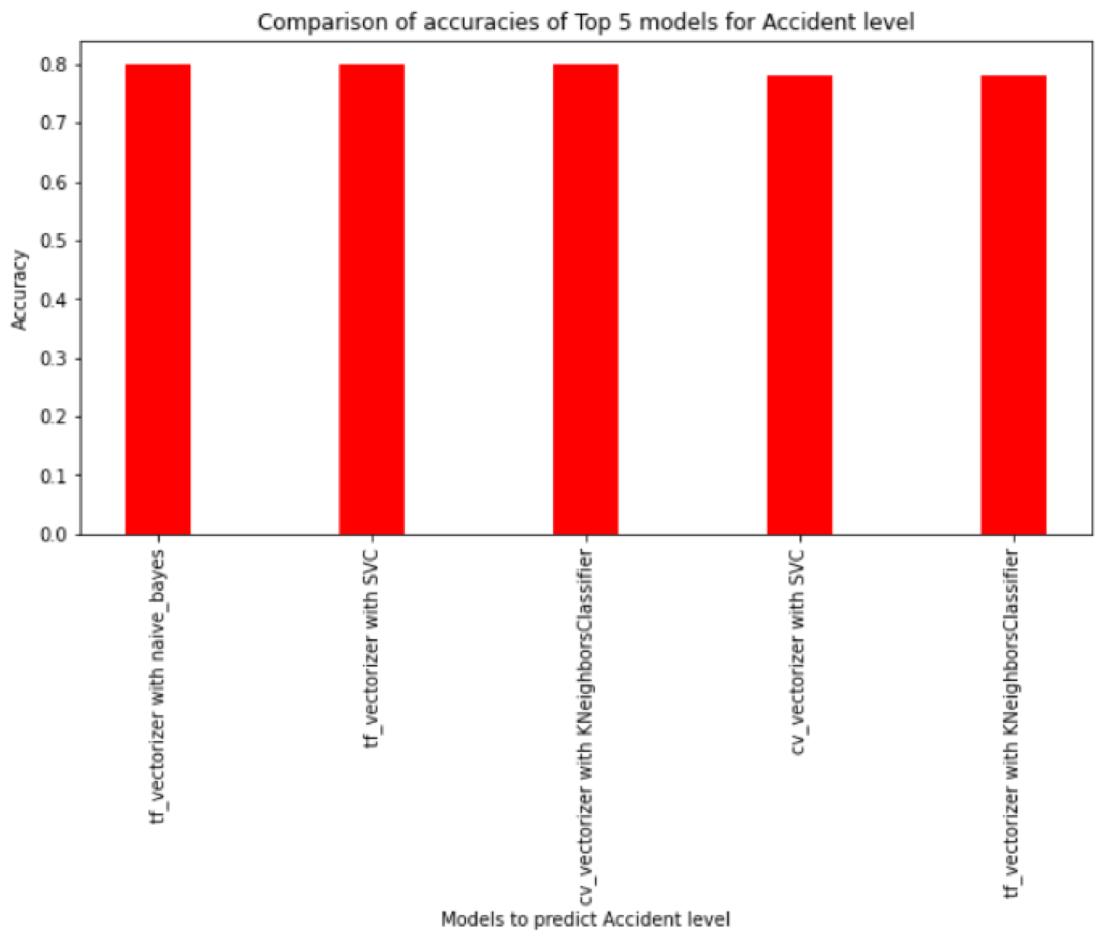
Other models used:

The following table displays all other models performance:

Model	Accuracy
Random Forest Classifier with Glove Accuracy	0.72
Bagging Classifier with Glove Accuracy	0.69
TF-IDF with Naïve bayes	0.80
TF-IDF with SVC	0.80
TF-IDF with KNeighborsClassifier	0.78
Countvectorizer with Naïve bayes	0.76
Countvectorizer with SVC	0.78
Countvectorizer with KNeighborsClassifier	0.80

Model Comparison:

Following are the top 5 best performing models on the given dataset:



In comparison of all the above models for target label Accident level, we can say that tf_vectorizer with naive_bayes , tf_vectorizer with SVC and cv_vectorizer with KNeighborsClassifier shows similar accuracy.

4) How to improve Model performance?

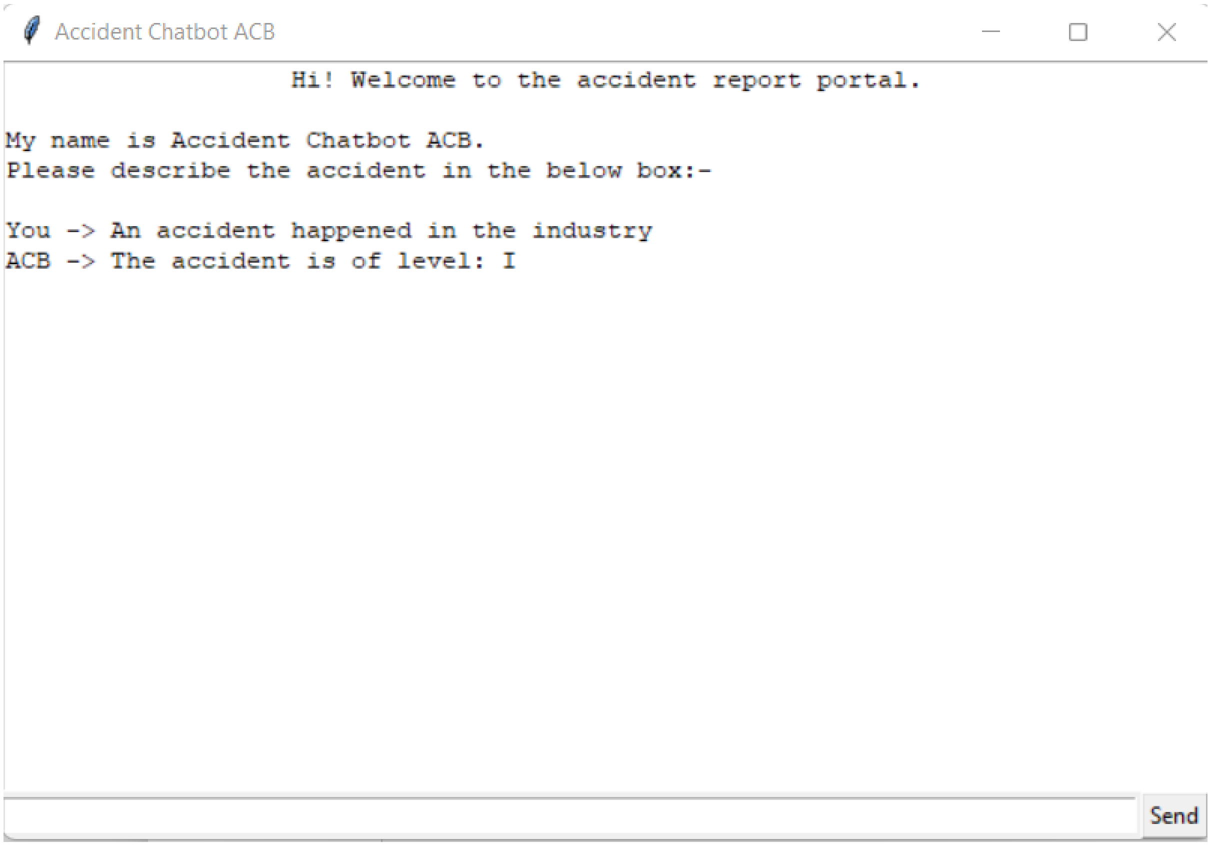
- Enhancements can be made to improve performance by Model tuning and using sampling techniques like SMOTE for imbalanced data.

- By changing parameters of the model using other vectorizer techniques like ELMO etc.

- Data Augmentation methods can be used to improve performance.

Appendix

Graphical User Interface



```
                   Hi! Welcome to the accident report portal.

My name is Accident Chatbot ACB.
Please describe the accident in the below box:-

You -> An accident happened in the industry
ACB -> The accident is of level: I
```