Great Learning

AIML

CAPSTONE PROJECT

NATURAL LANGUAGE PROCESSING

CHATBOT INTERFACE

Mentor: *Amit Ajit Lakhan*

Team:

Anu Priyam
Anusha K Holla
Alekhya
Shafna Mohamed Sharafuddin

Abstract

Industries around the world are in the dark when it comes to educating their stakeholders and taking precautionary measures in a time bound manner about industry safety. The costs incurred are often so high like injury and death of a person, causing pain and suffering to them and their families and resulting in hospitalization charges, litigation fees, loss of reputation and loss of employee morale. The purpose of the project is to make a chatbot that can be easily used by personnel in companies interested in improving safety and safety related regulations. Performance management related to safety is not new, but risk and safety management solutions which integrate key performance indicators, incident and near miss data, modeling results and subjective inputs from the workspace are propelling organization into the next phase. We are planning to design an open source RASA chatbot where we plug in our custom deep learning models which can be upgraded as technology and algorithms improve over time. The end result will be an end-to-end chatbot implementation for companies interested in increasing timely safety related information access that reduce injuries, deaths and related litigation expenses, emotional trauma and lost employee morale most of which once lost is impossible to retrieve thus making the offering a must have.

Content

1. Problem Statement

The database comes from one of the biggest industries in Brazil and in the world. It is an urgent need for the industries /companies around the globe to understand why employees still suffer some injuries/accidents in plants. Sometimes they also die in such an environment.

Data Description: This database is basically records of accidents from 12 different plants in 3 different countries where every line in the data is an occurrence of an accident.

Attributes Description:

Data: Timestamp or time/date information
Countries: Which country the accident occurred (anonymised)
Local: The city where the manufacturing plant is located (anonymised)
Industry sector: Which sector the plant belongs to
Accident level: From I to VI, it registers how severe was the accident (I means not severe but VI means very severe)
Potential Accident Level: Depending on the Accident Level, the database also registers how severe the accident could have been (due to other factors involved in the accident)
Genre: If the person is male or female
Employee or Third Party: If the injured person is an employee or a third party
Critical Risk: Some description of the risk involved in the accident
Description: Detailed description of how the accident happened.

Dataset:
https://www.kaggle.com/ihmstefanini/industrial-safety-and-health-analytics-database

Project Objective: Design a ML/DL based chatbot utility which can help the professionals to highlight the safety risk as per the incident description.

2. EDA

Data Insights:

On the inspection of the dataset , it appears that:

- The dataset consists of 425 incidents in raw data, we can say that dataset is limited. So building the model with high accuracy is difficult.
- There are five accident level and six potential accident level
- The incident is from 3 countries and 12 cities
- There are thirty-three different types of critical risks associated with accident
- The incident collected in data are from January 2016 to July 2017
- Males are more involved in accidents than female as number of females work in industrial environment is lesser than male
- Countries and Local columns tell about the location of the accident as in which country and which city the accident occurred. However the actual country or city names are omitted.
- Genre column contains gender details with two values Male or Female.
- Accident level and Potential accident level are metrics of severity of accident which are our target variable.
- Employee or Third Party indicates employee type which can be renamed to Employee type for user readability
- Description column explains how an accident happened which is the major independent variable that needs most of the data preprocessing.
- There are no null values found in the dataset. But some of the records were marked as others in industry and critical risks
- Index column unnamed:0 can be dropped
- All the columns except Date are categorical.
- There are 7 duplicate records in the data

Data Cleansing:

1. Removed Unnamed: 0' column as its duplicate a index column

```
#Drop unwanted columns in dataset
def dropunwantedcolumns(column):
    if (column != ''):
        df.drop(column, axis=1,inplace=True)
        print("Dropped unwanted column -",column)
```

```
Drop unwanted columns
---------------------
Dropped unwanted column - Unnamed: 0
```

2. There are 7 duplicates record in the data that has been dropped

```
#remove duplicates in dataset
def dropduplicates():
    if(df.duplicated().sum()==0):
        print("No Duplicates")
    else:
        print("Dropping",df.duplicated().sum()," duplicates ")
        df.drop_duplicates(keep='first', inplace=True)
```

```
 Duplicates check
-------------------
Dropping 7  duplicates
```

3. We checked for missing values and unexpected values. We could not find any missing values.

```
#Checks for missing value and unexpected values and impute with relevant data
def check_missingvalues(dataframe):
    missingvalue = False
    unexpected_values = ["n/a", "na", "--", " ","?","-","@","#","n-a"]

    for feature in dataframe.columns:
        #check for NaN values
        if(dataframe[feature].isnull().sum()!=0 or dataframe[feature].isna().sum()!=0 ):
            print("Data missing in column", feature)
            missingvalue= True
        #check for unexpected values like special characters.
        if(not dataframe[dataframe[feature].isin(unexpected_values)].empty):
            print("Unexpected value found in",feature,"column in the indices ", dataframe[dataframe[feature].isin(unexpected_va
            #changing unexpected values to NaN
            df.replace(unexpected_values,np.nan, inplace=True)
            missingvalue= True
    if(not missingvalue):
        print("No missing value in dataframe")
    else:
        print("Missing Values Found")
```

4. We have renamed column names for better readability

```
print("Renaming column name for better readability")
print("--------"*6)
df.rename(columns={'Data':'Date', 'Countries':'Country', 'Genre':'Gender', 'Employee or Third Party':'Employee type'}, inplace=T
print(df.columns)
print("\n\n")
```

```
Renaming column name for better readability
------------------------------------------------
Index(['Date', 'Country', 'Local', 'Industry Sector', 'Accident Level',
       'Potential Accident Level', 'Gender', 'Employee type', 'Critical Risk',
       'Description'],
      dtype='object')
```

5. Finding unique values for categorical

```
Getting Unique values in the dataset
------------------------------------
Country ['Country_01' 'Country_02' 'Country_03']

Local ['Local_01' 'Local_02' 'Local_03' 'Local_04' 'Local_05' 'Local_06'
 'Local_07' 'Local_08' 'Local_10' 'Local_09' 'Local_11' 'Local_12']

Industry Sector ['Mining' 'Metals' 'Others']

Accident Level ['I' 'IV' 'III' 'II' 'V']

Potential Accident Level ['IV' 'III' 'I' 'II' 'V' 'VI']

Gender ['Male' 'Female']

Employee type ['Third Party' 'Employee' 'Third Party (Remote)']

Critical Risk ['Pressed' 'Pressurized Systems' 'Manual Tools' 'Others'
 'Fall prevention (same level)' 'Chemical substances' 'Liquid Metal'
 'Electrical installation' 'Confined space'
 'Pressurized Systems / Chemical Substances'
 'Blocking and isolation of energies' 'Suspended Loads' 'Poll' 'Cut'
 'Fall' 'Bees' 'Fall prevention' '\nNot applicable' 'Traffic' 'Projection'
 'Venomous Animals' 'Plates' 'Projection/Burning' 'remains of choco'
 'Vehicles and Mobile Equipment' 'Projection/Choco' 'Machine Protection'
 'Power lock' 'Burn' 'Projection/Manual Tools'
 'Individual protection equipment' 'Electrical Shock'
 'Projection of fragments']
```

2.1 Univariate Analysis

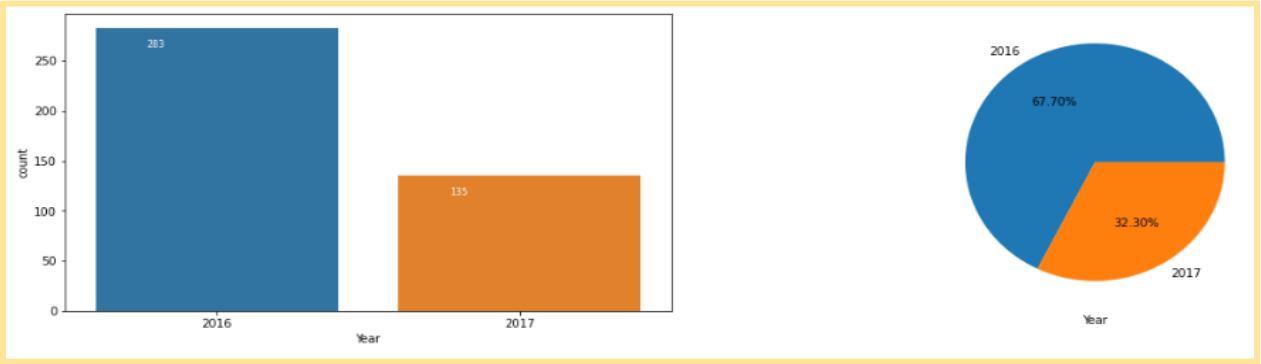
2.1.1 Date Feature:

The accident's occurrence has been analyzed year-wise, month-wise and day-wise to know if there is a trend.
Temporary dataframe has been created to analyze year and month of accident occurrence.

	Date	Year	Month	Weekday
0	2016-01-01	2016	1	Friday
1	2016-01-02	2016	1	Saturday
2	2016-01-06	2016	1	Wednesday
3	2016-01-08	2016	1	Friday
4	2016-01-10	2016	1	Sunday

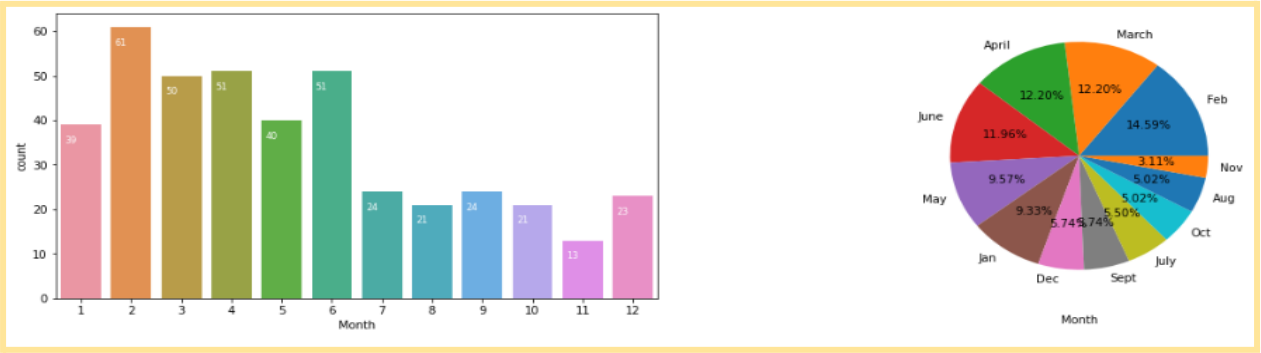
Year Analysis:

Most accidents happened in 2016. Count is 283 ,which is equivalent to 67.70%.



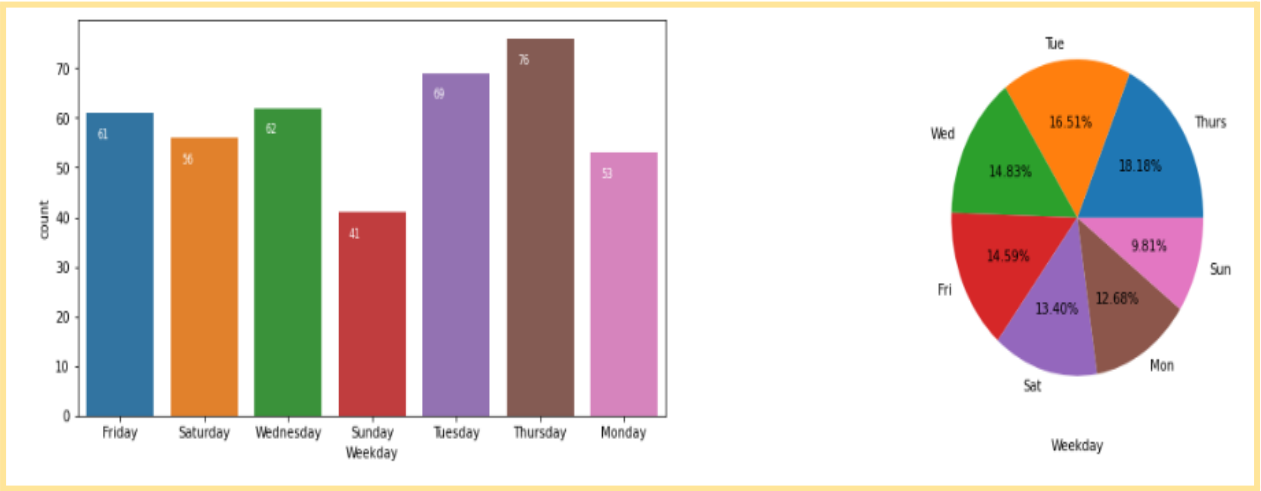
Month Analysis

No trend found in the months but most accidents happened in the month of Feb. Count is 61, which is equivalent to 14.59%.



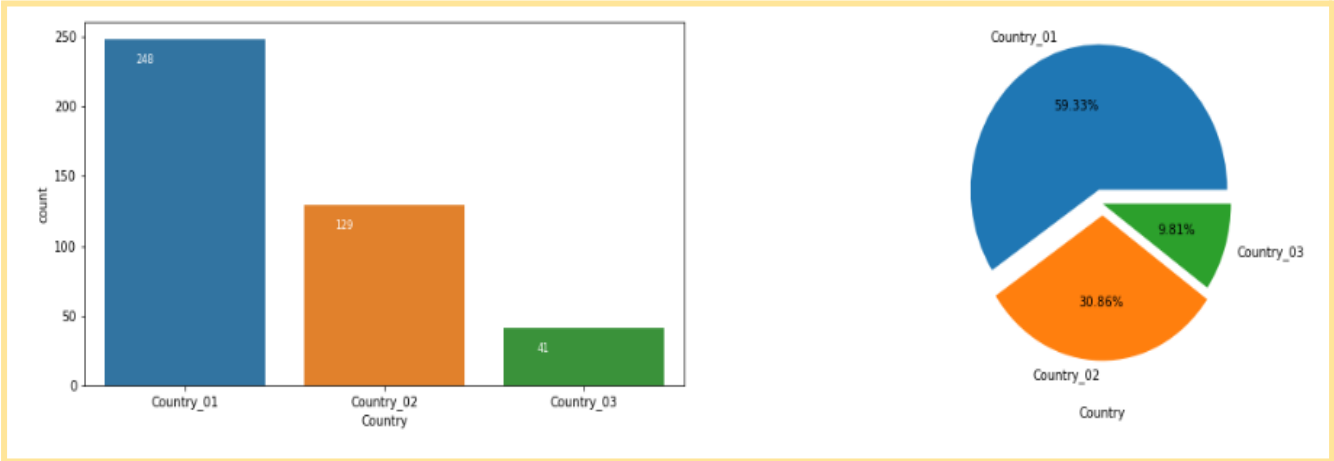
Day Analysis

No trend was found in the days but most accidents happened on Thursday. Count is 76, which is equivalent to 18.18%.



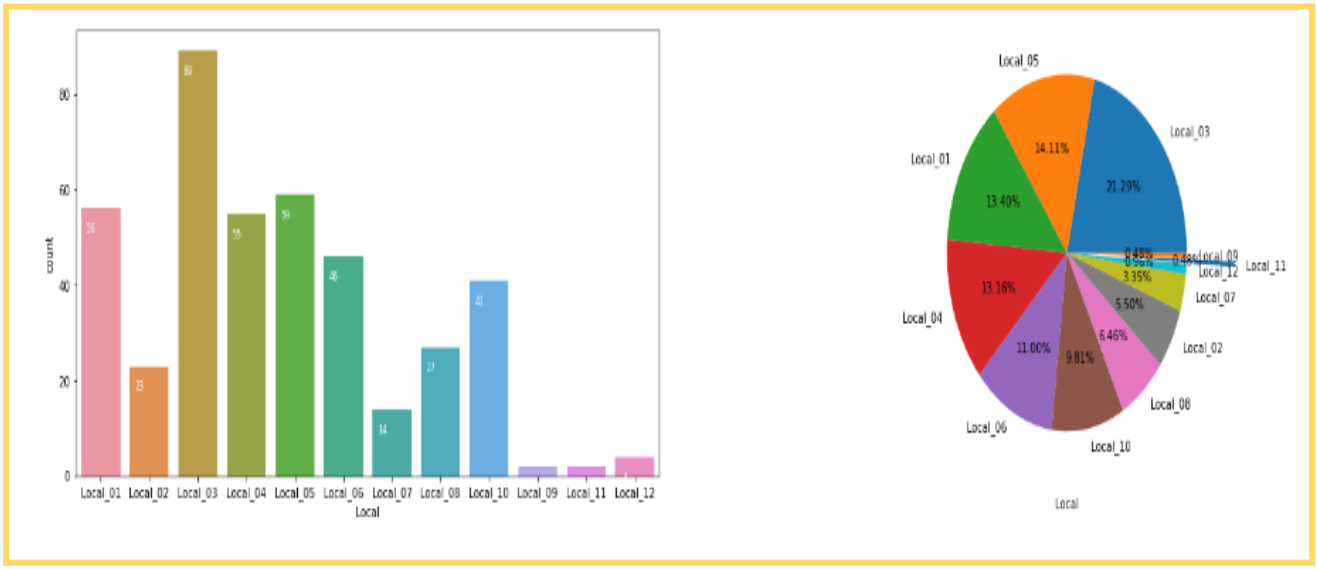
2.1.2. Country Feature Analysis:

Most accidents happened in Country_01. Its count is 248, which is equivalent to 59.33%.



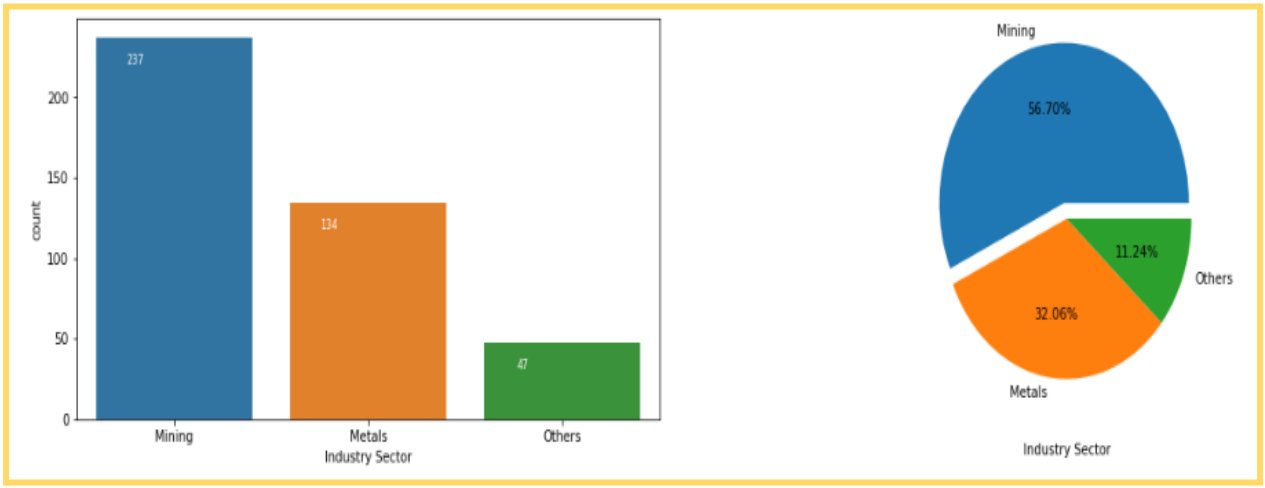
2.1.3. Local Feature Analysis:

Most accidents happened in Local_03. The count is 89 which is equivalent to 21.29%.



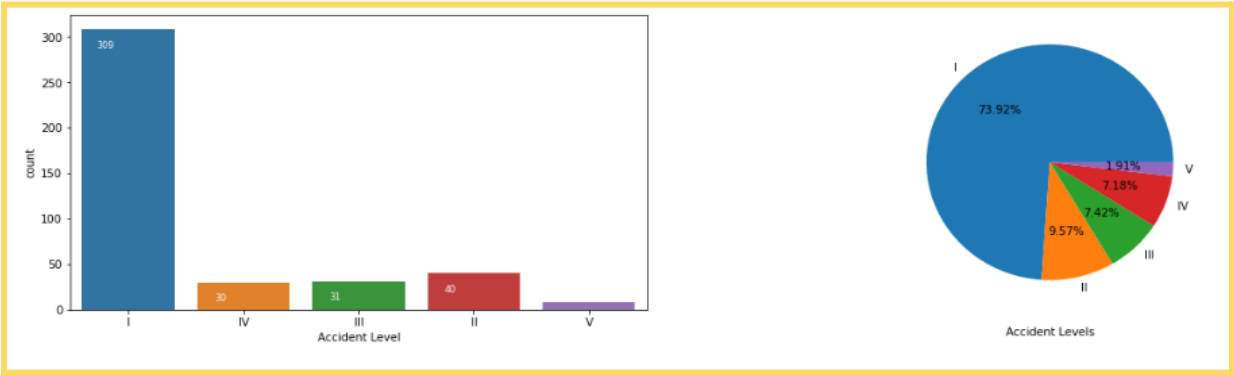
2.1.4. Industry Sector Analysis:

Most accidents happened in the mining industry sector. Count is 237 which is equivalent to 56.70%.



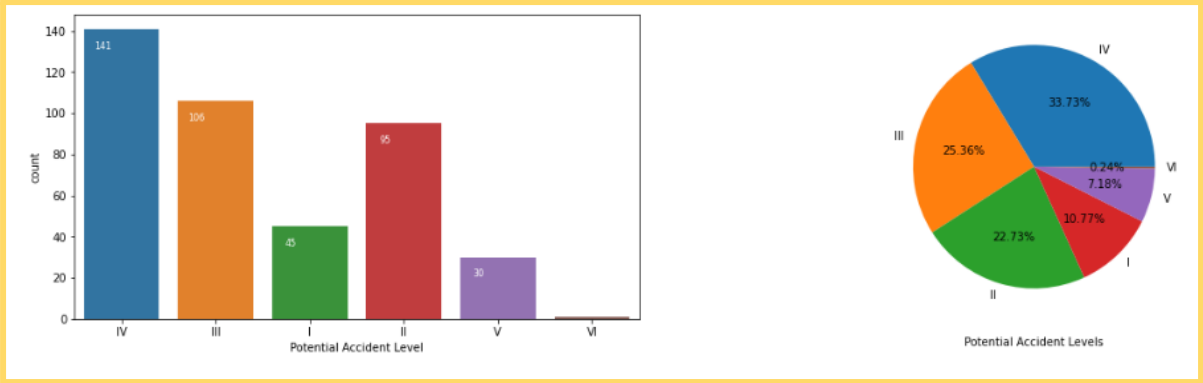
2.1.5. Accident Level Analysis:

Most of the accidents were from “Accident Level I”. Count is 309 which is equivalent to 73.92% of total accidents.



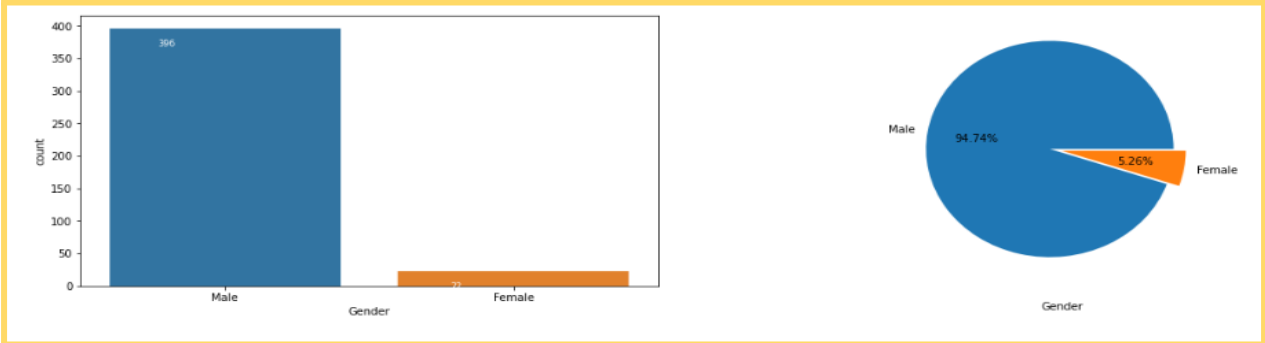
2.1.6. Potential Accident Level Analysis:

Most “Potential Accident Level” belongs to Level IV. Its count is 141 which is equivalent to 33.73% of total potential accidents.



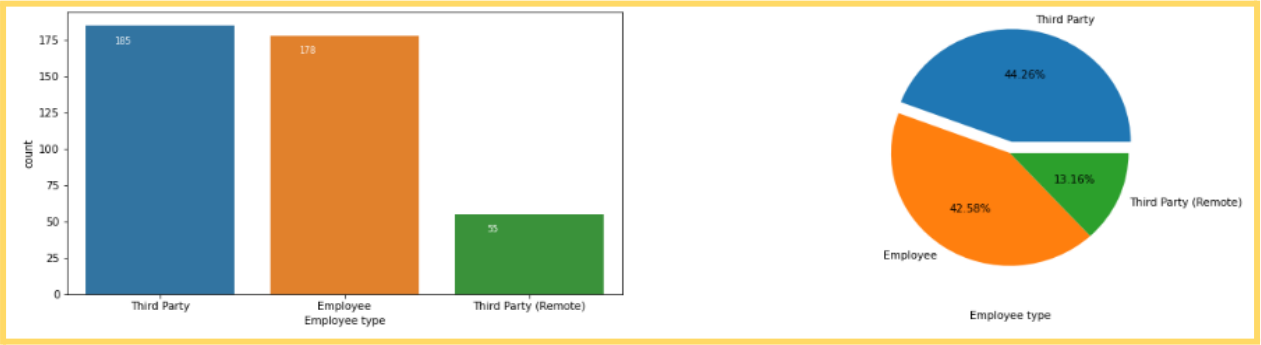
2.1.7. Gender Feature Analysis:

Most affected workers in accidents are Male. Their count is 396 which accounts to 94.74%.



2.1.8. Employee Type Analysis:

Most affected Employee types are Third Party Workers. Their count is 185, which is equivalent to 44.26%.

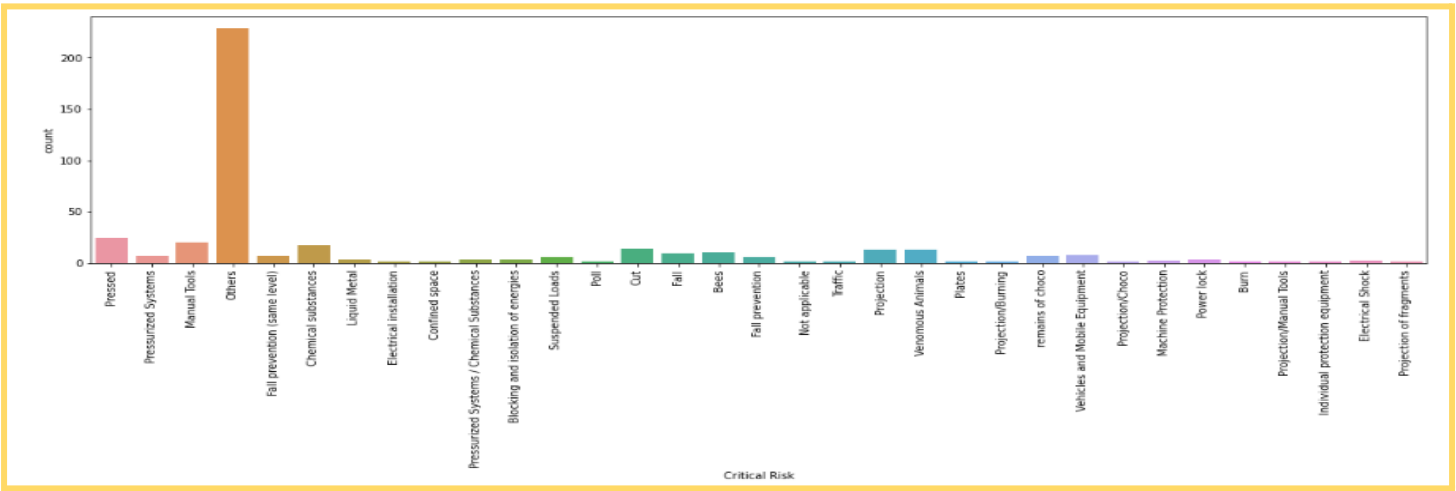


2.1.9. Critical Risk Feature Analysis:

Majority of critical risk cases were from the ‘Others’ and ‘Not applicable’ category. And these columns have 50% of missing categorization. Adding them in X as separate dependent variables may cause bias in the model.

Hence, we plan to append all critical risk columns excluding ‘Others’ to description inorder to reduce dimensionality and preserve the information in critical risk.

All features except Description are analysed and findings were noted. Will be doing Textual pre-processing to Description and analyse that feature later.



2.2 Bivariate and Multi-Variate Analysis:

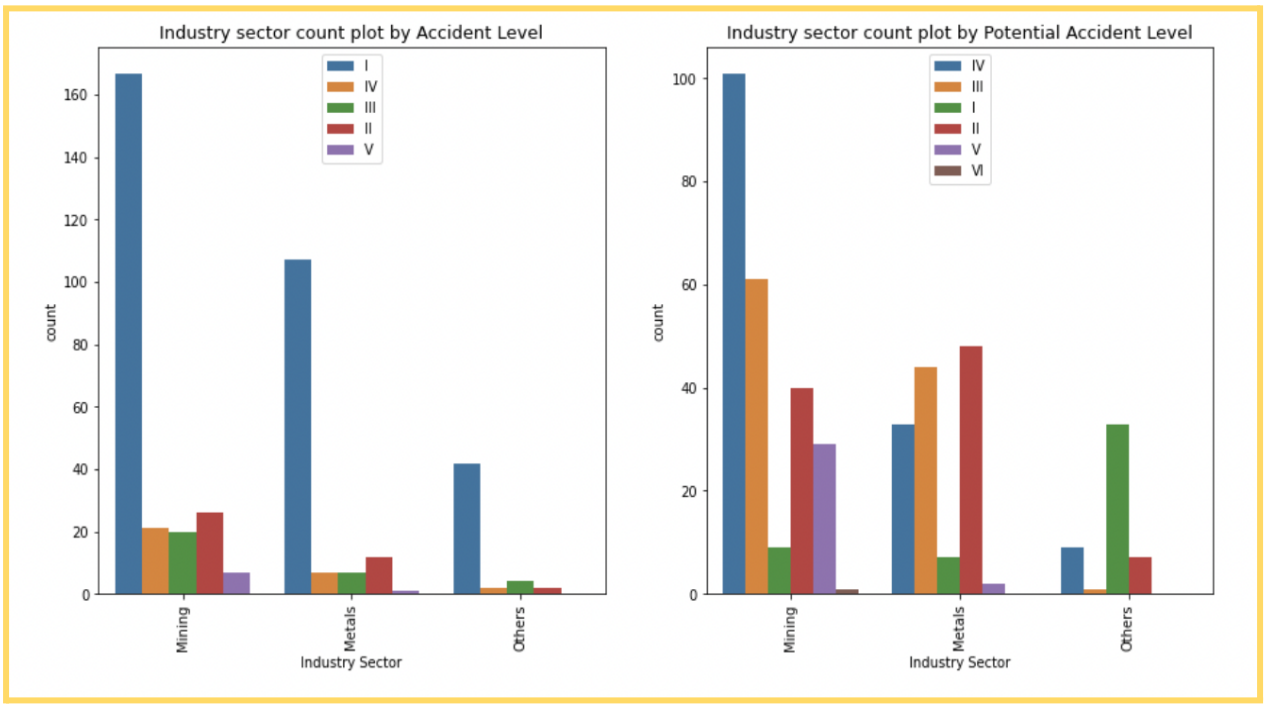
2.2.1. Industry Sector Vs Accident/ Potential Accident Level

a) Industry Sector VS Accident Level

- 'Accident Level I' is highest in all industry sectors namely Mining, Metals and Other).
- Most accidents happened in the Mining industry sector.
- 'Others' sector has fewer accidents in comparison to the rest of the industry sectors.
- There are very few cases for 'Accident Level V'. 'Others' sector did not report any case for Level V.

b) Industry Sector VS Potential Accident Level

- 'Potential Accident Level IV' is highest in all Industry Sectors(Mining, Metals and Others)
- Most accidents happened in the Mining Industry Sector.
- 'Others' have fewer accidents when compared to the rest of the sectors.
- There are only a few cases being reported for 'Accident Level VI'. The Metals and Others sector did not report any case for this accident level.



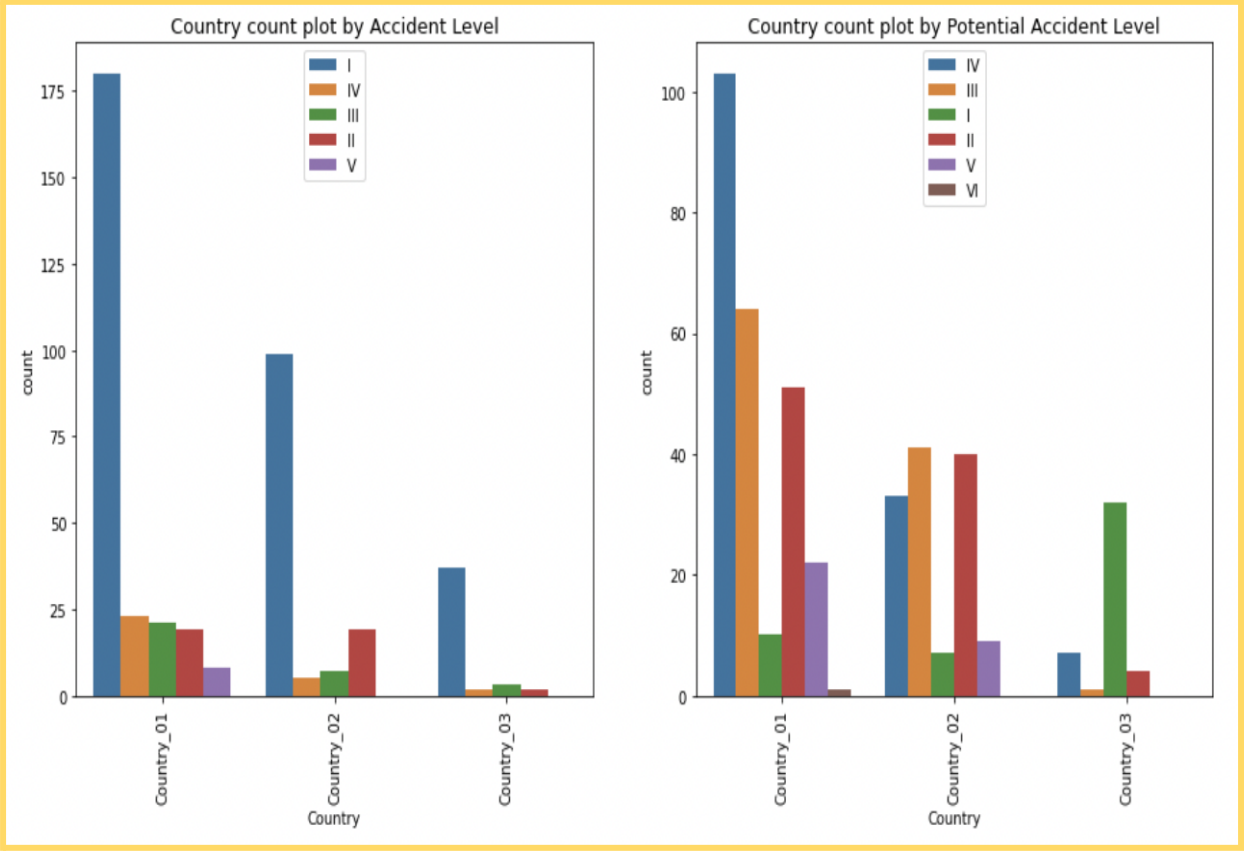
2.2.2 Country VS Accident/Potential Accident Level

a) Country VS Accident Level

- 'Accident level I' is highest in all countries.
- Most accidents happened in Country_01.
- Accident level in Country03 is lesser than other countries.

b) 2.2.2 Country VS Potential Accident Level

- Potential Accident level IV is highest in Country_01.
- Most accidents happened in Country_01.
- Accident level in Country_03 is lesser than other countries.



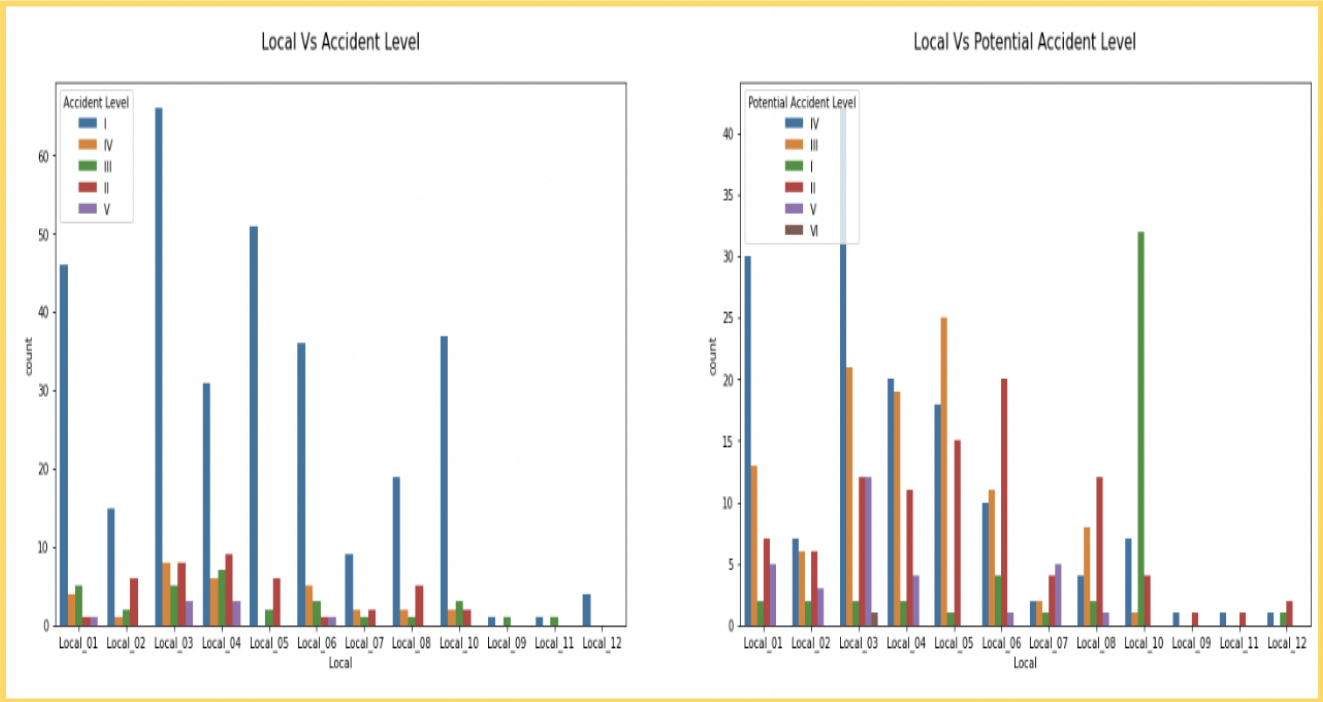
2.2.3. Local VS Accident Level/Potential Accident Level

a) Local VS Accident Level

- ‘Accident level I’ is highest in almost all localities.
- ‘Accident level I’ is highest in Local_03.
- Local_09,11 and 12 have the least number of accident levels.

b) Local VS Potential Accident Level

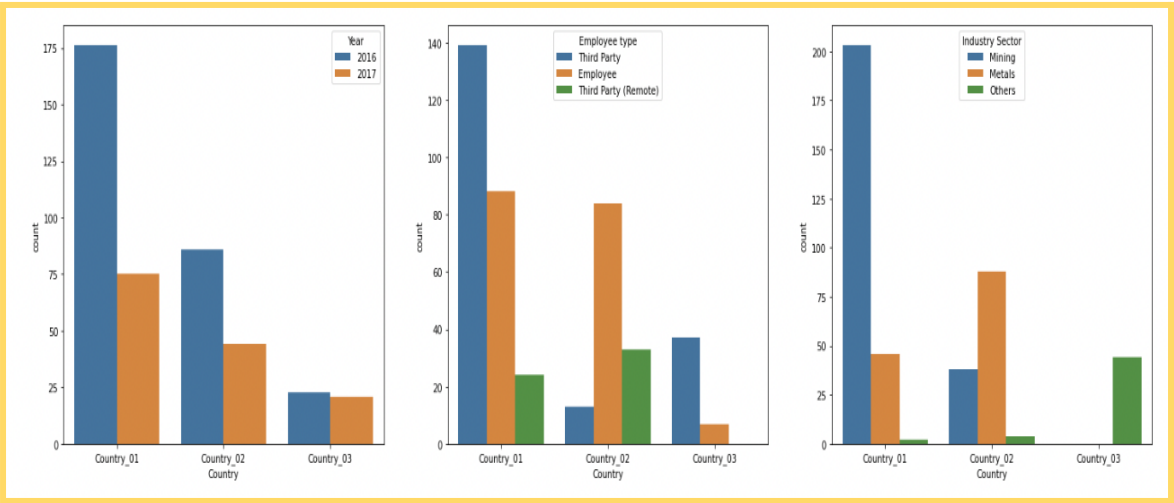
- ‘Potential Accident level IV’ is highest in Local_03.
- ‘Potential Accident level IV’ is higher in most localities.
- Local_09,11 and 12 have less potential accidents levels.
- ‘Potential Accident level I’ is highest in Local_10.
- ‘Potential Accident level III’ is highest in Local_05.



2.2.4. Country VS Year/Employee Type and Industry Sector

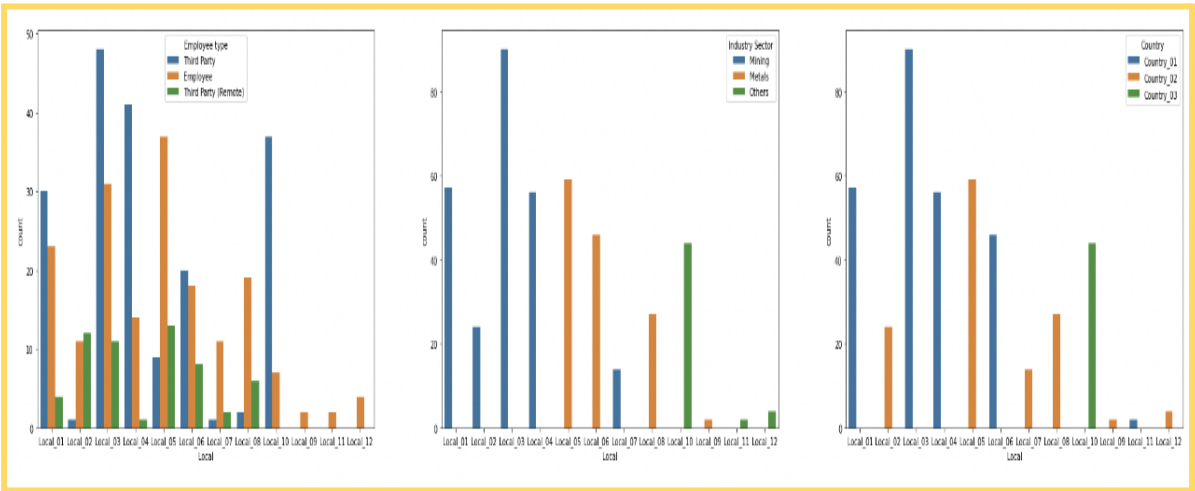
- In 2016 and 2017 Country_01 had more accidents than other countries.
- In 2016 and 2017 Country_03 had fewer accidents than other countries.
- Incidents in Country_01 have an Employee type of Third Party high in number.
- Incidents in Country_02 have an Employee type of Employee high in number.
- Incidents in Country_03 have an Employee type of Third Party high in number.

- In Country_01, incidents in the Mining sector are very high.
- In Country_02, incidents in the Metals sector are very high.
- In Country_03, incidents in Other sectors are very high.



2.2.5. Local VS Employee Type

- Local_03 has highest no of Third Party employees
- Local_03 has more employees than other localities.
- Local_10 doesn't have Third Party (Remote) employees.
- Local_09,11,12 don't have Employee and Third Party employees.
- Local_03 has the highest number of Mining industry sectors.
- Local_05 has the highest number of Metals industry sectors.

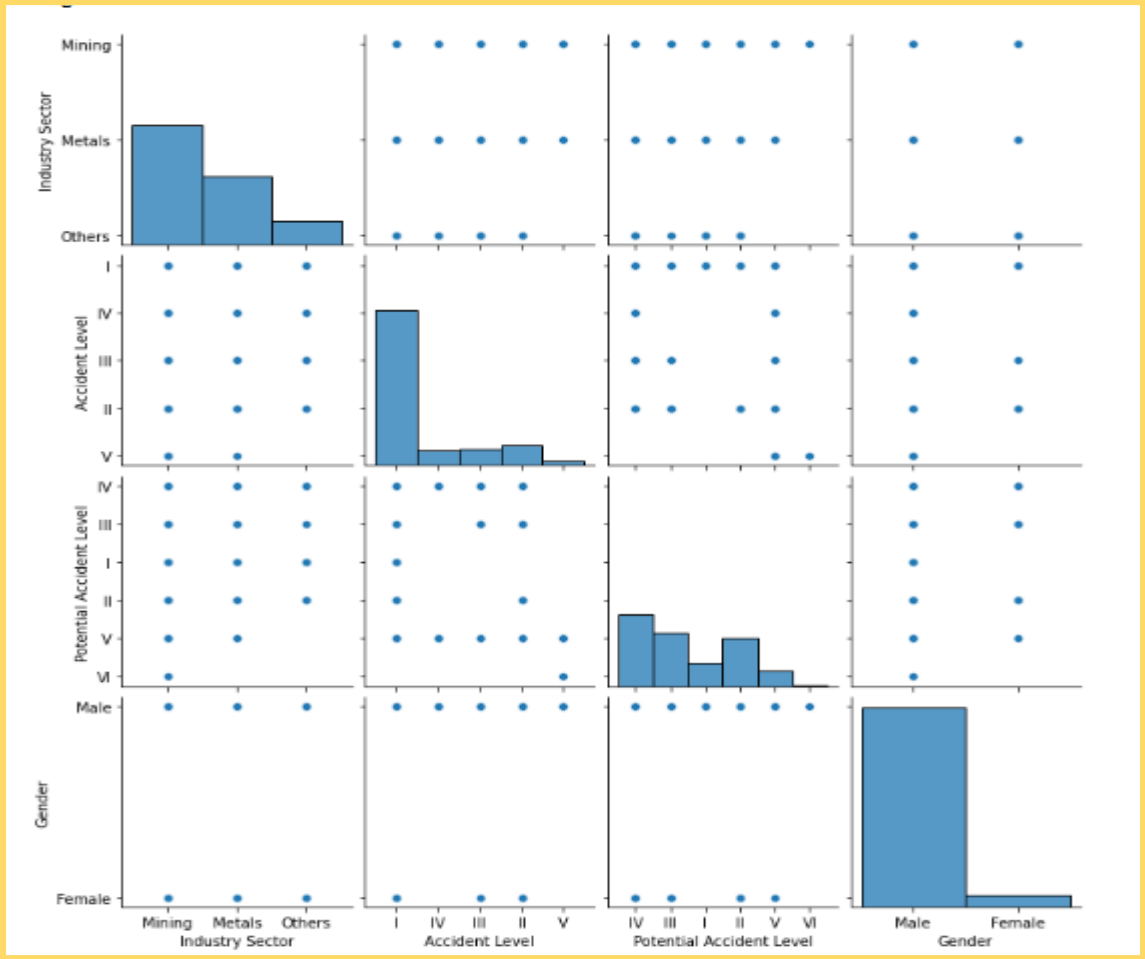


- Local_01,02,03,04 & 07 has Mining industry sectors.
- Local_05,06,08 & 09 has Metals industry sectors.
- Local_10,11,12 has Others industry sectors.
- Local_03 has a majority in Country_01.
- Local_05 has a majority in Country_02.
- Local_01,03,04,06 and 11 belong to Country_01.
- Local_02,05,07,08,09 and 12 belong to Country_02.
- Local_10 belongs to Country_03.

2.3 Multi-variate analysis using pairplot

A pair plot has been plotted against all variables to analyze if there is any relationship.

For the Metals industry sector, there is no potential Accident Level V1. In the Other industry sector, Potential Accident Level IV and VI doesn't happen and Accident Level V didn't happen.



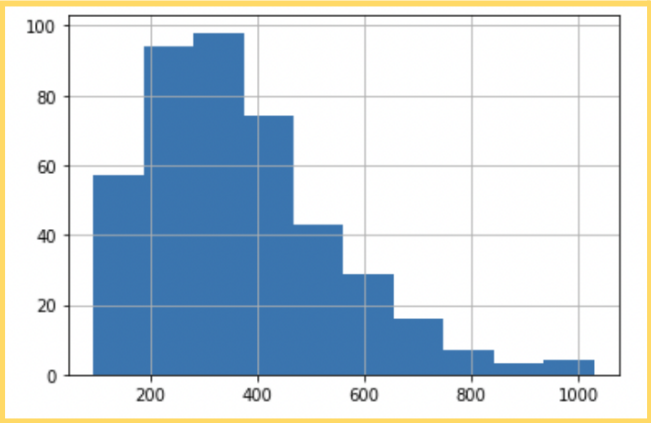
3. Data Preprocessing (NLP Preprocessing Techniques)

Text processing is the first step in the process of building a model.Few of the NLP pre-processing steps taken before applying model on the data are:

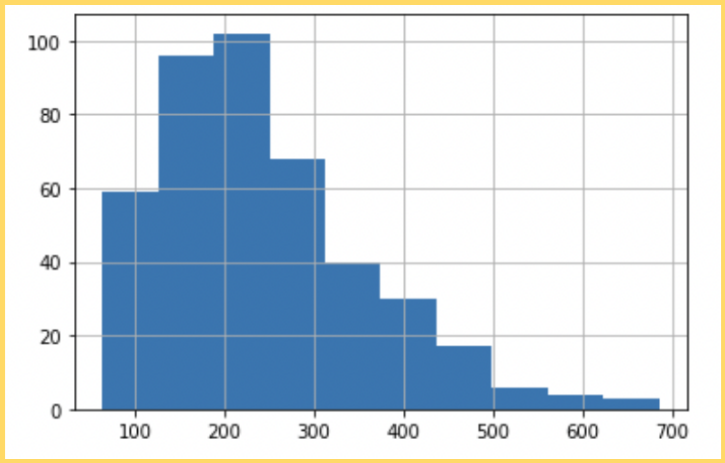
- Removing stopwords.
- Lemmatization.
- Converting to lowercase, avoid any capital case.
- Remove special characters and numbers.
- Remove extra white space.

```
def preprocess_text(text):

      #stopwords removal
      df['Cleaned Desc'] = text.apply(lambda t: ' '.join([words for words in t.split() if words not in stop]) )

      #lemmetization
      lemmatizer = WordNetLemmatizer()
      df['Cleaned Desc']=df['Cleaned Desc'].apply(lambda t: ' '.join([lemmatizer.lemmatize(word) for word in t.split(" ")]))

      #Convert to lowercase
      df['Cleaned Desc'] = df['Cleaned Desc'].apply(lambda s: s.lower())

      #Remove Special characters and Numbers
      pattern = r'([^a-zA-Z\s]+?)'
      df['Cleaned Desc']  = df['Cleaned Desc'].apply(lambda s : re.sub(pattern,"",s))

      # remove extra white spaces
      df['Cleaned Desc']  = df['Cleaned Desc'].apply(lambda s: s.strip())

      return df['Cleaned Desc']

stop=set(stopwords.words('english'))
df['Cleaned Desc']= preprocess_text(df['Description'])
```

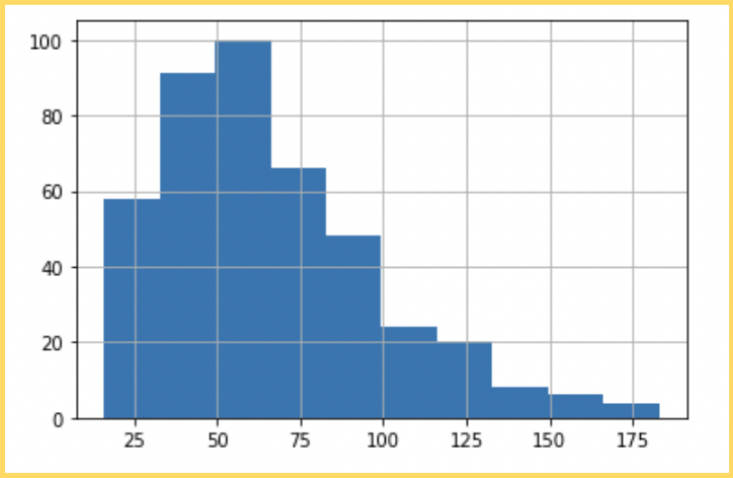
Analysing description and cleaned description



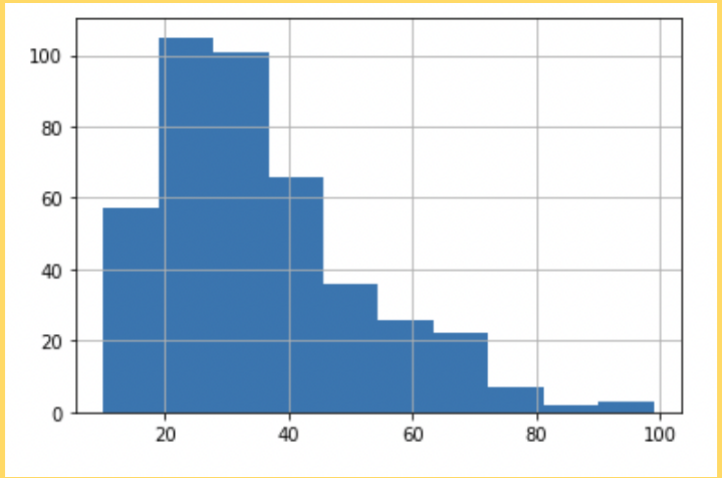
The max length of description is in the range of 200-400. The distribution is left skewed.



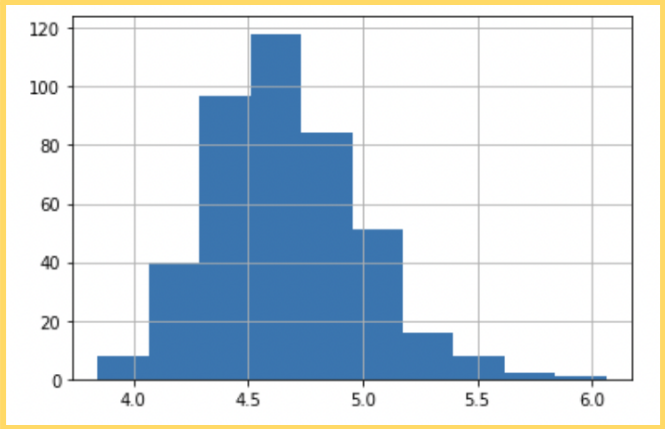
The max length of description after cleaning falls in the range of 120-250. The distribution is left skewed.



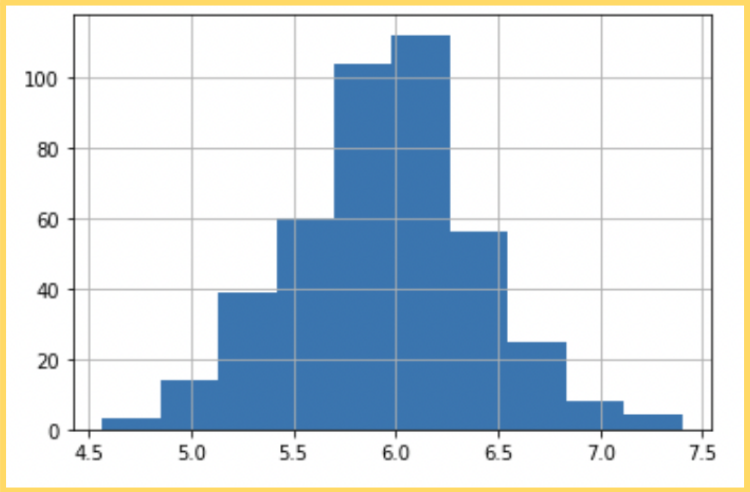
The words in description are distributed along the range of 12-175 where the highest frequency is in between 50-75 with the most number of words being 175. The distribution is left skewed.



The words in the cleaned description are distributed along the range of 5-100 where the maximum length is in the range of 80-100. The distribution is left skewed.

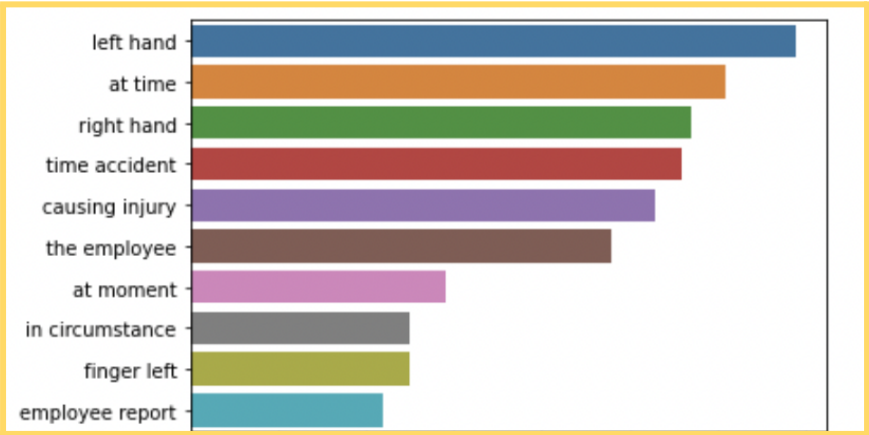


The below histogram depicts the average distribution of words. It is normally distributed.

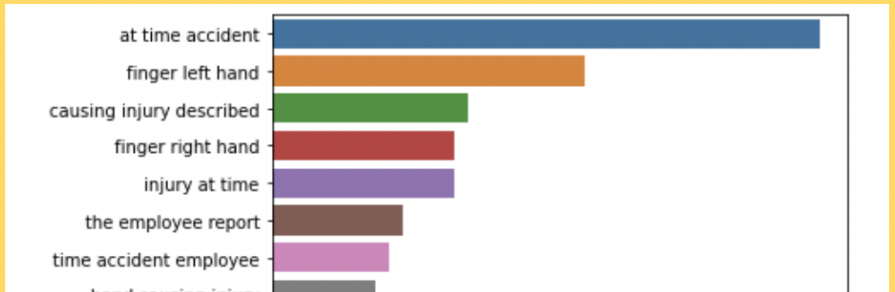


The below histogram depicts the average distribution of words in cleaned description which is also normally distributed.

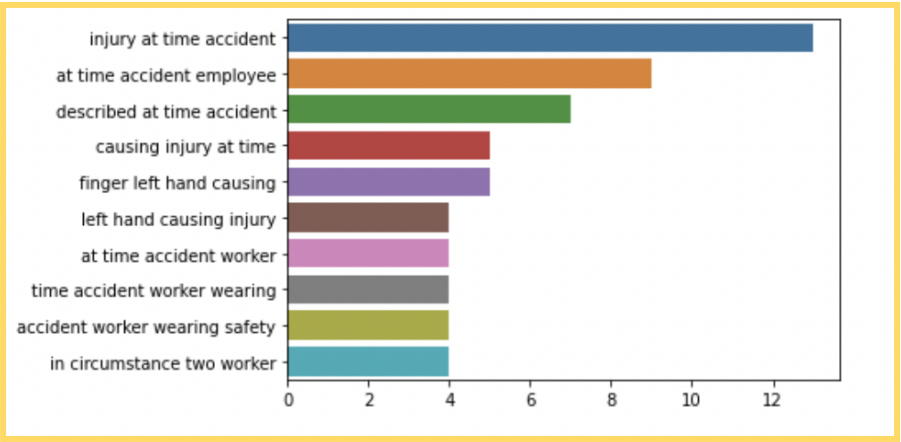
N-gram is a sequence of N words.



Maximum occurring 2 words in the cleaned description is ‘left hand’.



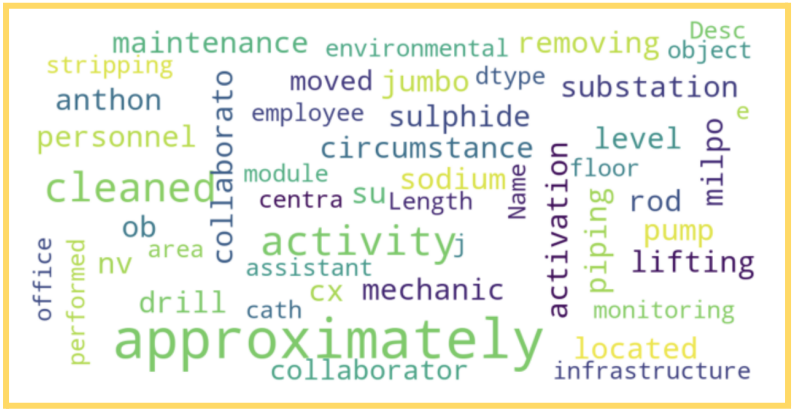
Maximum occurring 3 words in the cleaned description is ‘at time accident’.



Maximum occurring 4 words in the cleaned description is ‘injury at time accident’.

Word Cloud

Word cloud has been generated for representing textual data in which the size of each word indicates its frequency or importance. Significant textual data points are highlighted using a word cloud. ‘Approximately’ is used the most out of all words.



4. Data Modelling

Data modeling is the process of creating a visual representation of either a whole information system or parts of it to communicate connections between data points and structures.

While building a predictive model we follow several different steps. We first do exploratory
data analysis to understand the data well and do the required preprocessing.
Data Modelling also requires additional processing of the data based on the independent variables and target variables. Since our target variable is categorical data with multiple labels it requires encoding of the data.

4.1 Encoding categorical data:

To train any given model , we are often required to convert all the categorical features i.e text features to its numerical representation. Here, our target variable - accident level which has 5 levels represented in romanic numeric format is label encoded to numeric format based on the priority provided in the data description.

Label encoding is the technique of converting the labels to numeric format so as to convert them to machine readable form.

Accident Levels	Label Encoded
I	1
II	2
III	3
IV	4
V	5

Label Encoding

```
[ ] replace_struc = {'I': 1, 'II': 2, 'III': 3, 'IV': 4, 'V': 5}
    df['Accident Level'] = df['Accident Level'].map(replace_struc)
    replace_struc = {'I': 1, 'II': 2, 'III': 3, 'IV': 4, 'V': 5, 'VI': 6}
    df['Potential Accident Level'] = df['Potential Accident Level'].map(replace_struc)

[ ] X = df['Cleaned Desc']
    Y = df['Accident Level']
```

The independent variable ‘Cleaned Desc’ is text which is a user input that describes the accident, feature extraction is very important before training any model. Below are the few techniques that can be used for feature extraction in NLP.

4.2 Natural Language Processing techniques:

1) Bag of words:

A bag of words is a representation of text that describes the occurrence of words within a document. We just keep track of word counts and disregard the grammatical details and the word order. It is called a “bag” of words because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document.

We have used word Analyzer and ngram_range = (1,2)

Bag Of Words

```
In [43]: BagofWords = CountVectorizer(analyzer='word', ngram_range=(1, 2))
         BagofWords.fit(X_train)
         X_train_BW = BagofWords.transform(X_train).toarray()
         X_test_BW = BagofWords.transform(X_test).toarray()


         #Check the vocabulory size
         print("Len of Vocab",len(BagofWords.vocabulary_))
         BagofWords.get_feature_names_out()[:15]

         Len of Vocab 11602

Out[43]: array(['abb', 'abb furnace', 'abdomen', 'abdomen left', 'able',
                'able position', 'able remove', 'abratech', 'abratech company',
                'abrupt', 'abrupt contact', 'abrupt movement', 'abruptly',
                'abruptly dropped', 'abruptly imprisoning'], dtype=object)

In [44]: X_train_BW[310]

Out[44]: array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

Above screenshot represents the Bag of words algorithm applied on the independent variable.

2) TFIDF:

TF-IDF stands for Term Frequency Inverse Document Frequency of records. It can be defined as the calculation of how relevant a word in a series or corpus is to a

text. The meaning increases proportionally to the number of times in the text a word appears but is compensated by the word frequency in the corpus (data-set). TF-IDf for each word is calculated using the below formula.

tf-idf(t, d) = tf(t, d) * idf(t)

Where tf(t,d) - term frequency for a document d, idf(t) -is the number of documents in the corpus separated by the frequency of the text, idf(t) = log(N/ df(t))

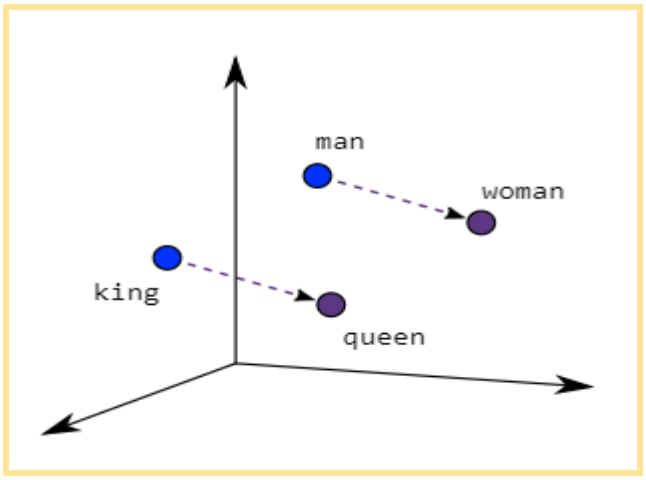
We chose ngram_range = (1,2). Below screenshot represents the TF-IDF applied on the independent variable.

TFIDF

```
In [47]: TFIDF = TfidfVectorizer(ngram_range=(1,2))
         TFIDF.fit(X_train)
         X_train_TFIDF = TFIDF.transform(X_train).toarray()
         X_test_TFIDF = TFIDF.transform(X_test).toarray()

In [48]: X_train_TFIDF[9]

Out[48]: array([0., 0., 0., ..., 0., 0., 0.])

In [49]: X_BW = BagofWords.transform(X).toarray()
         X_TFIDF = TFIDF.transform(X).toarray()
```

3) Word2Vec:

The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. This model is used to extract the notion of relatedness across words or products such as semantic relatedness, synonym detection, concept categorization, selectional preferences, and analogy. A Word2Vec model learns meaningful relations and encodes the relatedness into vector similarity.



Below screenshot represents the Word2Vec algorithm applied on the independent variable.

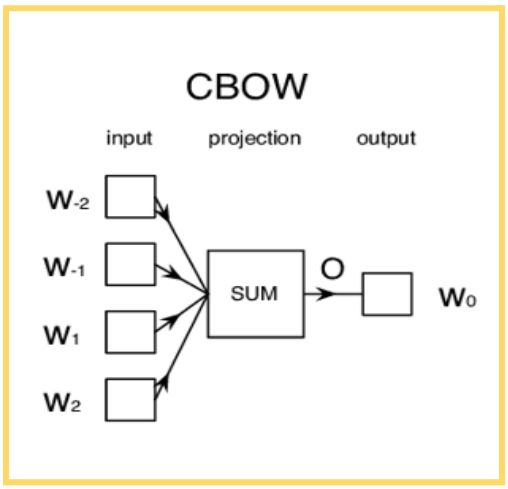
Word2Vec ¶

In [50]:
```
# Converting the words back to the sentence form for modelling
def word_vector(model,tokens, size):
    vec = np.zeros(size).reshape((1, size))
    count = 0.
    for word in tokens:
        try:
            vec += model.wv[word].reshape((1, size))
            count += 1.
        except KeyError: # handling the case where the token is not in vocabulary

            continue
    if count != 0:
        vec /= count
    return vec

training = df['Cleaned Desc'].tolist() # Covnerting the text to list
training = [sentence.split(' ')  for sentence in training] # Splitting on each sentence which gives the multi dimensional list
print("first sentence",np.array(training[0])) # Priting the first sentence
```

```
first sentence ['while' 'removing' 'drill' 'rod' 'jumbo' '' 'maintenance' 'supervisor'
 'proceeds' 'loosen' 'support' 'intermediate' 'centralizer' 'facilitate'
 'removal' 'seeing' 'mechanic' 'support' 'one' 'end' 'drill' 'equipment'
 'pull' 'hand' 'bar' 'accelerate' 'removal' 'this' 'moment' 'bar' 'slide'
 'point' 'support' 'tightens' 'finger' 'mechanic' 'drilling' 'bar' 'beam'
 'jumbo']
```

CBOW:

In the CBOW model, the distributed representations of context (or surrounding words) are combined to predict the word in the middle. This model tries to understand the context of the words and takes this as input. It then tries to predict words that are contextually accurate.



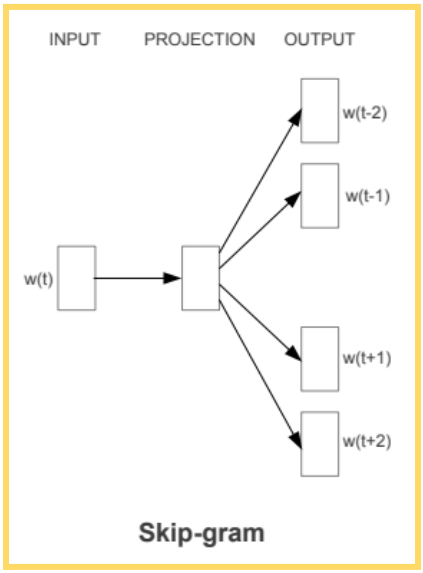
We have selected vector_size=100 . Below screenshot represents the CBOW model applied on the independent variable.

CBOW

```
In [51]: CBOWmodel = Word2Vec(sentences=training, min_count =2,sg=0,vector_size=100)
         print(CBOWmodel)
         # The trained word vectors are stored in a KeyedVectors instance, as model.wv
         print(CBOWmodel.wv['removing'].reshape(1,100)) # printing the 100 dimensional vector for first word
         vectors = CBOWmodel.wv.vectors # Storing the vectors of words which is trained on word2vec model
         tokenized_words = [i.split() for i in df['Cleaned Desc']]
         wordvec_arrays_CB = np.zeros((len(tokenized_words), 100))

         for i in range(len(tokenized_words)):
             wordvec_arrays_CB[i,:] = word_vector(CBOWmodel,tokenized_words[i], 100)

         wordvecCBOW_df = pd.DataFrame(wordvec_arrays_CB)
         print("Shape of CBOW model",wordvecCBOW_df.shape)

         X_train_CBOW, X_test_CBOW ,Y_train_CBOW,Y_test_CBOW= train_test_split(wordvecCBOW_df,Y, random_state=28, test_size=0.2)

         Word2Vec<vocab=1578, vector_size=100, alpha=0.025>
         [[-5.2536070e-02  4.4041775e-02  4.4438735e-02  2.8481623e-02
            1.0181993e-02 -1.3073681e-01  3.5021998e-02  1.7771196e-01
           -8.4058255e-02 -4.9166080e-02 -4.6569578e-02 -9.4332688e-02
           -1.0529362e-02  2.5269862e-02  5.7428777e-02 -4.6097059e-02
           -3.8951486e-03 -1.2669601e-01 -3.3499025e-02 -1.3104692e-01
            6.7745745e-02  3.1460695e-02  4.0525682e-02 -2.8795937e-02
           -1.6300384e-02 -1.0884155e-02 -5.7678435e-02 -7.2230287e-02
           -7.8621730e-02  3.4535157e-03  6.9671541e-02  1.7637255e-02
            3.0107856e-02 -4.7620438e-02 -3.3452000e-02  7.5077325e-02
```

Skip-GRAM:

The Skip-gram model architecture usually tries to achieve the reverse of what the CBOW model does. It tries to predict the source context words (surrounding words) given a target word (the center word).



Exploring the model by skipping 1- gram Below screenshot represents the Skip-Gram model applied on the independent variable.

Skip- GRAM

In [52]:
```
SkipGrammodel = Word2Vec(sentences=training, min_count =2,sg=1,vector_size=100)
print(SkipGrammodel)
# The trained word vectors are stored in a KeyedVectors instance, as model.wv
print(SkipGrammodel.wv['removing'].reshape(1,100)) # printing the 100 dimensional vector for first word
vectors = SkipGrammodel.wv.vectors # Storing the vectors of words which is trained on word2vec model
tokenized_words = [i.split() for i in df['Cleaned Desc']]
wordvec_arrays_sg = np.zeros((len(tokenized_words), 100))

for i in range(len(tokenized_words)):
    wordvec_arrays_sg[i,:] = word_vector(SkipGrammodel,tokenized_words[i], 100)

wordvecsg_df = pd.DataFrame(wordvec_arrays_sg)
X_train_SG, X_test_SG ,Y_train_SG,Y_test_SG= train_test_split(wordvecsg_df,Y, random_state=28, test_size=0.2)
```

```
Word2Vec<vocab=1578, vector_size=100, alpha=0.025>
[[-9.56720263e-02  1.32304609e-01  7.88415745e-02  9.58599970e-02
   2.34928411e-02 -2.55338132e-01  8.25216621e-02  4.03216273e-01
  -1.68989956e-01 -1.10606015e-01 -9.71852168e-02 -2.19066441e-01
  -3.63313816e-02  3.44090089e-02  1.21332392e-01 -9.21352580e-02
  -1.09549001e-01 -3.10132325e-01 -2.54149623e-02 -2.60484964e-01
```

Once the data is in usable shape, we have analyzed the data and the problem you are trying to solve, the next step is find the suitable model to train your data. This model is then used to compute prediction on the testing data and the results are evaluated using different error metrics.

Since our objective here is to predict the safety risk as per the accident description, we are considering various classification models of machine learning to classify the accident levels based on the user description.

As mentioned, our independent variable is the ‘Cleaned Desc’ column which is represented by X variable and target variable is ‘Accident Level’ represented by Y variable. We are splitting the data into 20% test and 80% train data in order to train the model and validate the model using the test data.

Below is the figure displaying X variable values after applying different data preprocessing techniques.

	Description	Cleaned Description	Bag of Words	TFIDF	CBOW	Skip-Gram
0	While removing the drill rod of the Jumbo 08 f...	while removing drill rod jumbo maintenance su...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.009821358187556533, -0.03424439897893795, 0...	[0.006641756437186684, -0.08056502033557211, 0...
1	During the activation of a sodium sulphide pum...	during activation sodium sulphide pump piping ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.006671975019708043, -0.019022620422038016, ...	[0.0083989896791536, -0.06394174806773663, 0.1...
2	In the sub-station MILPO located at level +170...	in substation milpo located level collaborato...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.012870737191406079, -0.04003890440383527, 0...	[0.006278332651293438, -0.06922777203310813, 0....
3	Being 9:45 am. approximately in the Nv. 1880 C...	being am approximately nv cx ob personnel be...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.009493149823589023, -0.03268715089354881, 0...	[0.005839022128236329, -0.0764739477877007, 0....
4	Approximately at 11:45 a.m. in circumstances t...	approximately am circumstance mechanic anthon...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.011707351938821376, -0.034826307795343125, ...	[0.005272399526022907, -0.08099419879061835, 0...
...	...	...	...	...	...	...
420	Being approximately 5:00 a.m. approximately, w...	being approximately am approximately lifting ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.011280051174080222, -0.034386607366503175, ...	[0.00655777992436751, -0.0766421365773394, 0.1...
421	The collaborator moved from the infrastructure...	the collaborator moved infrastructure office j...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.012497590096447513, -0.03952613833826035, 0...	[0.006247801078845643, -0.07021277252998617, 0....
422	During the environmental monitoring activity i...	during environmental monitoring activity area ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.013509365072853765, -0.04242452212584842, 0...	[0.008208392629105794, -0.0738680343094625, 0....
423	The Employee performed the activity of strippi...	the employee performed activity stripping cath...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.016543499733272352, -0.05034760429867004 4, ...	[0.008617289437863388, -0.07732686498447468, 0...

5. Model Building

5.1 Machine Learning Classifier

Below are the various Classification machine learning algorithms that can be used to train the given data set

5.1.1 Logistic Regression

The logistic model is a statistical model that models the probability of an event taking place by having the log-odds for the event be a linear combination of one or more independent variables.

Used data prepared with Bag of words in LR . We can see that the train score is 100% which seems to be an overfit of the model. The test score is 68% and after hyper parameter tuning with GridSearchCV of parameter penalty none and L2 . L2 seems to be the best parameter with a best score of 74.25% accuracy . Also did stratified K-fold cross validation with 6 k-fold and got the CV score of 74.25

Hyper Parameter explored	{"penalty" : ['none', 'l2']}
Best Parameter	{'penalty': 'l2'}
Technique used	GridsearchCV

5.1.2 K-Nearest Neighbour algorithm

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. To select the K that's right for our data, we run the KNN algorithm several times with different values of K and choose the K that reduces the number of errors we encounter while maintaining the algorithm's ability to accurately make predictions when it's given data it hasn't seen before.

Training the KNN model with data prepared using a bag of words, we can see that the train score is 75% and the test score is 68% . After hyper parameter tuning with GridSearchCV of parameter n_neighbors with range 1,20, the best parameter is n_neighbors as 3 with best score of 75.45% accuracy . Also did stratified K-fold cross validation with 6 k-fold and got the CV score of 75.45%.

Hyper Parameter explored	{'n_neighbors': range(1, 20, 2)}
Best Parameter	{'n_neighbors': 3}
Technique used	GridsearchCV

5.1.3 Naive Bayes

It is another popular classifier used in Data Science. The idea behind it is driven by Bayes Theorem which gives us the conditional probability of event A, given that event B has occurred.

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

where:

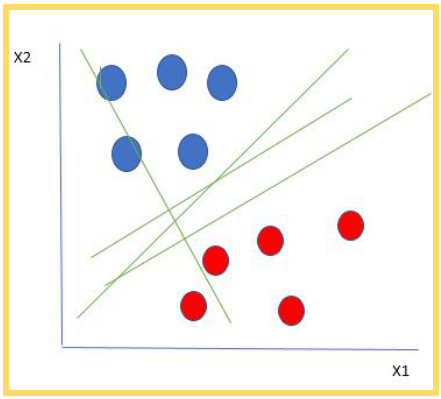
P(A|B) = Conditional Probability of A given B
P(B|A) = Conditional Probability of A given B
P(A) = Probability of event A
P(B) = Probability of event A

Training the Naive Bayes model with data prepared using a bag of words, we can see that the train score 100% which looks overfit and the test score is 65% . After hyper parameter tuning with GridSearchCV of parameter 'var_smoothing' with range 0,100, the best parameter is var_smoothing' as 4.3288 with best score of 75=4.26% accuracy . Also did stratified K-fold cross validation with 6 k-fold and got the CV score of 74.25%

Hyper Parameter explored	{'var_smoothing': np.logspace(0,-9, num=100)}
Best Parameter	'{var_smoothing': 4.328761281083062e-05}
Technique used	GridsearchCV

5.1.4 Support Vector Machine:

The objective of the SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.

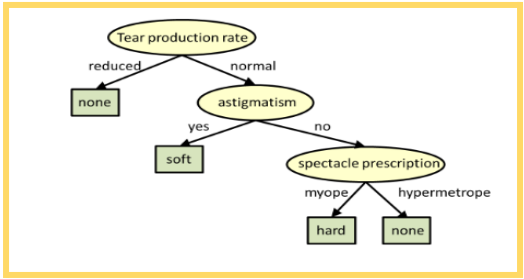


Training the SVM model with data prepared using a bag of words, we can see that the train score 83% and the test score is 68% . After hyper parameter tuning with GridSearchCV of parameter C-l2 regularization parameter with range 0.1to 0.5,kernel functions ‘rbf’.’sigmoid’.The best parameter are C=0.1, kernel-’rbf’ .The best score of 75.45% accuracy . Also did stratified K-fold cross validation with 6 k-fold and got the CV score of 74.85%

Hyper Parameter explored	{"C":[0.1, 0.5],"kernel":['rbf' , 'sigmoid']}
Best Parameter	{'C': 0.1, 'kernel': 'rbf'}
Technique used	GridsearchCV

5.1.5 Decision Tree classifier:

Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree. We can represent any boolean function on discrete attributes using the decision tree.CART (Classification and Regression Trees).



Training the Decision Tree model with data prepared using a bag of words, we can see that the train score 100% which seems to be an overfit and the test score is 63%. After

hyper parameter tuning with GridSearchCV of parameters max_depth with range 3 to 6 , min samples ranging from 2 to 6 ,criterion as gini and entropy..The best parameter are 'criterion': 'entropy', 'max_depth': 3, 'min_samples_leaf': 4 .The best score of 72.46% accuracy . Also did stratified K-fold cross validation with 6 k-fold and got the CV score of 62.07%

Hyper Parameter explored	{"max_depth":range(3,6,2),"min_samples_leaf":range(2,6,2),"criterion":["gini", ""]}
Best Parameter	{'criterion': 'entropy', 'max_depth': 3, 'min_samples_leaf': 4}
Technique used	GridsearchCV

5.1.6 AdaBoost Algorithm:

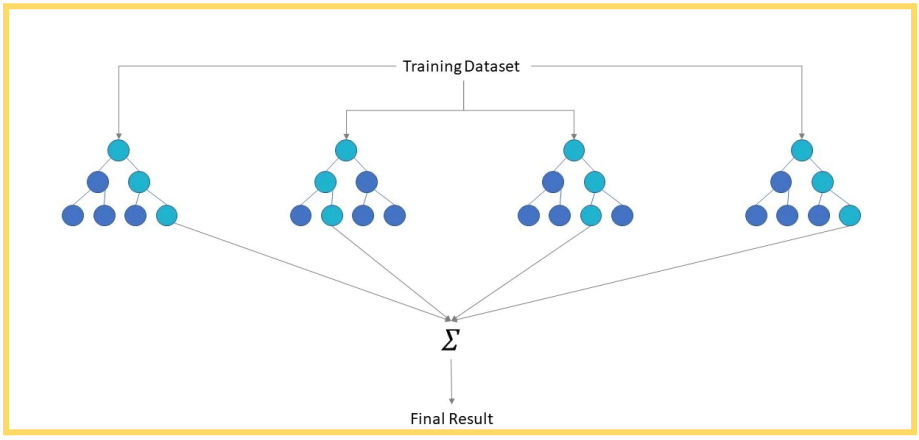
AdaBoost also called Adaptive Boosting is a technique in MachAdaBoost also called Adaptive Boosting is a technique in Machine Learning used as an Ensemble Method. The most common algorithm used with AdaBoost is decision trees with one level that means with Decision trees with only 1 split. This algorithm builds a model and gives equal weights to all the data points. It then assigns higher weights to points that are wrongly classified. In the next model, it will prioritize the points with higher weights. In this way, it will train the models until minimum error is seen.

Training the ADA Boost model with data prepared using a bag of words, we can see that the train score 77% and the test score is 65%. After hyper parameter tuning with GridSearchCV of parameters learning rate with range 0.3 to 0.8 , n_estimators ranging from 100 to 500. The best parameters are learning rate 0.3 and n_estimators as 100. The best score of 70.96% accuracy . Also did stratified K-fold cross validation with 6 k-fold and got the CV score of 72.48%

Hyper Parameter explored	{"learning_rate":[0.3,0.8],"n_estimators":[100,500]}
Best Parameter	{'learning_rate': 0.3, 'n_estimators': 100}
Technique used	GridsearchCV

5.1.7 Random Forest Classifier:

Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.



Training the Random Forest model with data prepared using a bag of words, we can see that the train score 100% which seems to be overfit and the test score is 68%. After hyper parameter tuning with GridSearchCV of parameters max_depth with range 3 to 7 , min_samples_leaf as 2,4,6 , 'min_samples_split' - 2,4,6 . The best parameters are max_depth': 3, 'min_samples_leaf': 4, 'min_samples_split': 2, 'n_estimators': 100. The best score of 75.45% accuracy. Also did stratified K-fold cross validation with 6 k-fold and got the CV score of 75.85%

Hyper Parameter explored	{ 'max_depth': [3,7,10],'min_samples_leaf': [4, 6],'min_samples_split': [2, 4, 6], 'n_estimators': [100, 200]}}
Best Parameter	{'max_depth': 3, 'min_samples_leaf': 4, 'min_samples_split': 2, 'n_estimators': 100}
Technique used	GridsearchCV

5. 2 Performance of Machine Learning Algorithms:

We have used various classifier machine learning algorithms to train the dataset and here are train and test scoring tables for the same.

5.2.1 Assumption 1: Predicting accident level with description

Scoring Table of bag of words

	Modelname	CV Score	Train Score	Test Score	Precision Score	Recall Score	F1 Score	Best Score	Best Parameter
0	Bag of Words - LR	74.25	1.00	0.68	0.67	0.68	0.55	74.25	{'penalty': 'l2'}
1	Bag of Words - KNN	75.45	0.75	0.68	0.78	0.68	0.55	75.45	{'n_neighbors': 3}
2	Bag of Words - NB	74.25	1.00	0.65	0.56	0.65	0.54	74.26	{'var_smoothing': 4.328761281083062e-05}
3	Bag of Words - SVM	74.85	0.83	0.68	0.78	0.68	0.55	75.45	{'C': 0.1, 'kernel': 'rbf'}
4	Bag of Words - CART	67.07	1.00	0.63	0.65	0.63	0.59	72.46	{'criterion': 'entropy', 'max_depth': 3, 'min_...
5	Bag of Words - AB	72.48	0.77	0.65	0.66	0.65	0.54	70.96	{'learning_rate': 0.3, 'n_estimators': 100}
6	Bag of Words - RF	74.85	1.00	0.68	0.67	0.68	0.55	75.45	{'max_depth': 3, 'min_samples_leaf': 4, 'min_s...

We can see that Logistic regression, KNN, SVM and Random Forest models have a test score of 0.68 which is the highest when compared to the rest of the models. We can see that LR, NB,Decision tree and RF are overfitting model

Scoring Table of TF-IDF

	Modelname	CV Score	Train Score	Test Score	Precision Score	Recall Score	F1 Score	Best Score	Best Parameter
0	TFIDF - LR	75.45	0.75	0.68	0.78	0.68	0.55	75.45	{'penalty': 'l2'}
1	TFIDF - KNN	73.65	0.76	0.62	0.63	0.62	0.52	75.45	{'n_neighbors': 7}
2	TFIDF - NB	15.86	0.17	0.26	0.68	0.26	0.28	74.26	{'var_smoothing': 0.0001873817422860383}
3	TFIDF - SVM	75.45	0.75	0.68	0.78	0.68	0.55	75.45	{'C': 0.1, 'kernel': 'rbf'}
4	TFIDF - CART	56.00	1.00	0.52	0.49	0.52	0.49	73.35	{'criterion': 'gini', 'max_depth': 3, 'min_sam...
5	TFIDF - AB	74.86	0.75	0.68	0.78	0.68	0.55	74.26	{'learning_rate': 0.3, 'n_estimators': 100}
6	TFIDF - RF	74.56	1.00	0.67	0.56	0.67	0.55	75.45	{'max_depth': 3, 'min_samples_leaf': 4, 'min_s...

We can observe that the highest test accuracy obtained is 0.68 using Logistic regression,SVM and Ada-Boosting model for the TF-IDF . Decision tree and Random forest are overfitting

Scoring Table of CBOW

	Modelname	CV Score	Train Score	Test Score	Precision Score	Recall Score	F1 Score	Best Score	Best Parameter
0	CBOW - LR	75.45	0.75	0.68	0.78	0.68	0.55	75.45	{'penalty': 'l2'}
1	CBOW - KNN	73.95	0.76	0.62	0.63	0.62	0.52	75.45	{'n_neighbors': 9}
2	CBOW - NB	14.66	0.17	0.26	0.68	0.26	0.28	15.00	{'var_smoothing': 6.579332246575683e-05}
3	CBOW - SVM	75.45	0.75	0.68	0.78	0.68	0.55	75.45	{'C': 0.1, 'kernel': 'rbf'}
4	CBOW - CART	55.05	1.00	0.52	0.50	0.52	0.51	74.55	{'criterion': 'entropy', 'max_depth': 3, 'min_...
5	CBOW - AB	75.45	0.75	0.68	0.78	0.68	0.55	47.60	{'learning_rate': 0.3, 'n_estimators': 100}
6	CBOW - RF	74.55	1.00	0.68	0.67	0.68	0.55	75.45	{'max_depth': 3, 'min_samples_leaf': 4, 'min_s...

We can observe that the highest test accuracy obtained using CBOW is also 0.68. Logistic regression, SVM, Ada-boosting and random forest algorithms.

Scoring Table of Skip1-Gram

	Modelname	CV Score	Train Score	Test Score	Precision Score	Recall Score	F1 Score	Best Score	Best Parameter
0	Skip_1Gram - LR	75.45	0.75	0.68	0.78	0.68	0.55	75.45	{'penalty': 'l2'}
1	Skip_1Gram - KNN	73.35	0.77	0.62	0.49	0.62	0.53	75.45	{'n_neighbors': 11}
2	Skip_1Gram - NB	20.96	0.21	0.12	0.44	0.12	0.11	21.82	{'var_smoothing': 1.0}
3	Skip_1Gram - SVM	75.45	0.75	0.68	0.78	0.68	0.55	75.45	{'C': 0.1, 'kernel': 'rbf'}
4	Skip_1Gram - CART	53.65	1.00	0.48	0.45	0.48	0.46	74.85	{'criterion': 'entropy', 'max_depth': 3, 'min_...
5	Skip_1Gram - AB	69.79	0.48	0.40	0.56	0.40	0.42	70.36	{'learning_rate': 0.3, 'n_estimators': 100}
6	Skip_1Gram - RF	74.85	1.00	0.68	0.67	0.68	0.55	75.45	{'max_depth': 3, 'min_samples_leaf': 4, 'min_s...

We can observe that the highest accuracy obtained using the Skip1-Gram applied independent variable is also 0.68. Logistic regression, SVM, and random forest algorithms are giving the highest accuracy for the dataset

Scoring Table Summary for assumption 1 [Predicting accident level based on description]

	Modelname	CV Score	Train Score	Test Score	Precision Score	Recall Score	F1 Score	Best Score	Best Parameter
0	Bag of Words - LR	74.25	1.00	0.68	0.67	0.68	0.55	74.25	{'penalty': 'l2'}
1	Bag of Words - KNN	75.45	0.75	0.68	0.78	0.68	0.55	75.45	{'n_neighbors': 3}
2	Bag of Words - NB	74.25	1.00	0.65	0.56	0.65	0.54	74.26	{'var_smoothing': 4.328761281083062e-05}
3	Bag of Words - SVM	74.85	0.83	0.68	0.78	0.68	0.55	75.45	{'C': 0.1, 'kernel': 'rbf'}
4	Bag of Words - CART	67.07	1.00	0.63	0.65	0.63	0.59	72.46	{'criterion': 'entropy', 'max_depth': 3, 'min_...
5	Bag of Words - AB	72.48	0.77	0.65	0.66	0.65	0.54	70.96	{'learning_rate': 0.3, 'n_estimators': 100}
6	Bag of Words - RF	74.85	1.00	0.68	0.67	0.68	0.55	75.45	{'max_depth': 3, 'min_samples_leaf': 4, 'min_s...
7	TFIDF - LR	75.45	0.75	0.68	0.78	0.68	0.55	75.45	{'penalty': 'l2'}
8	TFIDF - KNN	73.65	0.76	0.62	0.63	0.62	0.52	75.45	{'n_neighbors': 7}
9	TFIDF - NB	15.86	0.17	0.26	0.68	0.26	0.28	74.26	{'var_smoothing': 0.0001873817422860383}
10	TFIDF - SVM	75.45	0.75	0.68	0.78	0.68	0.55	75.45	{'C': 0.1, 'kernel': 'rbf'}
11	TFIDF - CART	56.00	1.00	0.52	0.49	0.52	0.49	73.35	{'criterion': 'gini', 'max_depth': 3, 'min_sam...
12	TFIDF - AB	74.86	0.75	0.68	0.78	0.68	0.55	74.26	{'learning_rate': 0.3, 'n_estimators': 100}
13	TFIDF - RF	74.56	1.00	0.67	0.56	0.67	0.55	75.45	{'max_depth': 3, 'min_samples_leaf': 4, 'min_s...
14	CBOW - LR	75.45	0.75	0.68	0.78	0.68	0.55	75.45	{'penalty': 'l2'}
15	CBOW - KNN	73.95	0.76	0.62	0.63	0.62	0.52	75.45	{'n_neighbors': 9}
16	CBOW - NB	14.66	0.17	0.26	0.68	0.26	0.28	15.00	{'var_smoothing': 6.579332246575683e-05}
17	CBOW - SVM	75.45	0.75	0.68	0.78	0.68	0.55	75.45	{'C': 0.1, 'kernel': 'rbf'}
18	CBOW - CART	55.05	1.00	0.52	0.50	0.52	0.51	74.55	{'criterion': 'entropy', 'max_depth': 3, 'min_...
19	CBOW - AB	75.45	0.75	0.68	0.78	0.68	0.55	47.60	{'learning_rate': 0.3, 'n_estimators': 100}
20	CBOW - RF	74.55	1.00	0.68	0.67	0.68	0.55	75.45	{'max_depth': 3, 'min_samples_leaf': 4, 'min_s...
21	Skip_1Gram - LR	75.45	0.75	0.68	0.78	0.68	0.55	75.45	{'penalty': 'l2'}
22	Skip_1Gram - KNN	73.35	0.77	0.62	0.49	0.62	0.53	75.45	{'n_neighbors': 11}
23	Skip_1Gram - NB	20.96	0.21	0.12	0.44	0.12	0.11	21.82	{'var_smoothing': 1.0}
24	Skip_1Gram - SVM	75.45	0.75	0.68	0.78	0.68	0.55	75.45	{'C': 0.1, 'kernel': 'rbf'}
25	Skip_1Gram - CART	53.65	1.00	0.48	0.45	0.48	0.46	74.85	{'criterion': 'entropy', 'max_depth': 3, 'min_...
26	Skip_1Gram - AB	69.79	0.48	0.40	0.56	0.40	0.42	70.36	{'learning_rate': 0.3, 'n_estimators': 100}
27	Skip_1Gram - RF	74.85	1.00	0.68	0.67	0.68	0.55	75.45	{'max_depth': 3, 'min_samples_leaf': 4, 'min_s...

Top 5 performing models for assumption 1

All this top 5 performing model have same score

```
In [72]: Mlmodel.sort_values(['CV Score','Precision Score'],ascending = [False, False]).head()
```

Out[72]:

	Modelname	CV Score	Train Score	Test Score	Precision Score	Recall Score	F1 Score	Best Score	Best Parameter
1	Bag of Words - KNN	75.45	0.75	0.68	0.78	0.68	0.55	75.45	{'n_neighbors': 3}
7	TFIDF - LR	75.45	0.75	0.68	0.78	0.68	0.55	75.45	{'penalty': 'l2'}
10	TFIDF - SVM	75.45	0.75	0.68	0.78	0.68	0.55	75.45	{'C': 0.1, 'kernel': 'rbf'}
14	CBOW - LR	75.45	0.75	0.68	0.78	0.68	0.55	75.45	{'penalty': 'l2'}
17	CBOW - SVM	75.45	0.75	0.68	0.78	0.68	0.55	75.45	{'C': 0.1, 'kernel': 'rbf'}

```
In [89]: # best model for predicting accident level
         Mlmodel.iloc[0]
```

```
Out[89]: Modelname          Bag of Words - LR
         CV Score                       74.25
         Train Score                      1.0
         Test Score                      0.68
         Precision Score                 0.67
         Recall Score                    0.68
         F1 Score                        0.55
         Best Score                     74.25
         Best Parameter     {'penalty': 'l2'}
         Name: 0, dtype: object
```

Obesrvations:

- We can observe that the top 5 performing models have the same score of CV=75.45% and Precision is 78% with test score 68%
- We can say that KNN model with Bag of words, Logistic Regression and Support Vector Machine model with TF-IDF , Logistic Regression and Support Vector Machine with CBOW are the preferred models to train the dataset to predict the target variable Accident Level accurately.
- Considering time complexity Bag of words - Logistic regression can be considered as best model

5.2.2 Assumption 2: - Predicting potential accident level with description column

X variable is cleaned description and y variable is Potential accident Level

Scoring Table Summary for assumption 2 [Predicting potential accident level based on description]

	Modelname	CV Score	Train Score	Test Score	Precision Score	Recall Score	F1 Score
0	Bag of Words - LR	41.90	1.00	0.43	0.45	0.43	0.43
1	Bag of Words - KNN	22.47	0.35	0.26	0.64	0.26	0.18
2	Bag of Words - NB	43.43	1.00	0.39	0.40	0.39	0.36
3	Bag of Words - SVM	39.53	0.92	0.37	0.50	0.37	0.35
4	Bag of Words - CART	34.09	1.00	0.32	0.33	0.32	0.31
5	Bag of Words - AB	32.66	0.38	0.31	0.49	0.31	0.24
6	Bag of Words - RF	41.02	1.00	0.43	0.43	0.43	0.35
7	TFIDF - LR	33.24	0.34	0.35	0.66	0.35	0.22
8	TFIDF - KNN	28.73	0.54	0.35	0.33	0.35	0.33
9	TFIDF - NB	24.24	0.25	0.23	0.68	0.23	0.15
10	TFIDF - SVM	33.84	0.34	0.33	0.78	0.33	0.17
11	TFIDF - CART	28.79	1.00	0.27	0.27	0.27	0.27
12	TFIDF - AB	29.06	0.31	0.21	0.20	0.21	0.19
13	TFIDF - RF	31.75	1.00	0.39	0.40	0.39	0.34
14	CBOW - LR	33.24	0.34	0.35	0.66	0.35	0.22
15	CBOW - KNN	30.22	0.54	0.35	0.33	0.35	0.33
16	CBOW - NB	23.95	0.25	0.23	0.68	0.23	0.15
17	CBOW - SVM	33.84	0.34	0.33	0.78	0.33	0.17
18	CBOW - CART	32.32	1.00	0.21	0.22	0.21	0.22
19	CBOW - AB	26.94	0.31	0.21	0.20	0.21	0.19

20	CBOW - RF	32.02	1.00	0.40	0.41	0.40	0.35
21	Skip_1Gram - LR	33.84	0.34	0.33	0.78	0.33	0.17
22	Skip_1Gram - KNN	36.56	0.60	0.32	0.31	0.32	0.30
23	Skip_1Gram - NB	23.36	0.27	0.23	0.51	0.23	0.19
24	Skip_1Gram - SVM	33.84	0.34	0.33	0.78	0.33	0.17
25	Skip_1Gram - CART	31.46	1.00	0.24	0.26	0.24	0.24
26	Skip_1Gram - AB	29.97	0.39	0.29	0.35	0.29	0.26
27	Skip_1Gram - RF	42.23	1.00	0.35	0.36	0.35	0.31

Top 5 performing models for assumption 2

```
In [76]: Mlmodel1.sort_values(['CV Score','Precision Score'],ascending = [False, False]).head()
```

Out[76]:

	Modelname	CV Score	Train Score	Test Score	Precision Score	Recall Score	F1 Score
2	Bag of Words - NB	43.43	1.00	0.39	0.40	0.39	0.36
27	Skip_1Gram - RF	42.23	1.00	0.35	0.36	0.35	0.31
0	Bag of Words - LR	41.90	1.00	0.43	0.45	0.43	0.43
6	Bag of Words - RF	41.02	1.00	0.43	0.43	0.43	0.35
3	Bag of Words - SVM	39.53	0.92	0.37	0.50	0.37	0.35

Best Model for Assumption 2

Bag of words of RF , Naive Bayes and Logistic regression are overfitting and thier precision score is lesser than SVM .

Hence,Bag of Words - SVM is best model for predicting potential accident level with description

```
In [77]: Mlmodel1.iloc[3]
```

```
Out[77]: Modelname          Bag of Words - SVM
         CV Score                       39.53
         Train Score                     0.92
         Test Score                      0.37
         Precision Score                  0.5
         Recall Score                    0.37
         F1 Score                        0.35
         Name: 3, dtype: object
```

5.2.3 Assumption 3 - Predicting potential accident level with all the columns

X = is combined data from all the columns except Date and accident level column, y(target variable) is potential accident level

Scoring Table Summary for assumption 3 [Predicting accident level based on description]

	Modelname	CV Score	Train Score	Test Score	Precision Score	Recall Score	F1 Score
0	Bag of Words - LR	45.21	1.00	0.40	0.42	0.40	0.41
1	Bag of Words - KNN	27.26	0.42	0.24	0.56	0.24	0.16
2	Bag of Words - NB	44.31	1.00	0.39	0.44	0.39	0.37
3	Bag of Words - SVM	42.53	0.95	0.40	0.52	0.40	0.40
4	Bag of Words - CART	43.13	1.00	0.36	0.35	0.36	0.35
5	Bag of Words - AB	34.71	0.39	0.32	0.46	0.32	0.25
6	Bag of Words - RF	45.81	1.00	0.42	0.42	0.42	0.36
7	TFIDF - LR	33.24	0.34	0.35	0.66	0.35	0.22
8	TFIDF - KNN	27.54	0.54	0.35	0.33	0.35	0.33
9	TFIDF - NB	24.24	0.25	0.23	0.68	0.23	0.15
10	TFIDF - SVM	33.84	0.34	0.33	0.78	0.33	0.17
11	TFIDF - CART	25.17	1.00	0.25	0.25	0.25	0.24
12	TFIDF - AB	26.98	0.35	0.37	0.37	0.37	0.36
13	TFIDF - RF	31.13	1.00	0.39	0.42	0.39	0.35
14	CBOW - LR	33.24	0.34	0.35	0.66	0.35	0.22
15	CBOW - KNN	30.51	0.54	0.35	0.33	0.35	0.33
16	CBOW - NB	23.95	0.25	0.23	0.68	0.23	0.15
17	CBOW - SVM	33.84	0.34	0.33	0.78	0.33	0.17
18	CBOW - CART	29.63	1.00	0.24	0.24	0.24	0.23
19	CBOW - AB	29.62	0.35	0.37	0.37	0.37	0.36
20	CBOW - RF	31.74	1.00	0.40	0.43	0.40	0.35
21	Skip_1Gram - LR	33.84	0.34	0.33	0.78	0.33	0.17
22	Skip_1Gram - KNN	36.57	0.59	0.35	0.35	0.35	0.33
23	Skip_1Gram - NB	23.37	0.28	0.23	0.51	0.23	0.19
24	Skip_1Gram - SVM	33.84	0.34	0.33	0.78	0.33	0.17
25	Skip_1Gram - CART	31.75	1.00	0.30	0.29	0.30	0.29
26	Skip_1Gram - AB	29.06	0.40	0.30	0.27	0.30	0.28
27	Skip_1Gram - RF	37.75	1.00	0.36	0.37	0.36	0.34

Top 5 performing models for assumption 3

```
In [87]: Mlmodel2.sort_values(['CV Score','Precision Score'],ascending = [False, False]).head()
```

Out[87]:

	Modelname	CV Score	Train Score	Test Score	Precision Score	Recall Score	F1 Score
6	Bag of Words - RF	45.81	1.00	0.42	0.42	0.42	0.36
0	Bag of Words - LR	45.21	1.00	0.40	0.42	0.40	0.41
2	Bag of Words - NB	44.31	1.00	0.39	0.44	0.39	0.37
4	Bag of Words - CART	43.13	1.00	0.36	0.35	0.36	0.35
3	Bag of Words - SVM	42.53	0.95	0.40	0.52	0.40	0.40

Best Model for Assumption 3

Bag of words of RF , Naive Bayes and Logistic regression are overfitting and thier precision score is lesser than SVM .

Hence,Bag of Words - SVM is best model for predicting potential accident level with all the x variables

```
In [88]: Mlmodel2.iloc[3]
```

```
Out[88]: Modelname          Bag of Words - SVM
         CV Score                       42.53
         Train Score                     0.95
         Test Score                       0.4
         Precision Score                 0.52
         Recall Score                     0.4
         F1 Score                         0.4
         Name: 3, dtype: object
```

Summary:

In Assumption 2, considering the only description as X , the SVM model was able to predict the potential accident levels with 39.53%. However when all the columns are considered for modeling, as in Assumption 3, the performance of the model increases to 42.53 % in predicting the potential accident level. So we can consider this approach to further improve the model.

6. Improvement Suggestions

While predicting potential accident level , when we choose the description alone as x variable our score is 39% only , when we include all the other dependent variables we are able to improve the score +4% [approx] .

One of the main reasons for not achieving very accuracy could be the lack of a large dataset. Most of the labeled text datasets are not big enough to train deep neural networks because these networks have a huge number of parameters and training such networks on small datasets will cause overfitting.

We can further improve the model by doing “Target balance using SMOTE technique”. In upcoming weeks we will try the neural networks , RNN and LSTM to improve the model accuracy .

We are also aware that NLP models are typically shallow and thus require different fine-tuning methods. BERT is a big neural network architecture with Millions of parameters. So, Training a BERT model from scratch on a small dataset would result in overfitting.So , we propose to use a pre-trained BERT model that was trained on a huge dataset, as a starting point and then we can further train the model on our relatively smaller dataset

Milestone 2 :

Objective:

- ★ To perform target balancing inorder to properly train the model.
- ★ To train the model using neural networks.
- ★ To fine tune the trained models using neural networks.
- ★ To design, train and test RNN and LSTM classifiers to best classify the potential accident level.

As mentioned in milestone 1, one of the main reasons to not obtain a good score while building the machine learning classifier models is lack of a large dataset. We see that there are only 334 trainabe rows in the data set. This bias in the training dataset can influence many machine learning algorithms, leading some to ignore the minority class entirely. This is a problem as it is typically the minority class on which predictions are most important.One approach to addressing the problem of class imbalance is to randomly resample the training dataset.

7. Model Improvement and Evaluation:

7.1 Random resampling:

provides a naive technique for rebalancing the class distribution for an imbalanced dataset.

- ❖ Random oversampling:
  involves randomly duplicating examples from the minority class and adding them to the training dataset. Examples from the training dataset are selected randomly with replacement.
  In some cases, seeking a balanced distribution for a severely imbalanced dataset can cause affected algorithms to overfit the minority class, leading to increased generalization error. The effect can be better performance on the training dataset, but worse performance on the holdout or test dataset.

- ❖ Random undersampling
  deletes examples from the majority class and can result in losing information invaluable to a model. This has the effect of reducing the number of examples in the majority class in the transformed version of the training dataset. This process can be repeated until the desired class distribution is achieved, such as an equal number of examples for each class.

We are preferring the random over sampling method as the model requires more samples to train on imbalanced class distribution. SMOTE is one of the techniques for Random Oversampling. We have used SMOTE to

We can see from the below screenshot that the target levels are imbalanced.

```
# Applying SMOTE to over sample the Minority class and then to under sample the Majority class

from collections import Counter
from sklearn.datasets import make_classification
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
from matplotlib import pyplot
from numpy import where

over = SMOTE(sampling_strategy={1: 140,2: 140, 3: 140, 5: 140})
under = RandomUnderSampler(sampling_strategy={4: 135})

steps = [('o', over), ('u', under)]
pipeline = Pipeline(steps=steps)
# transform the dataset
Xs, ys = pipeline.fit_resample(Xc, y)
# summarize the new class distribution
counter = Counter(y)
print(counter)
X_trains, X_tests, y_trains, y_tests = train_test_split(Xs, ys, test_size = 0.2, random_state =1, shuffle=True)
```

```
Counter({4: 141, 3: 106, 2: 95, 1: 45, 5: 31})
```

- **SMOTE (Synthetic Minority Oversampling Technique)** :

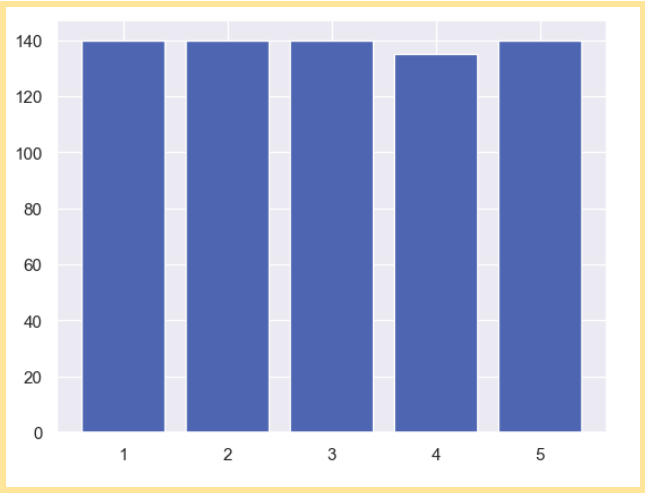
synthesizes new minority instances between existing minority instances. It randomly picks up the minority class and calculates the K-nearest neighbor for that particular point. Finally, the synthetic points are added between the neighbors and the chosen spot.

Below screenshot gives the target class count after applying the SMOTE to balance the target class by oversampling.

```
from collections import Counter
from matplotlib import pyplot
counter = Counter(ys)
for k, v in counter.items():
  per = v/len(ys)*100
  print('Class=%d, n=%d (%.3f%%)' % (k, v, per))
# plot the distribution
pyplot.bar(counter.keys(), counter.values())
pyplot.show()
```

```
Class=1, n=140 (20.144%)
Class=2, n=140 (20.144%)
Class=3, n=140 (20.144%)
Class=4, n=135 (19.424%)
Class=5, n=140 (20.144%)
```

Fig shows that target classes are balanced with equal number of samples



Screenshot of the test and train scores after oversampling.

```
smotemodelscore['Modelname']='Smote - '+smotemodelscore['Modelname']
smotemodelscore
```

	Modelname	CV Score	Train Score	Test Score	Precision Score	Recall Score	F1 Score
0	Smote - LR	69.06	0.93	0.72	0.73	0.72	0.72
1	Smote - KNN	56.29	0.68	0.60	0.71	0.60	0.53
2	Smote - NB	75.91	0.94	0.80	0.81	0.80	0.80
3	Smote - SVM	77.53	1.00	0.81	0.84	0.81	0.81
4	Smote - CART	62.41	1.00	0.61	0.60	0.61	0.60
5	Smote - AB	37.24	0.54	0.53	0.59	0.53	0.52
6	Smote - RF	70.15	1.00	0.76	0.78	0.76	0.77

We can see that after oversampling the accuracy of most of the models have increased by double. SVM model performs far better than all with 81% precision and 79.8% test accuracy.

With SMote accuracy of the model is improved twice the previous model.

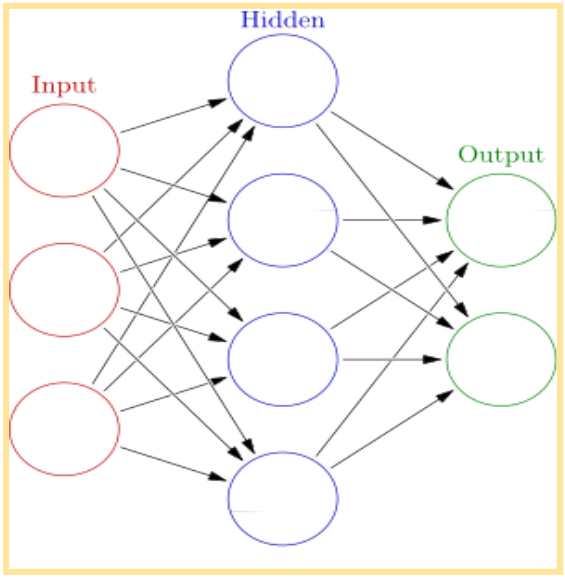
Among them SVM performs far better than all with 81% precision score and 79.8% test accuracy

```
1 smotemodelscore.iloc[3]
```

```
Modelname          Smote - SVM
CV Score                 77.53
Train Score                1.0
Test Score                0.81
Precision Score           0.84
Recall Score              0.81
F1 Score                  0.81
Name: 3, dtype: object
```

7.2 Neural Networks:

Neural network is a computational learning system that uses a network of functions to understand and translate a data input of one form into a desired output, usually in another form. The concept of the artificial neural network was inspired by human biology and the way neurons of the human brain function.



A neural network consists of artificial neurons or processing elements and is organized in three interconnected layers: input, hidden that may include more than one layer, and output.

The ***input*** layer contains input neurons that send information to the hidden layer. The ***hidden*** layer sends data to the ***output*** layer. Every neuron has weighted inputs (synapses), an activation function (defines the output given an input), and one output. Synapses are the adjustable parameters that convert a neural network to a parameterized system.

The weighted sum of the inputs produces the activation signal that is passed to the activation function to obtain one

output from the neuron. The commonly used activation functions are linear, step, sigmoid, tanh, and rectified linear unit (ReLu) functions.

GloVe embedding

GloVe stands for ***Global Vectors for word representation***. It is an unsupervised learning algorithm developed by researchers at Stanford University aiming to generate word embeddings by aggregating global word co-occurrence matrices from a given corpus.

In other words, GloVe is a word vector technique that rides the wave of word vectors after a brief silence. Just to refresh, word vectors put words to a nice vector space, where similar words cluster together and different words repel.

The advantage of GloVe is that, unlike Word2vec, GloVe does not rely just on local statistics (local context information of words), but incorporates global statistics (word co-occurrence) to obtain word vectors.

```
import gensim
import gensim.downloader as api
from gensim.scripts.glove2word2vec import glove2word2vec
from gensim.models import Word2Vec, KeyedVectors
# Glove file - we are using model with 200 embeddings
glove_input_file = 'glove.6B.200d.txt'

# Name for word2vec file
word2vec_output_file = 'glove.6B.200d.txt.word2vec'

# Converting glove embedding to word2vec embedding
glove2word2vec(glove_input_file, word2vec_output_file)
glove_model = KeyedVectors.load_word2vec_format(word2vec_output_file, binary=False)
# Getting Pre-trained embedding
embedding_vector_length= glove_model.vector_size
print("embedding_vector_length",embedding_vector_length)
vocab_size = len(tokenizer.word_index)+1
print("vocab_size",vocab_size)
num_words = min(max_features, vocab_size)
print("num_words",num_words)
embedding_matrix = np.zeros((num_words, embedding_vector_length))
print("embedding_matrix shape",embedding_matrix.shape)
```

We can use the already trained model which is hosted at https://nlp.stanford.edu/projects/glove/ We can download the zip files from the above URL. There are five text files representing the word and its corresponding vectors in various dimensions like 50 dimensions, 100 dimensions etc. We are using '***glove.6B.200d.text'*** here, 200d refers to 200 dimensions for each word.

```
embedding_vector_length 200
vocab_size 2720
num_words 2720
embedding_matrix shape (2720, 200)
```

The embedding vector length is 200, vocabulary size is 2720 and number of words is 2720. The embedding matrix shape is 2720X 200.

Sequential Model

A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.

- **Dense Layer:** dense() is an inbuilt function of Tensorflow. js library. This function is used to create fully connected layers, in which every output depends on every input.

  Syntax: tf.layers.dense(args).

  It is a simple layer of neurons in which each neuron receives input from all the neurons of the previous layer, thus called dense. Dense Layer is used to classify images based on output from convolutional layers.

- **Activation functions** are mathematical equations that determine the output of a neural network model. Activation functions also have a major effect on the neural network's ability to converge and the convergence speed, or in some cases, activation functions might prevent neural networks from converging in the first place. Activation function also helps to normalize the output of any input in the range between 1 to -1 or 0 to 1.

- **Relu:** The sigmoid and hyperbolic tangent activation functions cannot be used in networks with many layers due to the vanishing gradient problem. The rectified linear activation function overcomes the vanishing gradient problem, allowing models to learn faster and perform better. The rectified linear activation is the default activation when developing multilayer Perceptron and convolutional neural networks.Though it looks like a linear function, it's not. ReLU has a derivative function and allows for backpropagation.

- **Softmax** is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector. The most common use of the softmax function in applied machine learning is in its use as an activation function in a neural network model. Each value in the output of the softmax function is interpreted as the probability of membership for each class the cross entropy loss. The purpose of loss functions is to compute the quantity that a model should seek to minimize during training.
- **Optimizers** help us to reduce the value of the cost function used in the model. The cost function is nothing but the error function which we want to reduce during the model building and largely depends on the model's internal parameters.
  - ***Adam optimization*** is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.This metric creates two local variables, total and count that are used to compute the frequency with which y_pred matches y_true. This frequency is ultimately returned as binary accuracy: an idempotent operation that simply divides total by count.

One of the most common problems of data science professionals is to avoid over-fitting. It is a situation when your model is performing very well on the training data but is unable to predict the test data accurately. The regularization techniques help to improve a model and allows it to converge faster. The regularization helps in preventing the over-fitting of the model and the learning process becomes more efficient. Below are some of the techniques used.

- **Dropout** is a regularization technique were randomly selected neurons are ignored during training. They are “dropped out” randomly. This means that their contribution to the activation of downstream neurons is temporarily removed on the forward pass, and any weight updates are not applied to the neuron on the backward pass.

- **Batch normalization** applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.Importantly, batch normalization works differently during training and during inference.

- **Early stopping** is a form of regularization used to avoid overfitting when training a learner with an iterative method, such as gradient descent. Such methods update the learner so as to make it better fit the training data with each iteration.

- A **callback function** is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or actionUsing callback function to stop the model, as the loss is not reducing or accuracy is not improving

- ➔ We have used the sequential model to train the embedded vectorised dataset.
- ➔ The model contains one input layer with 100 input parameters and activation function as ‘relu’.
- ➔ There are 8 hidden layers with a combination of batch normalization and dropout layers to best train the model without an overfit. ‘Relu’ activation function is used in all hidden layers.
- ➔ The Output layer is defined with 5 out variables which are the potential accident levels that are to be predicted. ‘Softmax’ activation function is used in the output layer.

Below code snippet can be referred for the same.

```
nn_model = Sequential()
# Embedding layer
nn_model.add(Embedding(input_dim= num_words, output_dim= embedding_vector_length,weights = [embedding_matrix],trainable = False,input_length = maxlen))
# Flatten the data as will use Dense layer
nn_model.add(Flatten())

# Adding Hidden Layers(Dense layers)
nn_model.add(Dense(100, activation='relu', input_shape=()))
nn_model.add(Dropout(0.4))
nn_model.add(BatchNormalization())
nn_model.add(Dense(50, activation='relu'))
nn_model.add(Dropout(0.4))
nn_model.add(BatchNormalization())
nn_model.add(Dense(25, activation='relu'))
nn_model.add(Dropout(0.4))
# Adding output layer
nn_model.add(Dense(5, activation='softmax'))
# Compiling the model
nn_model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
nn_model.summary()
```

Below is the summary of the model with trainable and non- trainable parameters

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 99, 200)           544000

 flatten_3 (Flatten)         (None, 19800)             0

 dense_13 (Dense)            (None, 100)               1980100

 dropout_10 (Dropout)        (None, 100)               0

 batch_normalization_6 (Batc  (None, 100)              400
 hNormalization)

 dense_14 (Dense)            (None, 50)                5050

 dropout_11 (Dropout)        (None, 50)                0

 batch_normalization_7 (Batc  (None, 50)               200
 hNormalization)

 dense_15 (Dense)            (None, 25)                1275

 dropout_12 (Dropout)        (None, 25)                0

 dense_16 (Dense)            (None, 5)                 130
```

```
=================================================
Total params: 2,531,155
Trainable params: 1,986,855
Non-trainable params: 544,300
_________________________________________________
```

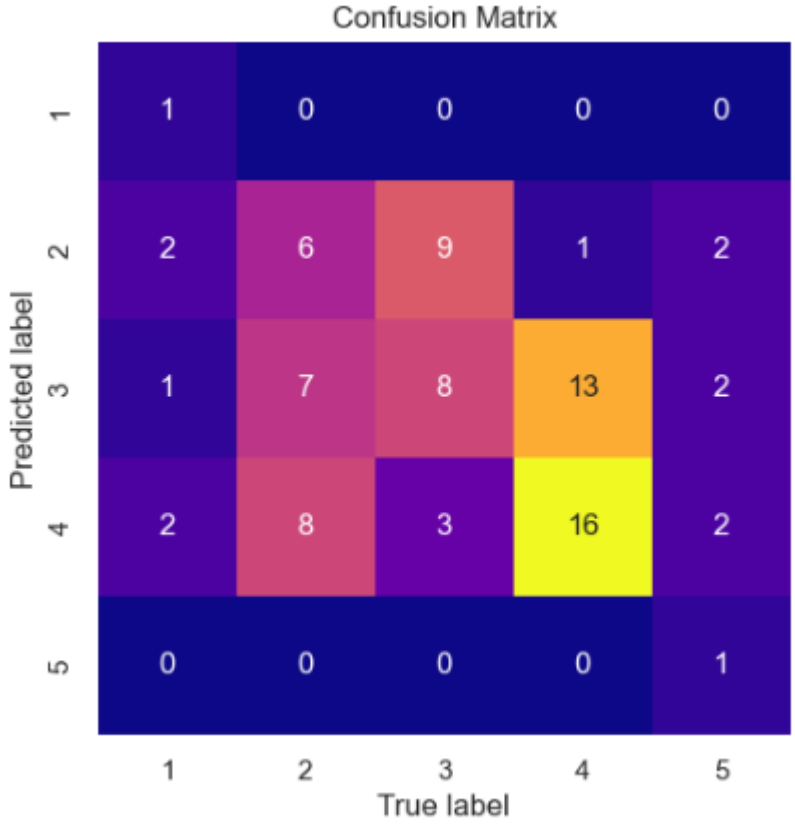
Below code snippet shows the early stopping and call back functions used.

```
# Using callback function to stop the model the loss is not reducing or accuracy is not improving
early = EarlyStopping(monitor='val_loss', patience=7, verbose=1,min_delta=0.0001,  mode='auto') # min_delta=0.0
reduce_learning = ReduceLROnPlateau(patience=5, verbose=1, min_lr=1e-6, factor=0.2)
callback_list = [early, reduce_learning]
nn_model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=100, batch_size=32, callbacks= [callbac
```

```
Epoch 1/100
11/11 [==============================] - 3s 56ms/step - loss: 2.7557 - accuracy: 0.2395 - val_loss: 2.5036 - val_a
ccuracy: 0.2500 - lr: 0.0010
Epoch 2/100
11/11 [==============================] - 0s 31ms/step - loss: 2.3522 - accuracy: 0.2515 - val_loss: 2.2291 - val_a
ccuracy: 0.2500 - lr: 0.0010
Epoch 3/100
11/11 [==============================] - 0s 30ms/step - loss: 2.0883 - accuracy: 0.2904 - val_loss: 1.9242 - val_a
ccuracy: 0.2500 - lr: 0.0010
Epoch 4/100
11/11 [==============================] - 0s 30ms/step - loss: 2.0206 - accuracy: 0.3024 - val_loss: 1.6968 - val_a
ccuracy: 0.2619 - lr: 0.0010
Epoch 5/100
11/11 [==============================] - 0s 30ms/step - loss: 1.8843 - accuracy: 0.3323 - val_loss: 1.5686 - val_a
ccuracy: 0.3214 - lr: 0.0010
Epoch 6/100
```

We can see from the below confusion matrix that the target class is predicted correctly for potential accident level I, 6 for level II,8 for level III, 16 for level IV and 1 for level V.

Below is the confusion matrix for y_pred of the trained model



Classification report of the model for each target class.

True label

	precision	recall	f1-score	support
1	1.00	0.17	0.29	6
2	0.30	0.29	0.29	21
3	0.26	0.40	0.31	20
4	0.52	0.53	0.52	30
5	1.00	0.14	0.25	7
accuracy			0.38	84
macro avg	0.61	0.31	0.33	84
weighted avg	0.48	0.38	0.38	84

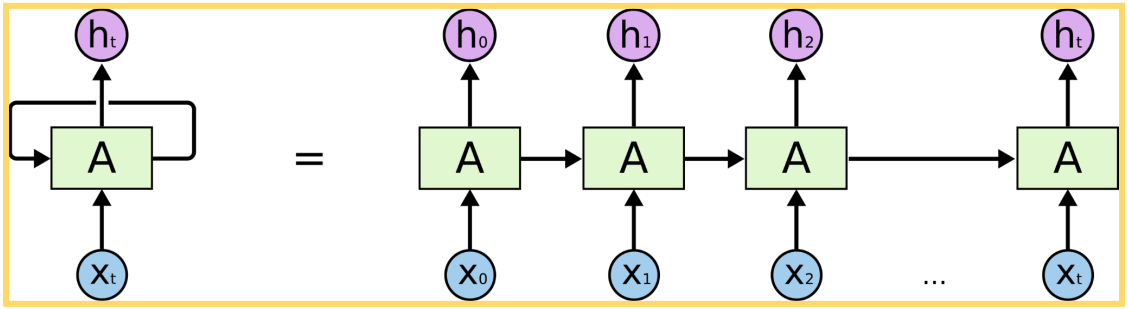
The best score after training the model

```
{'Modelname': 'Neural Network',
 'CV Score': '',
 'Train Score': 0.8592814371257484,
 'Test Score': 0.38095238095238093,
 'Precision Score': 0.48,
 'Recall Score': 0.38,
 'F1 Score': 0.38}
```

The test accuracy score obtained is 38.1% with precision score 0.48 and F1 score as 0.38. We can see that the train score is 85.92%.

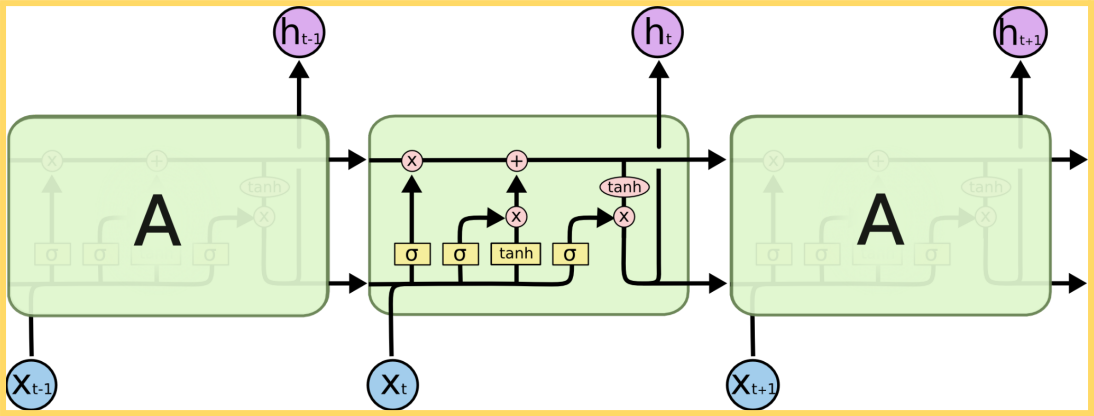
7.3 RNN(Recurrent neural network)

A recurrent neural network (RNN), unlike a feedforward neural network, is a variant of a recursive artificial neural network in which connections between neurons make a directed cycle. It means that output depends not only on the present inputs but also on the previous step's neuron state. This memory lets users solve NLP problems like connected handwriting recognition or speech recognition. In a paper, Natural Language Generation, Paraphrasing and Summarization of User Reviews with Recurrent Neural Networks, authors demonstrate a recurrent neural network (RNN) model that can generate novel sentences and document summaries



7.4 LSTM (Long Short Term Memory)

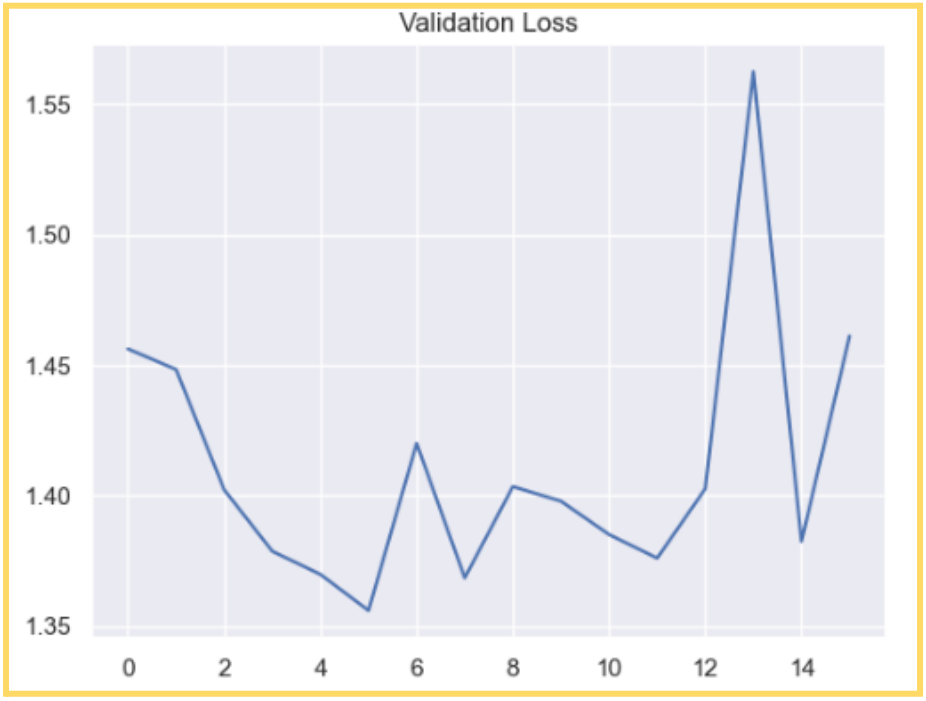
Long Short-Term Memory (LSTM) is a specific recurrent neural network (RNN) architecture that was designed to model temporal sequences and their long-range dependencies more accurately than conventional RNNs. LSTM does not use activation function within its recurrent components, the stored values are not modified, and the gradient does not tend to vanish during training. Usually, LSTM units are implemented in “blocks” with several units. These blocks have three or four “gates” (for example, input gate, forget gate, output gate) that control information flow drawing on the logistic function.



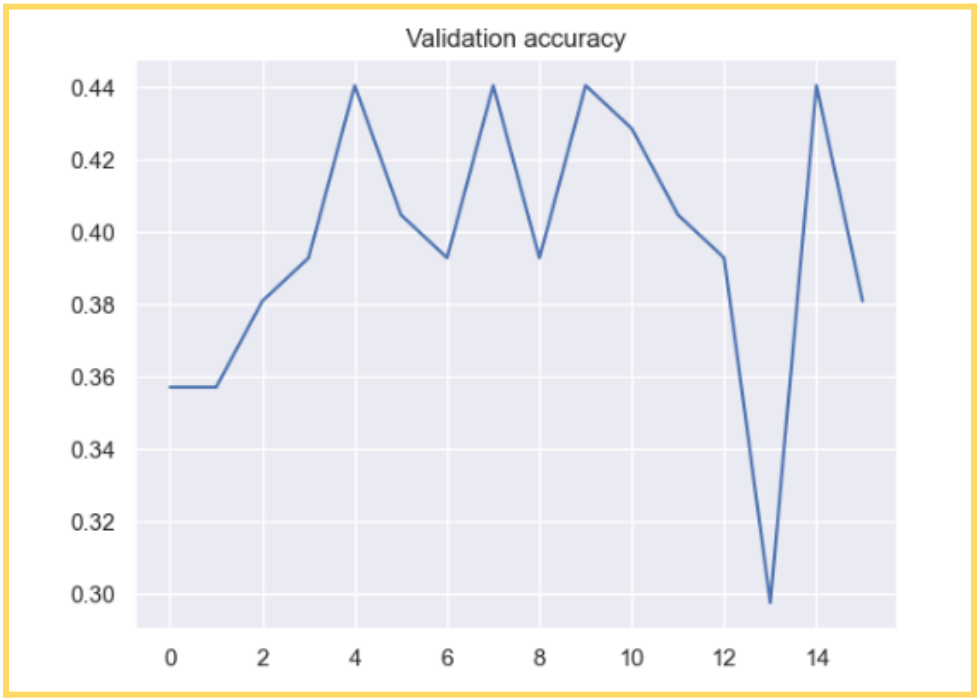
LSTM Model Summary

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_5 (Embedding)      (None, 99, 200)           544000

 bidirectional_1 (Bidirectio  (None, 99, 200)           240800
 nal)

 global_max_pooling1d_1 (Glo  (None, 200)               0
 balMaxPooling1D)

 dropout_13 (Dropout)         (None, 200)               0

 dense_17 (Dense)             (None, 5)                 1005

=================================================================
Total params: 785,805
Trainable params: 241,805
Non-trainable params: 544,000
_________________________________________________________________
```

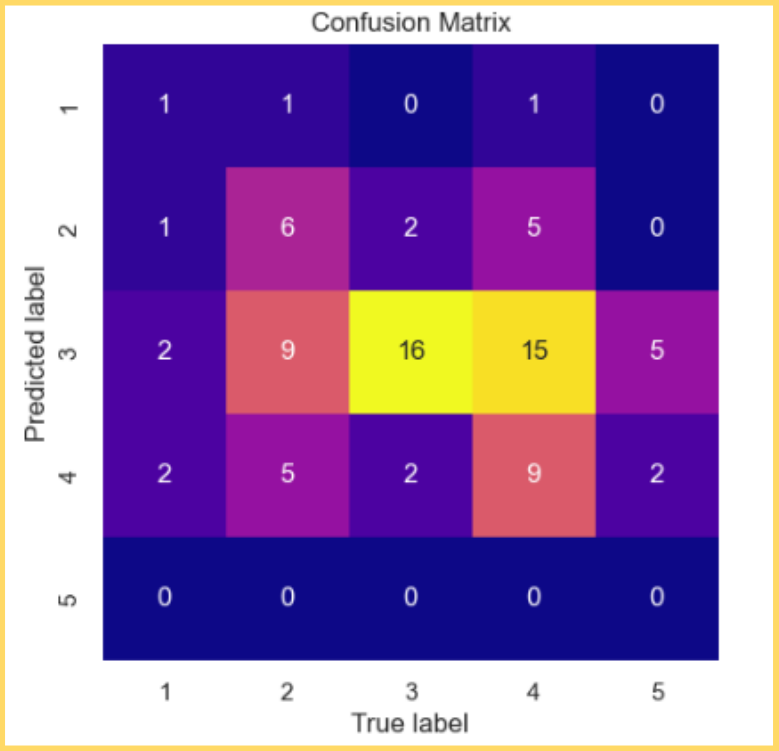
Graph showing the history of the model for Validation data loss score



Graph showing the history of the model for Validation data accuracy score



We can see from the below confusion matrix that the target class is predicted correctly for potential accident level I once, 6 for level II, 16 for level III, 9 for level IV and 0 for level V.



	precision	recall	f1-score	support
1	0.33	0.17	0.22	6
2	0.43	0.29	0.34	21
3	0.34	0.80	0.48	20
4	0.45	0.30	0.36	30
5	0.00	0.00	0.00	7
accuracy			0.38	84
macro avg	0.31	0.31	0.28	84
weighted avg	0.37	0.38	0.34	84

Best score for LSTM:

```
{'Modelname': 'LSTM Network',
 'CV Score': '',
 'Train Score': 0.7904191616766467,
 'Test Score': 0.38095238095238093,
 'Precision Score': 0.46,
 'Recall Score': 0.38,
 'F1 Score': 0.34}
```

The test accuracy score obtained is 34.5% with precision score 0.45 and F1 score as 0.33. We can see that the train score is 76.64%.

Ideally, the neural network model must provide best accuracy for the given dataset in order to predict the target based on textual data. However we can observe that both sequential models with drop out and batch normalization and applying LSTM did not increase the accuracy of the model. This is due to the number of records in the dataset. To train an NLP model we need a huge dataset which will be able to predict the target well after analyzing each text in the description(textual data).

Hence we can see from the below table that the top performing model for the given dataset with 425 records are Machine learning classifiers after data preprocessing and sampling the data.

```
#Consolidated score board of models which is trained with all the columns [] and predicted potential accident level is
Finalmodels=pd.concat([Mlmodel2,smotemodelscore], ignore_index=True)
Finalmodels.sort_values(['Test Score','Precision Score'],ascending = [False, False])
```

	Modelname	CV Score	Train Score	Test Score	Precision Score	Recall Score	F1 Score
32	Smote - NB	75.72	0.960000	0.780000	0.78	0.78	0.78
36	Smote - RF	71.94	1.000000	0.750000	0.76	0.75	0.75
33	Smote - SVM	76.62	1.000000	0.740000	0.77	0.74	0.74
30	Smote - LR	69.79	0.920000	0.680000	0.69	0.68	0.68
31	Smote - KNN	56.29	0.670000	0.610000	0.73	0.61	0.53
34	Smote - CART	59.0	1.000000	0.610000	0.59	0.61	0.59
35	Smote - AB	37.42	0.540000	0.500000	0.48	0.50	0.45
6	Bag of Words - RF	44.61	1.000000	0.490000	0.53	0.49	0.44
3	Bag of Words - SVM	42.53	0.950000	0.400000	0.52	0.40	0.40
0	Bag of Words - LR	45.5	1.000000	0.400000	0.42	0.40	0.41
2	Bag of Words - NB	44.3	1.000000	0.390000	0.44	0.39	0.37
28	Neural Network		0.859281	0.380952	0.48	0.38	0.38
29	LSTM Network		0.790419	0.380952	0.46	0.38	0.34
13	TFIDF - RF	33.23	1.000000	0.380000	0.42	0.38	0.32
5	Bag of Words - AB	34.74	0.460000	0.380000	0.40	0.38	0.34
27	Skip_1Gram - RF	39.8	1.000000	0.370000	0.40	0.37	0.34
20	CBOW - RF	33.51	1.000000	0.370000	0.36	0.37	0.31

We can see that the best performing model is SVM with the highest Cross validation score of 76.62% and 74% test score.

```
#best Model
Finalmodels.iloc[33]
```

```
Modelname          Smote - SVM
CV Score                 76.62
Train Score                1.0
Test Score                0.74
Precision Score           0.77
Recall Score              0.74
F1 Score                  0.74
Name: 33, dtype: object
```

7.5 Fine tuning the SVM model using hyper parameters and GridSearchCV

GridSearchCV implements a "fit" and a "score" method. It also implements "score_samples", "predict", "predict_proba", "decision_function", "transform" and "inverse_transform" if they are implemented in the estimator used. The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

HyperParameters used are

1) **C**: is the l2 regularization parameter. The value of C is inversely proportional to the strength of the regularization. Ranging from 0.1 to 0.5
2) **Kernel**: function transforms the training dataset into higher dimensions to make it linearly separable. Parameters used are 'rbf' and 'sigmoid'
    a) Gaussian Radial Basis Function (RBF): It is used when the data is non-linear.
    b) Sigmoid Kernel: It is usually used in neural networks.
3) **CV**= kfold takes the StratifiedKFold we defined. We have defined 5 folds.

The best score obtained after hyper tuning the Support Vector Machine classifier is 0.71 with best parameters C-0.5 and rbf kernel.

```
#Training the SVm with best parameters

hyperparameters={"C":[0.1, 0.5],"kernel":[ 'rbf' , 'sigmoid' ]}
gcv = GridSearchCV(SVC(),hyperparameters,cv=5,verbose=3,n_jobs=-1,)
best_modeltune = gcv.fit(X_trains, y_trains)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
```

```
message = ('SVC()',best_modeltune.best_score_, best_modeltune.best_params_)
print("%s -Best: %f using %s" % (message))
```

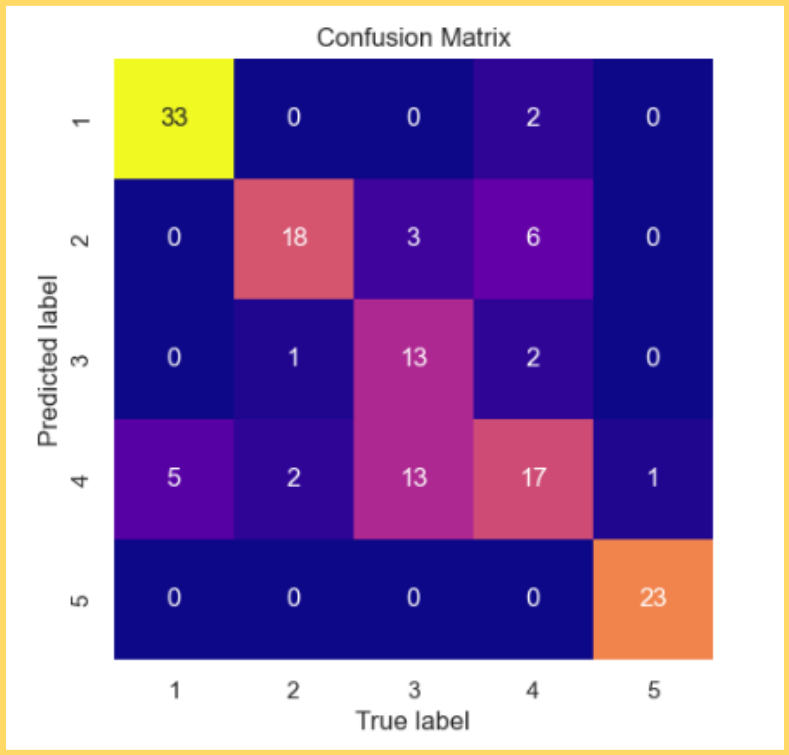
```
SVC() -Best: 0.712130 using {'C': 0.5, 'kernel': 'rbf'}
```

Scoring the SVC model with best parameters from above

```
bestmodel=SVC(C= 0.5,kernel= 'rbf')
bestmodel.fit(X_trains, y_trains)
Y_true, y_pred = y_tests, bestmodel.predict(X_tests) #prediction with test data
Y_traintrue, ytrain_pred = y_trains, bestmodel.predict(X_trains) #prediction with train data
#Training and testing scores
print("Test Accuracy ",accuracy_score(Y_true, y_pred))
print("Train Accuracy ",accuracy_score(Y_traintrue, ytrain_pred))
print("precision_score ",precision_score(Y_true, y_pred, average='weighted',zero_division=1).round(2))
print("Recall ",recall_score(Y_true, y_pred, average='weighted',zero_division=1).round(2))
print("F1 ",f1_score(Y_true, y_pred, average='weighted',zero_division=1).round(2))
```

```
Test Accuracy  0.7482014388489209
Train Accuracy  0.9514388489208633
precision_score  0.79
Recall  0.75
F1  0.75
```

We can observe that Train accuracy is 95.14% and Test accuracy is 74.82% which is better than the Neural network model and the Precision score is 0.79 and F1 score is 0.75. We can also observe that the Model is able to predict different target classes well as shown in the Confusion Matrix below. The target class is predicted correctly for potential accident level I for 33 records , 18 for level II, 13 for level III, 17 for level IV and 23 for level V with very few misclassifications.



7.8 Pickling model:

Pickle in Python is primarily used in serializing and deserializing a Python object structure. In other words, it's the process of converting a Python object into a byte stream to store it in a file/database, maintain program state across sessions, or transport data over the network

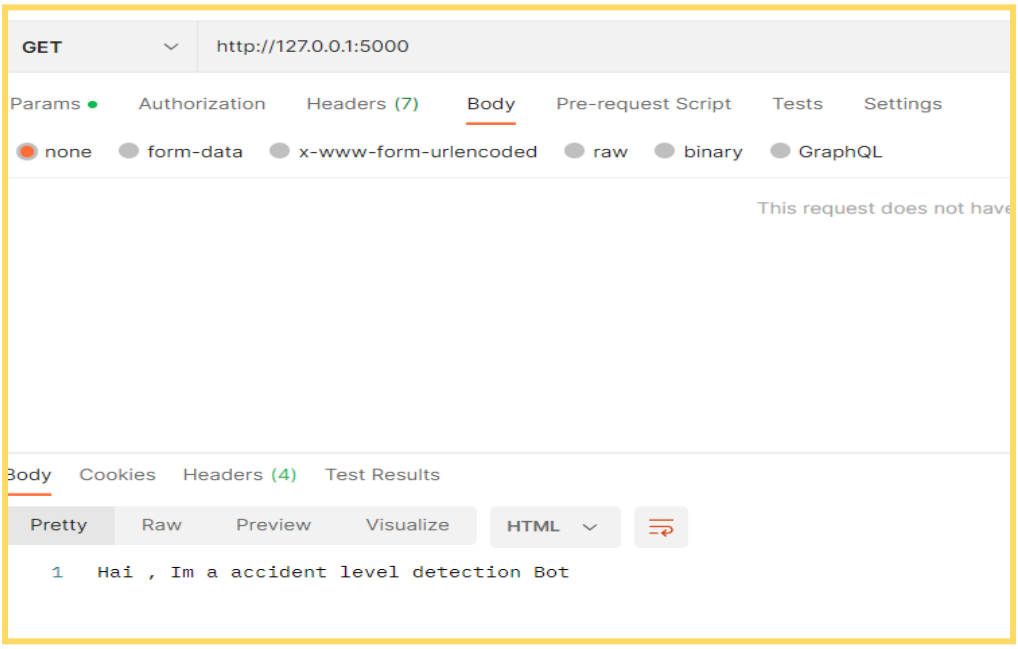
```
with open('industrysafe.pickle','wb') as pkl:
    pickle.dump(bestmodel,pkl)
```

7.9 Chatbot - FLASK API

Flask is a popular micro framework for building web applications. Since it is a micro-framework, it is very easy to use and lacks most of the advanced functionality which is found in a full-fledged framework. Therefore, building a REST API in Flask is very simple.

```
from flask import Flask,render_template,url_for,request,jsonify
import pickle
app = Flask(__name__)

@app.route('/')
def home():
    return "Hai , Im a accident level detection Bot"

@app.route('/predict',methods=['POST','GET'])

def predict():
    def preprocess_text(text):

        #stopwords removal
        Cleandesc=  ' '.join([words for words in text.split() if words not in stop])

        #Lemmetization
        lemmatizer = WordNetLemmatizer()
        Cleandesc= ' '.join([lemmatizer.lemmatize(word) for word in Cleandesc.split(" ")])

        #Convert to lowercase
        Cleandesc= Cleandesc.lower()

        #Remove Special characters and Numbers
        pattern = r'([^a-zA-Z\s]+?)'
        Cleandesc = re.sub(pattern,"",Cleandesc)

        # remove extra white spaces
        Cleandesc = Cleandesc.strip()

        return Cleandesc



    #if request.method == 'POST':
    new_complaint =request.form['message']
    stop=set(stopwords.words('english'))
    print(new_complaint)
    Cleandesc= preprocess_text(new_complaint)
    #Load it later
    loaded_vec = TfidfVectorizer(ngram_range=(1,2), analyzer='word', min_df=5, sublinear_tf=True,vocabulary=pickle.load(open("vectors.pkl", "rb")))
    prepdesc = loaded_vec.fit_transform(np.array([Cleandesc])).toarray()
    prediction = model.predict(prepdesc)


    return jsonify({"Potential accident level":str(prediction[0])})


if __name__ == '__main__':
    modelfile='industrysafe.pickle'
    model = pickle.load(open(modelfile, 'rb'))
    app.run()
```

Testing The API from Postman



8. Implications

1. We were able to predict the accident level with a test accuracy of 74.82% and f1-score of 75%
2. There were seven duplicate values in the dataset which were dropped.
3. There are no outliers/missing values in the dataset.
4. The day, month and year from the Date column were extracted and a new feature, weekday was created.
5. Target variable – 'Accident Level' distribution is not equal (I: 309, II: 40, III: 31, IV: 30, V: 8).
6. Class imbalance issue was handled using below methods and we found out that better results were achieved with original data for this particular dataset.
    a. Resampling techniques: Oversampling minority class
    b. SMOTE: Generate synthetic samples
7. Cross validation score is 76.62% for SVM which is high when compared to NB and RF.
8. SVM performs far better than all other models with 79% precision score and 74.82% test accuracy.
9. Finally, a bidirectional LSTM model can be considered to productionalized the model and predict the accident level.
10. In this project, we observed that the main causes of accidents are mistakes in hand-operation and time-related factors.To reduce the occurrences of accidents, more stringent safety standards in hand-operation will be needed during high risk tasks.
11. We realized that the detailed information of accidents mentioned in the 'Description' is valuable information to analyze the cause.
12. With additional information such as machining data (ex. CNC, Current, Voltage) in plants, weather information, employee's personal data (ex. age, experience in the industry sector, work performance), we can identify the cause of accidents more accurately.
13. With more observations than the current small dataset, we can feed more data into Machine Learning /Neural Networks/NLP models to train, evaluate the performance of those models and to obtain better results.
14. There are quite a lot of critical risk descriptions, but with the help of SME, we can decide whether this column has outliers or not and also SME can help us in understanding the data better.
15. Developed a flask prediction API that can be consumed in Chat application

9. Insights

General:

- Majority of accidents were happening in Mining followed by the Metal industry.
- Accident level (1) is the most commonly reported accident.
- Slightly major accidents happened with contract workers when compared to employees.
- The third party workers were having higher potential accident levels.
- In the dataset more accidents were registered in Country 1 and in Local_03

- For both accident levels, the incidence of Employee is higher at low accident levels, but the incidence of Third parties seems to have higher accident levels.

To Improve:

- The Dataset is relatively small and hence the data is too small for the model.
- Description Data is not structured, some key words can be used to describe the accurate description.
- The Dataset is unbalanced, might be due to the problem type here.
- Including the non-text inputs for the LSTM model could be done.
- The words describe(Body-related,Accident-related:) The severity of the accident is not evident by saying body parts name or accident name, they can be termed as injury level, so as to improve the model.
- And major critical risks are termed as others, which is not giving more clear information on risks to mitigate.
- The time of the day of accidents is not captured, it could help to draw conclusions on finding the root cause of the accidents.

Limitations

- ➢ We have less number of observations to analyze the cause of accidents correctly and rather we should collect more observations to get better results.
- ➢ Less number of features available in the dataset.
- ➢ Lack of access to quality data. A greater data observation can help.

Where does our model fall short in the real world?

- Once we deploy the finalized model in Production, we might get less f1-score as compared to productionalized model results.
- Since we are predicting the accident level, we need to be 100% sure or at least close to 100% so that we can prevent a lot of accidents in industry.

10. Closing Reflections

- ❖ **What did we learn from the process?**
  - ➢ How to work on a Data Science project end-to-end.
  - ➢ How to handle class imbalance data set.
  - ➢ It is an important industrial use case problem that requires a solution. Many organizations can have data, but to derive insights using the NLP techniques and also the other ML models will go a long way in solving such problems.
  - ➢ The learning also provided us how to deploy and how to utilize them in the real world with real use cases.

❖ **What should I do differently next time?**

- ➢ Perhaps we will explore more feature engineering and feature selection techniques.
- ➢ Transfer learning of bidirectional LSTM by removing the output layer and saving the last layer result and passing that learning to the best performing SVM classifier to earn more deep and best learning of the model.
- ➢ Developing a chat application for user interface to predict the accident level.
- ➢ collecting more samples for the industry safety domain because we had only 425 samples in raw data before sampling.
- ➢ Also we are aware that Chat GPT has better intelligence to respond to user queries and data processing . we must try to integrate our application with chat GPT for better results.

11. References:

- ☐ https://deepai.org/machine-learning-glossary-and-terms/neural-network
- ☐ https://colah.github.io/posts/2015-08-Understanding-LSTMs/
- ☐ https://medium.com
- ☐ https://towardsdatascience.com
- ☐ https://www.tensorflow.org/api_docs/python/tf/keras/

Thank you