AIML Capstone Project Report

CHATBOT INTERFACE

Group 3

Members:

Manjunath R

Shipra Jain

Tharoh Krishnamurti

THE OTHER TWO MEMBERS IN THE TEAM HAVE ZERO CONTRIBUTION TO THIS ENTIRE PROJECT (ALL THE MILESTONES)

Table of Contents

MILESTONE 1 3

Introduction (Industry and Problem Statement) 3

Objective of the Project 3

Dataset Description (Including EDA) 4

Univariate Analysis 4

Multivariate Analysis 9

Bivariate Analysis 11

Proposed Approach and Theory (Data Cleaning, Model Selection for Machine Learning Model) 15

Performance of the Model (Stats of Performance and Configuration Comparisons mostly by Grid Search Mechanism) 21

MILESTONE 2 27

Summary of problem statement, data and findings 27

Overview of the final process 28

Step-by-step walk through the solution 31

Model evaluation 40

Comparison to benchmark 42

Visualizations 44

Implications 48

Limitations 49

Closing Reflections 50

MILESTONE 1

Introduction (Industry and Problem Statement)

The project focuses on industrial safety, specifically developing a chatbot utilizing Natural Language Processing (NLP). It addresses the critical issue of understanding and preventing accidents and injuries in industrial environments. The project's relevance is underscored by the ongoing occurrences of such incidents, sometimes leading to fatalities, in various industries globally. This NLP-based solution aims to enhance safety protocols and awareness in industrial settings.

Objective of the Project

The primary objective is to design a machine learning (ML) and deep learning (DL) based chatbot utility. This tool is intended to assist professionals in identifying and highlighting safety risks based on incident descriptions. By analyzing accident reports and related data, the chatbot will provide insights into potential hazards and preventive measures, thereby contributing to improved safety standards in industrial workplaces.

Dataset Description (Including EDA)

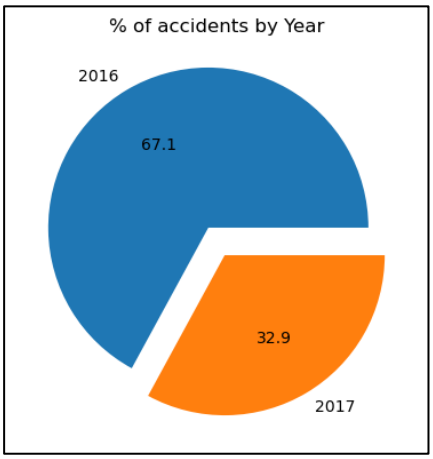
The dataset comprises accident records from 12 different plants across three countries. Key features include timestamps of accidents, anonymized country and city information, industry sectors, accident severity levels, potential accident levels, gender of the individuals involved, whether they are employees or third parties, critical risks involved, and detailed accident descriptions. This comprehensive dataset forms the basis for exploratory data analysis (EDA) to understand patterns and insights related to industrial accidents.

Columns 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 of 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.

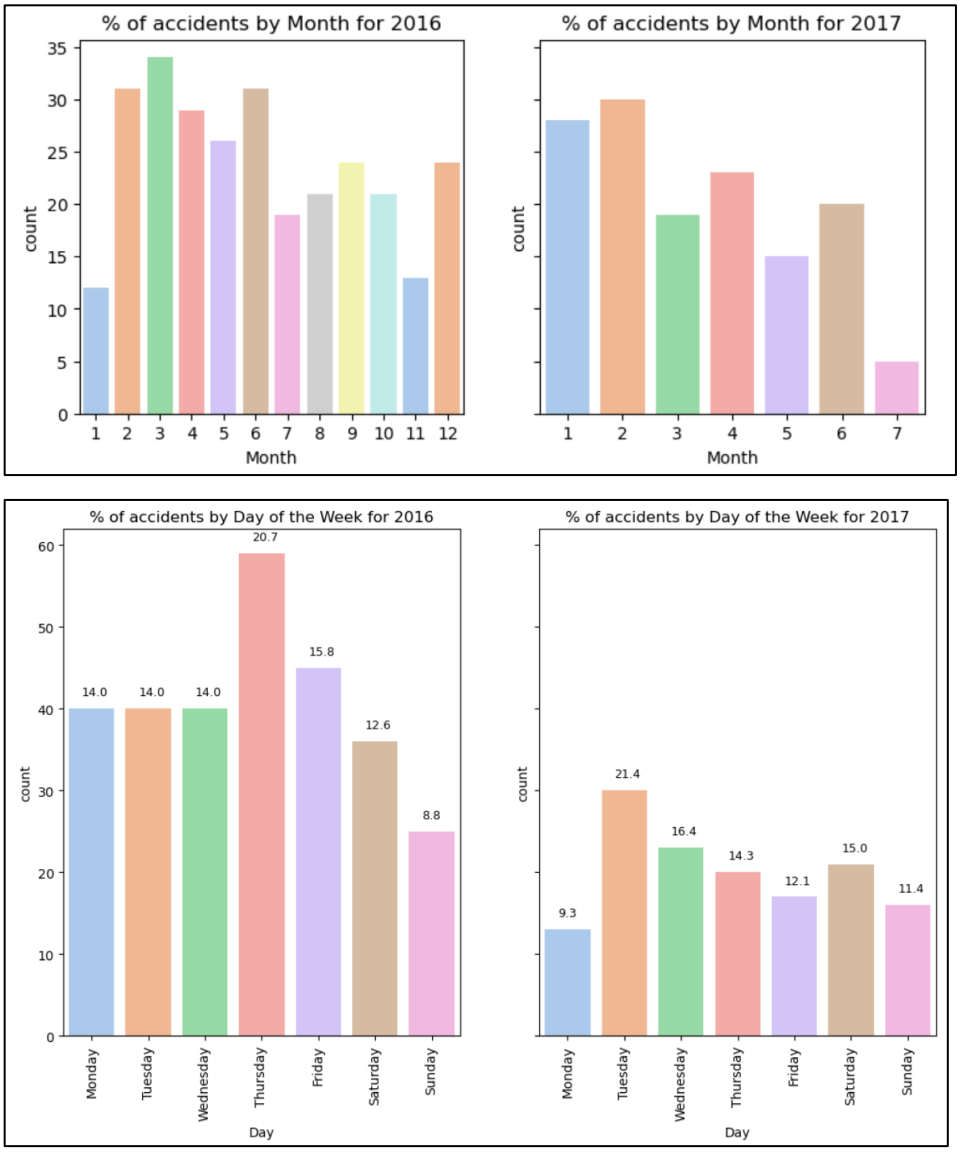
Univariate Analysis

1. **Analysis by Year:**



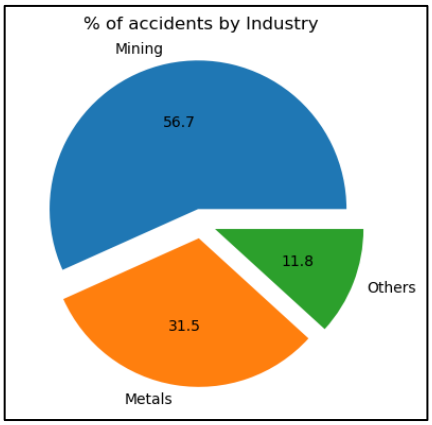
We utilized a pie chart to represent the percentage of accidents by year. It is important to note that the data is only available until 9th July 2017, which may not provide conclusive evidence to determine if accidents increased or decreased in 2017. A comparison of the first half of both years shows a reduction of about 20% in accidents in 2017.

2. **Analysis by Month and Day:**



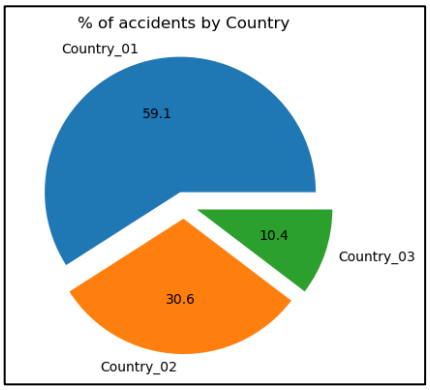
Count plots for 2016 and 2017 display the distribution of accidents by month and day. The data suggests that for each month from January to June, the number of accidents reduced in 2017. Approximately 70-80% of accidents occurred on weekdays, possibly influenced by to work-related travel.

3. **Industry Sector and Country Analysis:**



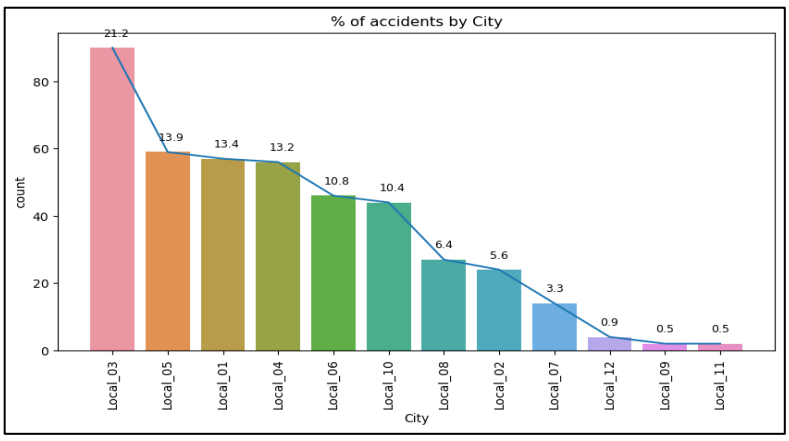
The Mining industry experiences the highest number of accidents, followed by the Metals industry.

4. **Country Analysis:**



Country_01 has the highest number of accidents, contributing to nearly 90% of the total accidents.

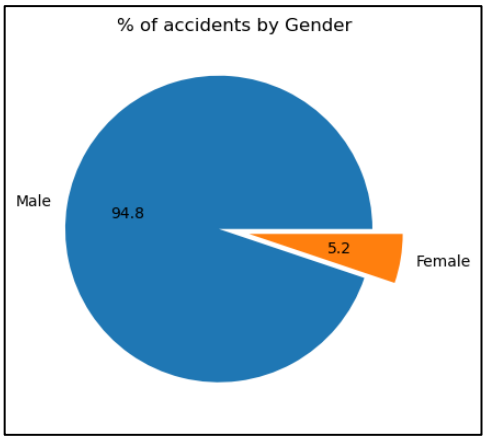
5. **City Analysis:**



A count plot and line plot together indicate that about 61% of the accidents occur in

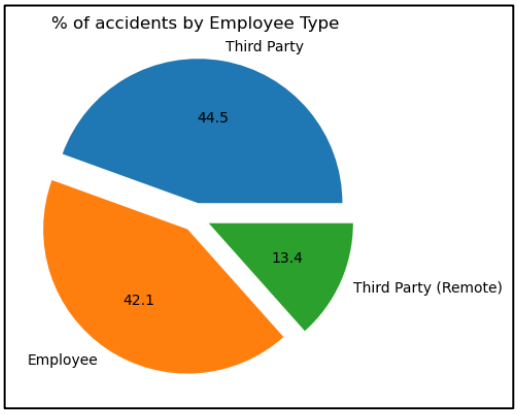
plants located in Local_01, Local_03, Local_04, and Local_05. Local_03 alone accounts for 21% of the accidents. The bottom most 6 cities - Local_02, Local_07, Local_08, Local_09, Local_11, Local_12 cities witnessed only 17% of the total accidents.

6. **Gender Analysis:**



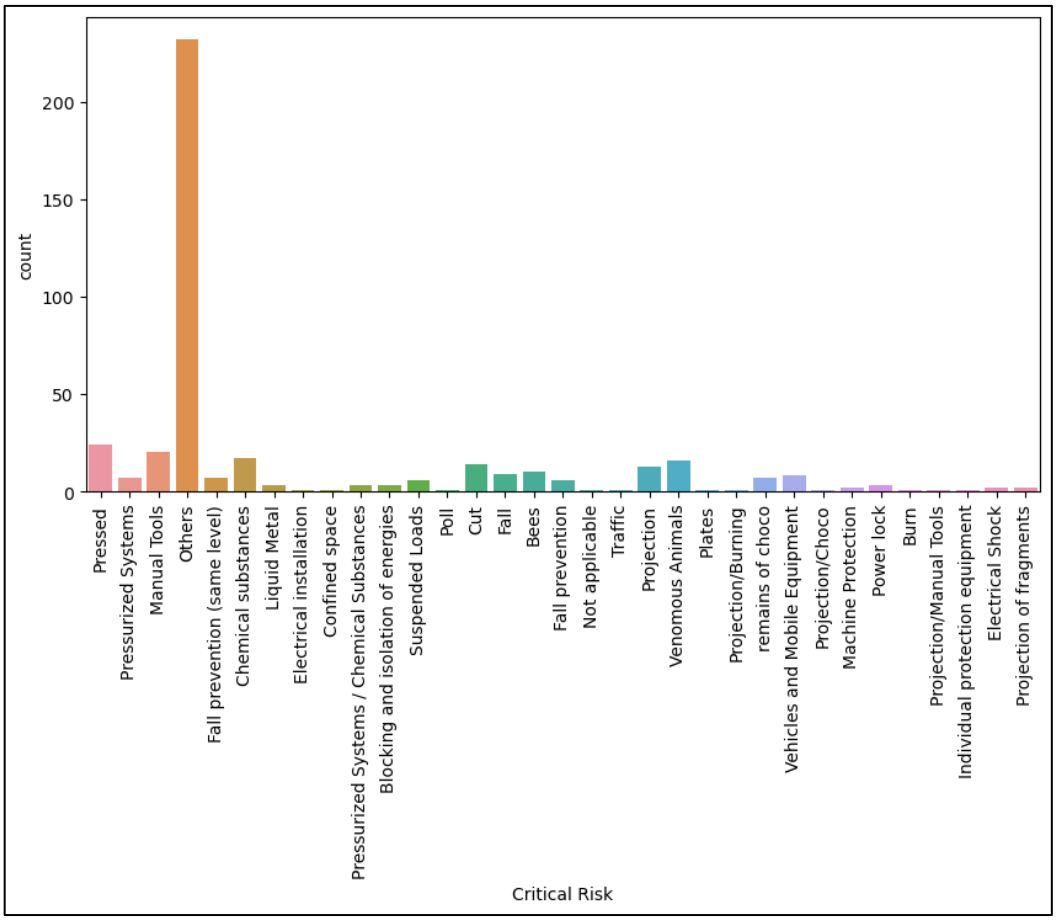
Only about 5% of the accidents involve females, potentially reflecting the gender distribution in industrial jobs.

7. **Employee Type Analysis:**



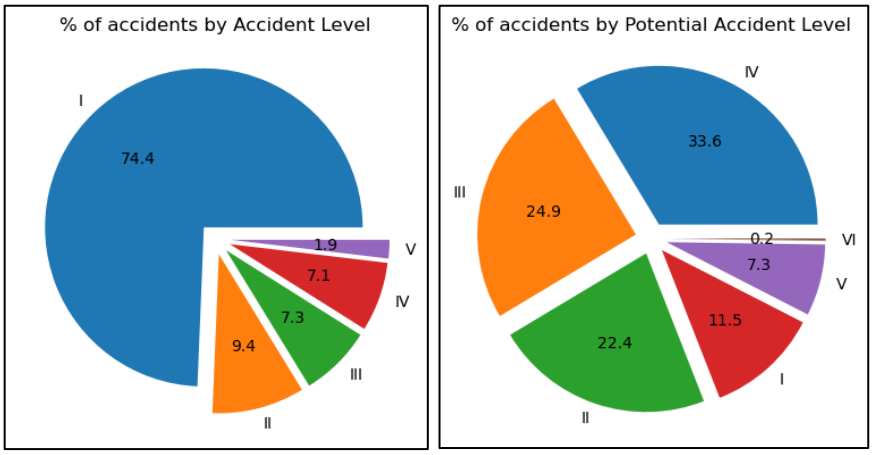
Third Party Remote workers encounter the fewest accidents, with overall 58% of the accidents involving Third Party employees, which could potentially be the case as they'd largely be working off-site.

8. **Critical Risk Analysis:**



The 'Others' category encompasses about 50% of the risks in the dataset. Common risks include Pressed, Manual Tools, Chemical Substances, Venomous Animals, Projection, and Cut.

9. **Accident Level & Potential Accident Level Analyses:**

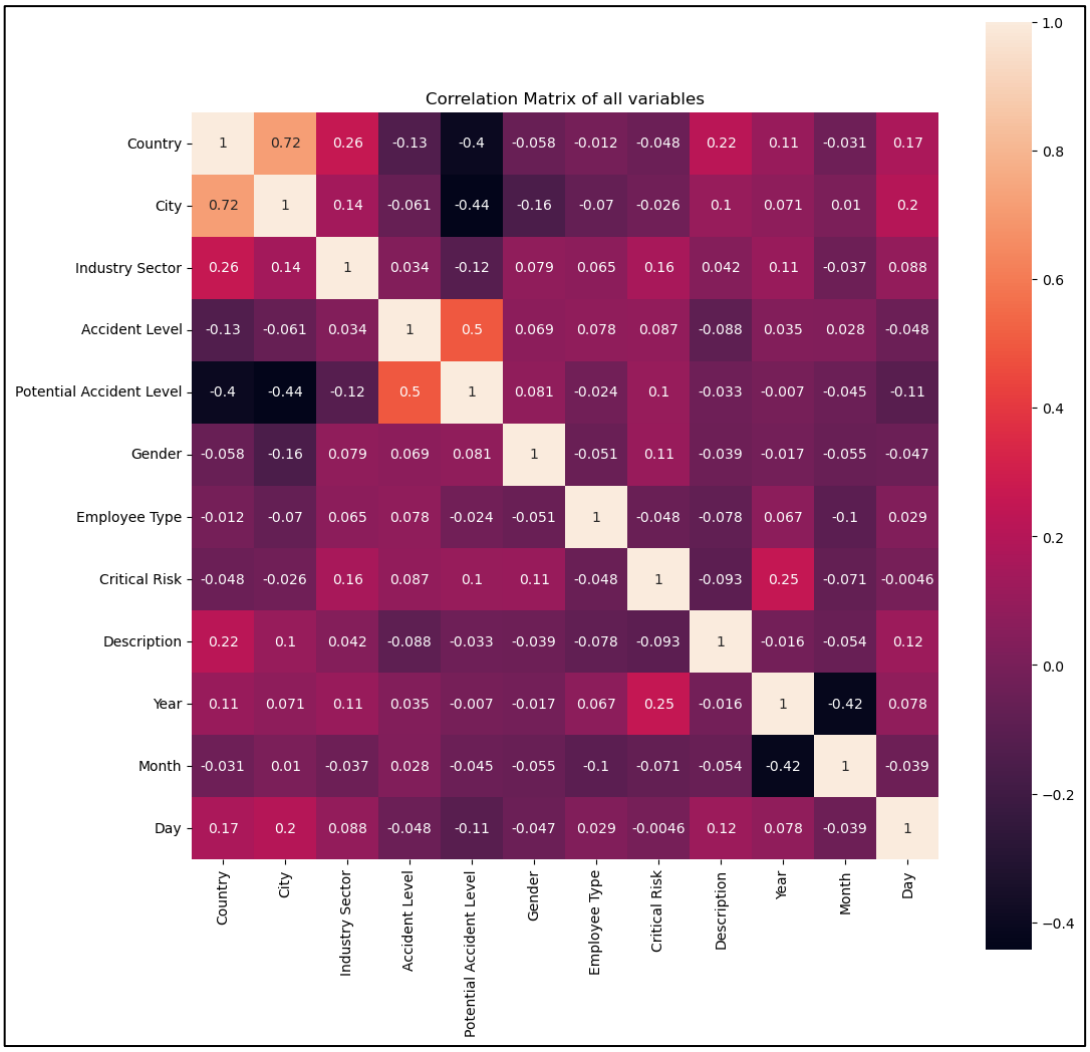


The majority of accidents are classified as Level I, occurring due to minor incidents. High risk or severe accidents (Level V) account for only 2%.

Approximately 81% of accidents could potentially be moderately severe (Levels II, III, and IV), and only 7.5% could have been the most severe (Level V). To simplify the comparison between Accident Level and Potential Accident Level, we adjusted Level VI to Level V in the data, as Level VI has only 1 accident recorded.

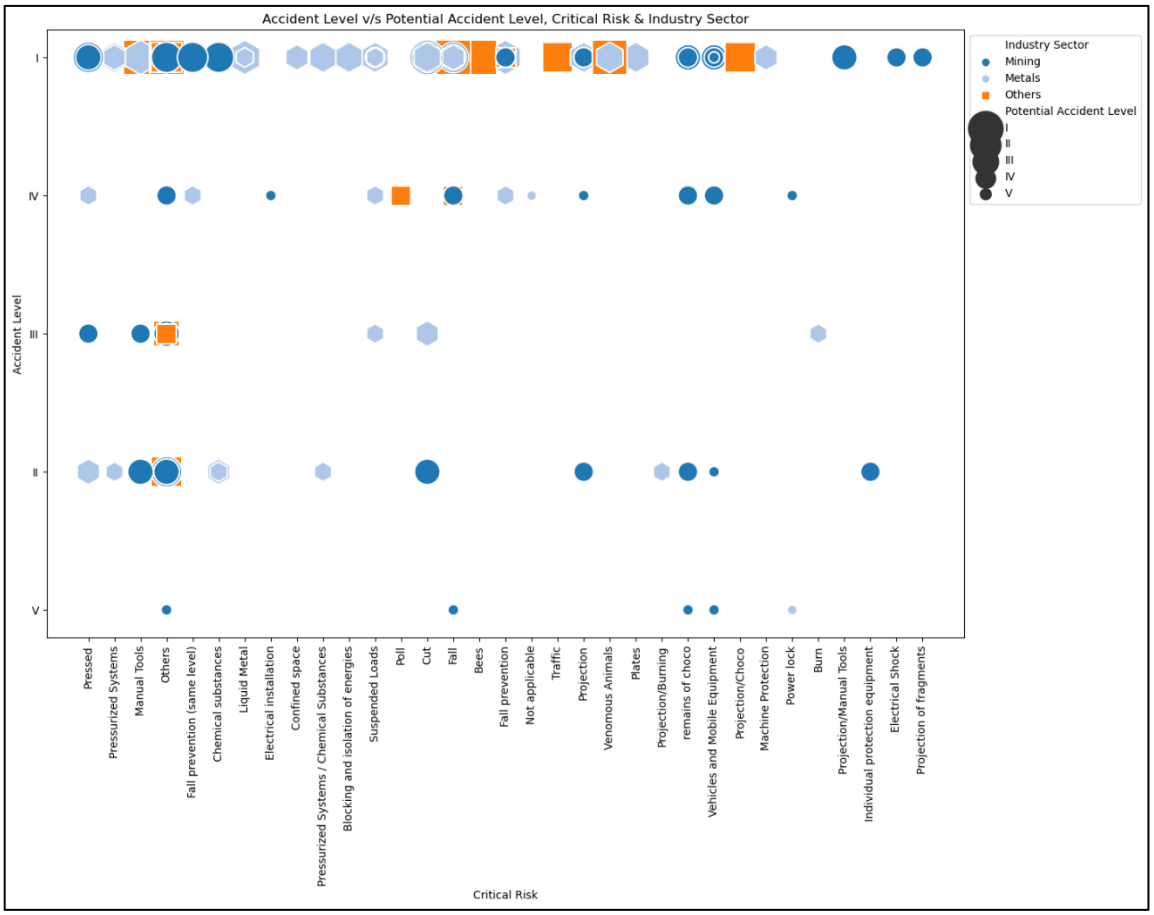
Multivariate Analysis

10. **Correlation Analysis:**



A heatmap of the correlation matrix shows the highest correlation between City and Country. Accident Level and Potential Accident Level share only a 50% correlation. Other variables exhibit low to moderate correlation.

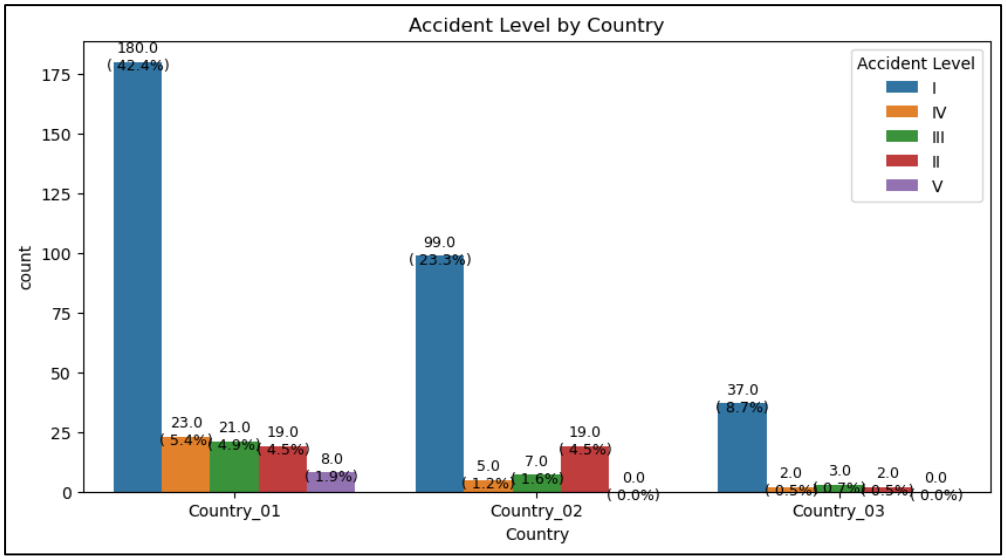
11. **Scatter Plot Analysis:**

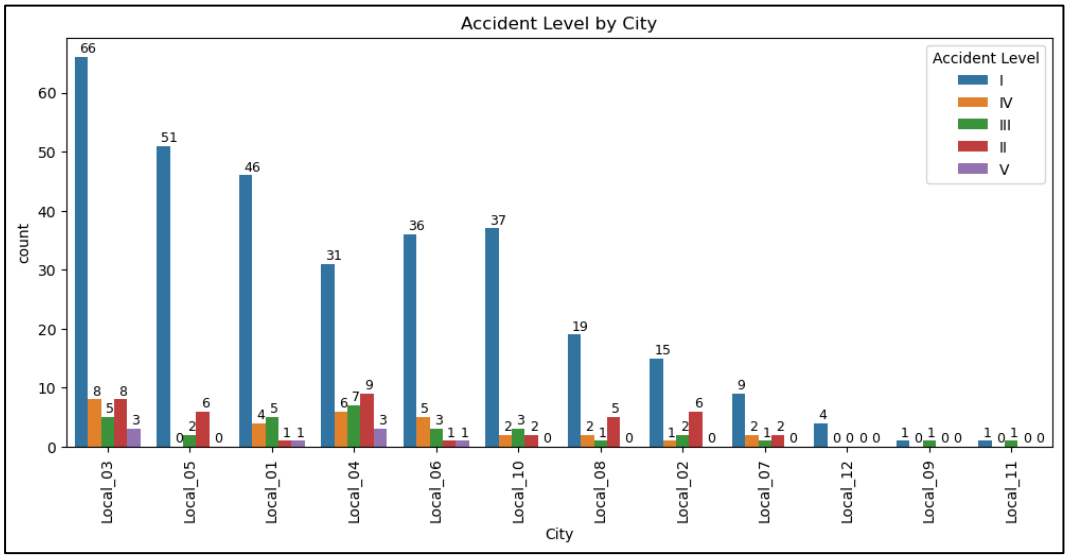


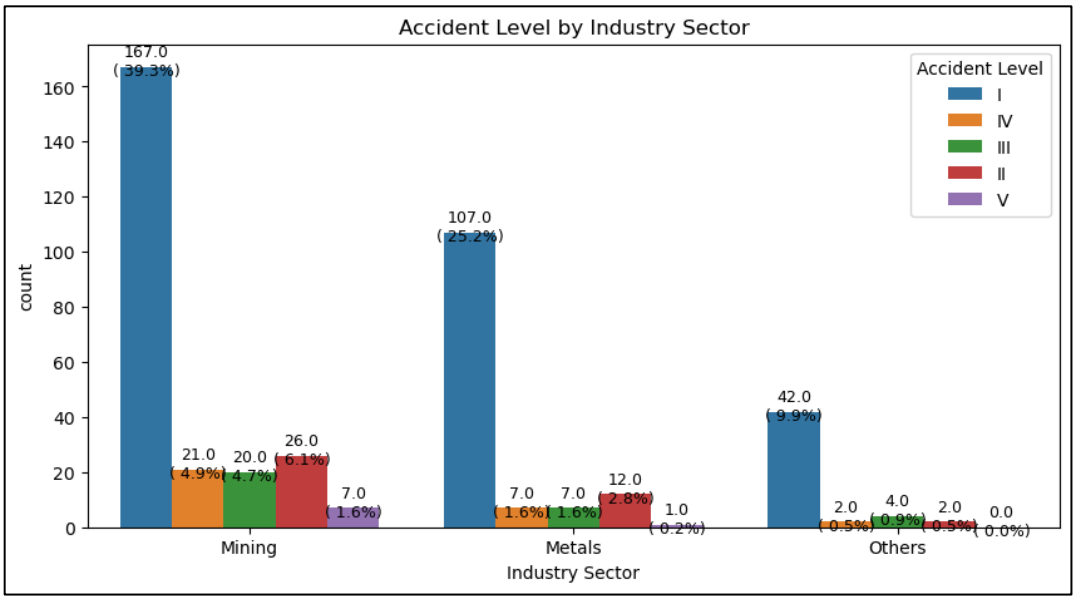
A scatter plot comparing Accident Level, Potential Accident Level, Critical Risk, and Industry Sector reveals that most accidents are of the least severity (Level I), of which many could have been potentially more severe (Levels II & III). The most severe accidents (Level V) are predominantly in the Mining industry, except for 'Power Lock' in the metals industry. The following risks are only seen in Metals industry: Pressurized Systems, Liquid Metal, Confined Space, Chemicals Substances, Blocking and isolation of engines, Suspended Loads, Burn, Machine Protection, with majority of them observed in Level I accidents.

Bivariate Analysis

12. Accident Level by Country, City & Industry Sector

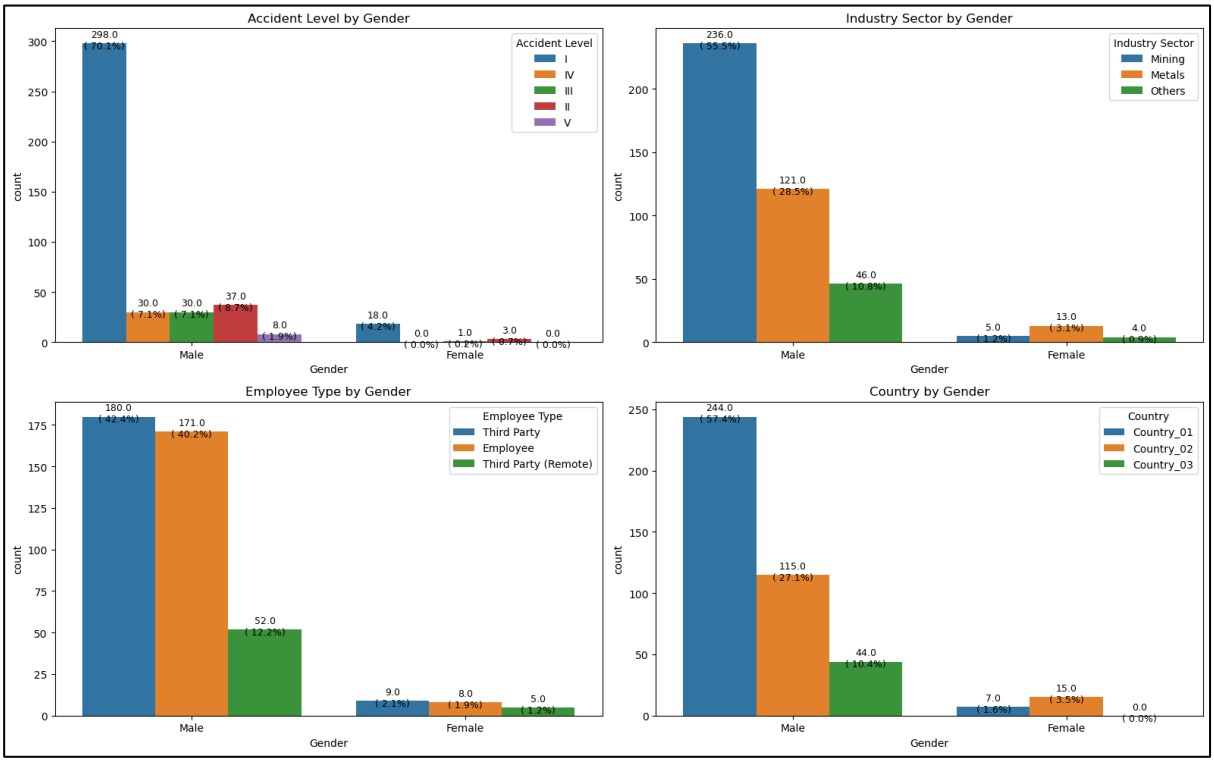






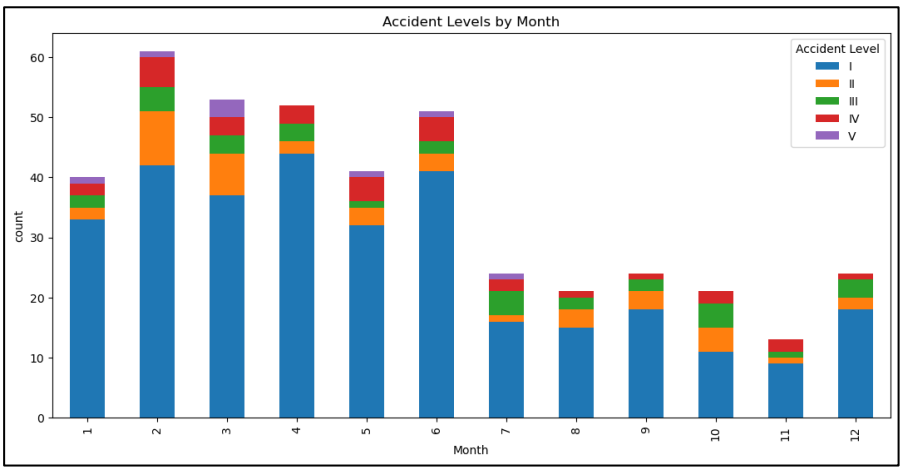
Count plots by Country, City, and Industry Sector indicate trends in accident levels across different categories. Country_01 has the majority of accidents, primarily of lower severity (Level I). Local_9, Local_11, Local_12 cities have only 2-4 accidents, could be due to low population or low staff in these plants, but we do not have any data to confirm that. Although country_02 has maximum number of plants but witnesses 2nd highest number of accidents, majority of which being least severity accidents (Level I). Plants in Local_01, Local_03, Local_04, & Local_06 cities only have faced the highest severity accidents (Level V). The risks associated could be Fall, Vehicles & Mobile Equipment, Remains of Choco & Power Lock and some are categorized under 'Others'.

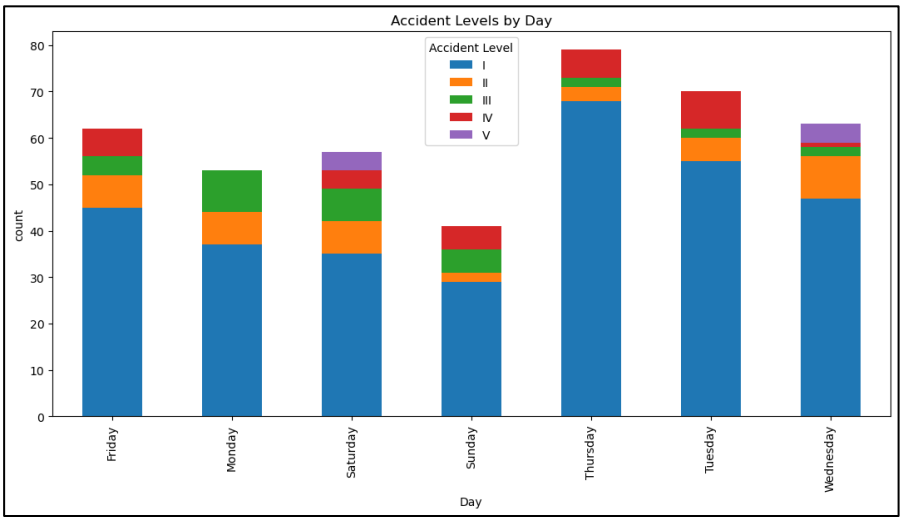
13. Different factors by Gender:



There are no females who suffered from an accident in Country_03, but even in other 2 countries together, only 6% of the accidents have females involved. About 60% of the females met with accident in Metals industry, and 23% in Mining industry, while the rest in 'Others'. 87% of the men, with almost equal distribution, are either direct Employees or Third Party employees.

14. **Accident Levels by Months & Days:**





Majority of potentially least severe accidents (level I) are seen in first half of the year. But these numbers don't include data post 9th July 2017, hence this does not provide a full view. Majority of moderate severity accidents (levels II, III, IV) are seen in February & March. Monday has witnessed only Level I, II & II accidents, while Level IV accidents have mostly occured on Tuesday, Thursday, Friday & Sunday.

Final Observations:

- The most severe accidents occur primarily in March, particularly on Wednesdays and Saturdays.
- The analysis highlights the concentration of moderate to high severity accidents in specific cities and industries, with various industry & non-industry specific risks involved.
- Gender analysis shows a significant difference in the distribution of accidents, with a higher occurrence among men.

Proposed Approach and Theory (Data Cleaning, Model Selection for Machine Learning Model)

The approach involves several stages, starting with data cleansing to ensure quality and reliability. This is followed by preprocessing techniques specific to NLP, preparing the data for analysis. The subsequent phase involves designing and testing basic machine learning classifiers, moving towards more complex neural network classifiers, including RNN or LSTM models. The selection of the most suitable model is based on their performance, aiming for the most effective tool for the chatbot's functionality.

In the "Data Preprocessing (NLP Preprocessing Techniques)" section of the analysis, we performed several steps to process and analyse the 'Description' column of the dataset. These steps are crucial for understanding the causes of accidents detailed in the descriptions. Here's a breakdown of the steps taken:

Data Copying and Initial Display:

Created a copy of the dataset and displayed the initial few entries of the 'Description' column.

Checking for Non-ASCII Characters:

Implemented a function to check if there are any characters beyond the ASCII character set in the descriptions. The outcome indicated no such characters were present.

Checking for HTML Tags and URLs:

Implemented a function to check for the presence of HTML tags or URLs in the text. The result showed no HTML tags or URLs.

Text Preprocessing:

- Lowercased all text to maintain uniformity.
- Handled accented characters and Unicode using Unicode normalization.
- Retained only alphanumeric characters, removing any special characters.
- Removed English stop words to focus on relevant words.
- Applied lemmatization to reduce words to their base or dictionary form.

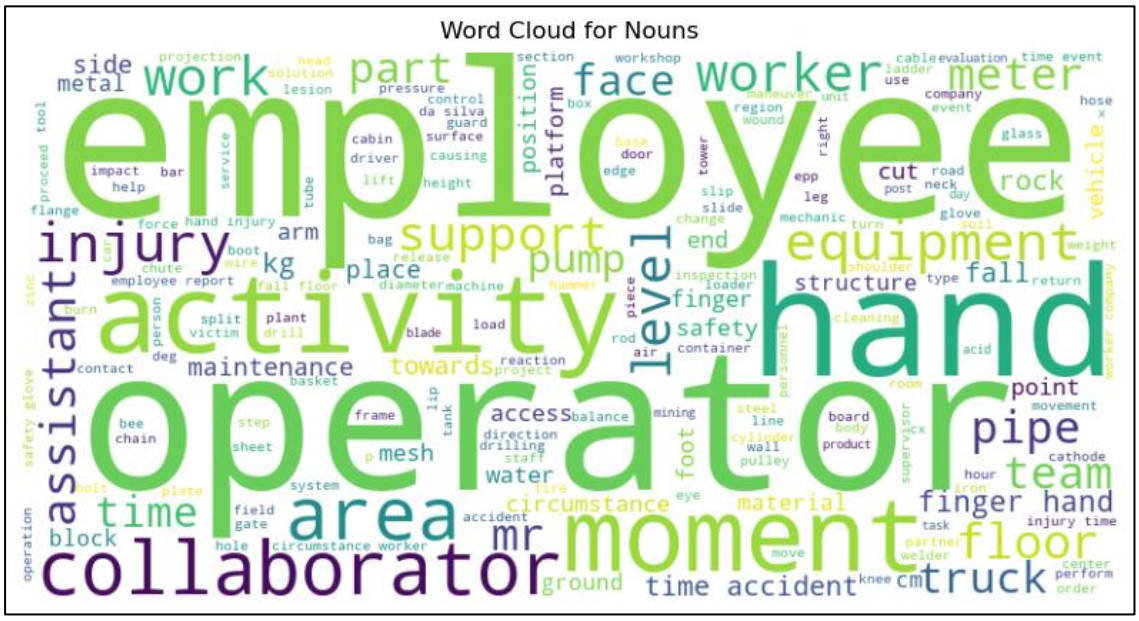
Word Cloud Generation:

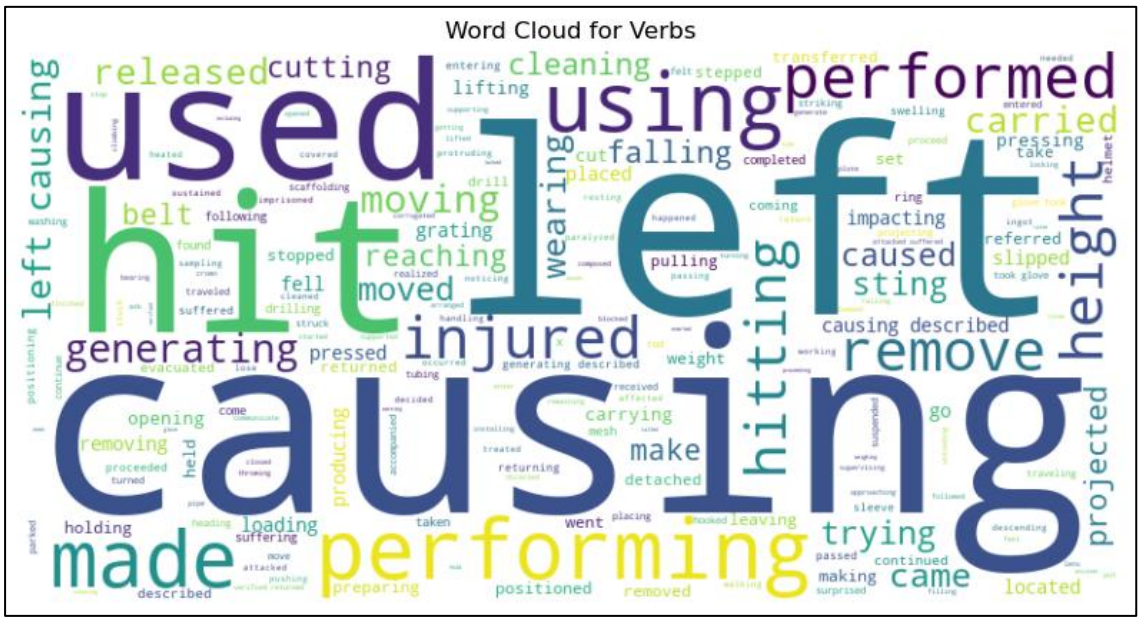
Generated a Word Cloud from the pre-processed descriptions to visually represent the most common words in the accident descriptions. This helps in quickly identifying key themes or frequent terms.

POS (Part-of-Speech) Tagging:

Performed POS tagging on the descriptions to categorize words into their respective parts of speech.

Extracted and generated separate Word Clouds for nouns and verbs. This helped in identifying common actions (verbs) and objects or subjects (nouns) associated with the accidents.





Observations:

- Noted that there were many action-related words (verbs) such as 'falling', 'cutting', 'drilling', which indicate the nature of the accidents.
- Identified various nouns like 'truck', 'pipe', 'bee', which point towards the objects involved in the accidents.
- This comprehensive preprocessing and analysis provides a clearer understanding of the causes and nature of the accidents in the dataset, which is essential for developing strategies to mitigate such incidents in the future.
-

Further Steps were taken to prepare the data for NLP model building:

Tokenization Using SpaCy:

- Loaded the English language model from SpaCy (en_core_web_sm).
- Converted the 'Description' column to string type.
- Applied SpaCy's tokenization to the 'Description' column, creating a new column 'Tokenized Text' that contains the tokens for each description.

Initial DataFrame Display:

- Displayed the DataFrame showing the first few rows, including columns like 'Date', 'Country', 'City', 'Industry Sector', 'Accident Level', 'Potential Accident Level', 'Gender', 'Employee Type', 'Critical Risk', 'Description', 'Year', 'Month', 'Day', 'POS_Tags', and 'Tokenized Text'.

N-gram Generation:

- Defined a function generate_ngrams to create n-grams (uni-grams, bi-grams, and tri-grams) from the tokenized text.
- Applied this function to the 'Description' column to create three new columns: 'Uni-grams', 'Bi-grams', and 'Tri-grams', each containing the respective n-grams for each description.

Most Frequent N-grams Identification:

- Extracted the most frequent uni-grams, bi-grams, and tri-grams using Python's Counter method.
- Displayed the top 20 most frequent n-grams in each category.

Vector Encoding with TF-IDF:

- Utilized TfidfVectorizer from sklearn, setting max features to 1000 and specifying English stopwords, with n-gram range set to (2, 2) (bi-grams).
- Transformed the 'Description' column into a TF-IDF matrix and converted it into a DataFrame (tfidf_df), which includes the TF-IDF scores for the top features.

Identification of Top Features:

- Summarized the TF-IDF scores for each feature across all documents.

- Sorted and displayed the top 30 features based on their aggregated TF-IDF scores.
- The following table gives the feature bi-gram and the TF-IDF score for them:

Feature	TF-IDF Score
left hand	16.026339
right hand	13.926010
causing injury	13.242063
time accident	10.120484
employee report	8.380265
finger left	7.739658
injury described	6.511859
medical center	6.181397
left foot	5.524003
right leg	5.479867
described injury	5.202557
finger right	4.763424
hand causing	4.761339
cm cm	4.331312
generating injury	4.217899
injured person	4.186464
hit right	4.007624
fragment rock	3.973550
middle finger	3.919644
circumstance worker	3.892055
support mesh	3.778769
right foot	3.681881
injury time	3.656111
da silva	3.262092
ring finger	3.258298
causing cut	3.255090
left leg	3.049839
employee used	3.027345
accident employee	2.984388
safety glove	2.938843

Vector Encoding with Word2Vec:

- Initialization of Word2Vec Model: Leveraged the Word2Vec implementation from the gensim library. The model was configured to create vectors of size 1000 for each word, ensuring a comprehensive and detailed representation. The window size was set to 5 to consider a balanced context of surrounding words. The min_count parameter was set to 1, allowing the model to learn representations for all words, even those with a single occurrence.
- Training on Tokenized Text: Applied the Word2Vec model to a list of tokenized sentences, tokens_list, derived from the 'Description' column of our dataset. This list represented the corpus for training, where each entry corresponded to a preprocessed and tokenized description.
- Sentence Vector Generation: Developed a custom function, get_sentence_vector, to compute sentence vectors. This function iteratively processed each word in a sentence, summing their respective Word2Vec vectors if present in the model's vocabulary. This approach effectively created a cumulative vector representation for each sentence.
- Vector Integration in DataFrame: Executed the get_sentence_vector function across all tokenized sentences in tokens_list. The resulting sentence vectors were then stored in a new column, 'Description_Vector', within the original DataFrame (pp_data). This integration facilitated a seamless combination of the original textual data with its corresponding vectorized representations.
- Dataframe Transformation: The transformed DataFrame, now enriched with sentence vectors, provided a dual view - the original text descriptions alongside their vector representations. This format allowed for a more nuanced analysis and application in downstream tasks such as clustering, classification, or similarity searches.

Saving the Cleansed Data:

- Saved the processed DataFrame to a .csv file, ensuring that all the preprocessing steps and feature extractions are preserved for future use in modelling or further analysis.

Performance of the Model (Stats of Performance and Configuration Comparisons mostly by Grid Search Mechanism)

The model's performance is evaluated through a series of tests and comparisons. The primary method for optimization is the grid search mechanism, which helps in fine-tuning the model parameters to achieve the best results. The performance metrics are likely to include accuracy, precision, recall, and F1 scores, among others. These statistics provide a quantitative measure of the model's effectiveness in accurately identifying and categorizing industrial safety risks through the chatbot interface.

The following steps were followed for building a machine learning model and comparing the results:

Modelling using TF-IDF vectors:

Data Splitting:

- The TF-IDF matrix (tfidf_matrix) was used as the feature set (X), and the 'Accident Level' column from the preprocessed data (pp_data) served as the target variable (Y).
- The dataset was split into training and testing sets using a 70-30 split ratio, with stratification based on the target variable to ensure a balanced representation of each class.

Classifier Training and Evaluation:

- Five different classifiers were selected: Logistic Regression, Random Forest, K-Nearest Neighbor, Support Vector Machine, and AdaBoost.
- Each classifier was trained on the training set and then used to make predictions on the test set.

Performance Evaluation:

- The performance of each classifier was evaluated using accuracy, precision, recall, and F1-score metrics.
- A confusion matrix for each classifier was also generated to understand the distribution of predictions across different accident levels.

Results Compilation:

- The accuracy of each classifier was compiled into a DataFrame for a clear comparison.
- Here is a tabulated summary of the results:

Name	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Logistic Regression	74.22	55.0	74.0	63.0
Random Forest	74.22	55.0	74.0	63.0
K-Nearest Neighbor	74.22	55.0	74.0	63.0
Support Vector Machine	74.22	55.0	74.0	63.0
AdaBoost	71.88	55.0	72.0	62.0

Observations:

- All the models, except AdaBoost are providing the same accuracy of 74%, 2% higher than AdaBoost. But for such a highly imbalanced dataset, it's better to use weighted average of Precision, Recall & F1-Score as performance metrics.
- All the models provide very low performance, with only 55% Precision & 62-63% F1-Score, as only accidents of Level I have been correctly predicted and all the accidents in Level II - V are incorrectly predicted as Level I.
- This is due to the dataset being small and highly imbalanced with majority of the accidents from Level I.

Modelling using Word2Vec vectors:

Data Preparation:

- The Word2Vec vectors (X_wv) are extracted from the 'Description_Vector' column of the preprocessed data (pp_data).
- The target variable (Y) is presumably another column from the same dataset, though its nature isn't specified in your description.

Splitting Training and Testing Data:

- The dataset is divided into training and testing sets using the train_test_split function.
- A 70-30 split is applied, meaning 70% of the data is used for training and 30% for testing.
- The split is stratified based on the target variable (Y) to ensure each class is represented proportionally in both training and testing sets.

Classifier Training and Evaluation:

- Five different classifiers are used: Logistic Regression, Random Forest, K-Nearest Neighbor, Support Vector Machine, and AdaBoost.
- Each classifier is trained on the training set (x_train_wv, y_train) and then used to make predictions on the test set (x_test_wv).

Performance Evaluation:

- The performance of each classifier is assessed using several metrics: accuracy, precision, recall, and F1-score.
- A confusion matrix is also generated for each classifier to visualize the distribution of predictions across different classes.

Results Compilation:

- The accuracy of each classifier is compiled into a DataFrame for easy comparison.
- The table shows the performance metrics for each model: accuracy, precision, recall, and F1-score.

Model	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1-Score (%)**
Logistic Regression	74.21875	55.0	74.0	63.0
Random Forest	74.21875	55.0	74.0	63.0
K-Nearest Neighbor	74.21875	55.0	74.0	63.0
Support Vector Machine	74.21875	55.0	74.0	63.0
AdaBoost	71.87500	56.0	72.0	63.0

Observations:

- In our results, all classifiers except AdaBoost exhibit the same accuracy (74.21%), precision (55.0%), recall (74.0%), and F1-score (63.0%).
- AdaBoost shows slightly lower accuracy (71.87%) but slightly higher precision (56.0%).
- The high recall but low precision suggests that the models are biased towards predicting the majority class, as indicated by the confusion matrices which show a large number of true positives for class 'I' and zero for other classes.
- We can therefore observe that there is no difference in the model performance on both the Vectorization methods: TF-IDF & Word2Vec. Hence, going forward we can choose either one, we're selecting TF-TDF here as the importance of the words in a sentence matters to understand the cause of the accident and identify the Accident Level.

These findings highlight the challenge in predicting the accident level accurately due to the skewed nature of the dataset. The similar performance across diverse models suggests that the issue might not be the model selection but the underlying data distribution. Future steps could include addressing this imbalance, possibly through techniques like oversampling, under sampling, or synthetic data generation.

Modelling using TF-IDF vectors and Grid Search:

Initialization of Classifiers:

The function defines a dictionary classifiers_dict with different machine learning classifiers like Logistic Regression, Random Forest, K-Nearest Neighbor, Support Vector Machine, and AdaBoost, each initialized with their respective settings.

Parameter Grid Definition:

A param_grid dictionary is created, specifying the hyperparameters to be tuned for each classifier. For example, C values for Logistic Regression and SVC, n_estimators and max_depth for Random Forest, n_neighbors for K-Nearest Neighbor, and parameters for AdaBoost.

Grid Search and Model Fitting:

For each classifier, the function performs Grid Search using GridSearchCV. It fits the model on the training data (x_train, y_train) with various combinations of parameters defined in param_grid.

Selection of the Best Model:

After the Grid Search, the best model is selected based on the highest accuracy score. grid_search.best_estimator_ gives the best model for each classifier.

Model Evaluation:

The best model is then used to predict on the test data (x_test). The predictions are evaluated using various metrics like accuracy, precision, recall, and F1-score.

The classification_report and confusion_matrix functions are used to generate detailed performance reports and confusion matrices for each model.

Compilation of Results:

The function collects the performance metrics for each classifier and stores them in a dataframe. This includes the model name, accuracy, precision, recall, and F1-score.

Output:

Finally, the function returns the dataframe with the performance metrics for each classifier.

The following is the performance report of the various machine learning models:

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Logistic Regression	74.21875	55.0	74.0	63.0
Random Forest	74.21875	55.0	74.0	63.0
K-Nearest Neighbor	74.21875	55.0	74.0	63.0
Support Vector Machine	74.21875	55.0	74.0	63.0
AdaBoost	71.87500	55.0	72.0	62.0

In this case, after performing this process, it is observed that there was no significant difference in model performance after using Grid Search. The maximum accuracy achieved was 74.2% and the F1-score was 63%, which suggests that the hyperparameter tuning did not substantially improve the performance of the models over the baseline. This could indicate

that either the models are already optimized with the default parameters, or the features used for training are not sufficiently informative to achieve higher performance, or that the dataset itself is challenging in a way that limits model performance.

MILESTONE 2

Summary of problem statement, data and findings

The database consisted of records of accidents from 12 different plants in 03 different countries in Brazil. The main field of study here was 'Description' which has a detailed description of how the accident occurred. The objective was to build a Chatbot utility which could highlight the safety risk, as provided in the dataset using 'Accident Level', as per the incident description.

Findings about the Dataset:

1. The Dataset contains only 425 records, which is a low volume to be able to train a model to perfectly classify the Accident Level.
2. The Dataset contains records from 1st Jan 2016 – 9th July 2017, which helped us to identify only partial trends across months and weeks.
3. Most of the 'Critical Risks' are grouped under 'Others' which is about 50% of the dataset. The more common risks observed in the accidents are: Pressed, Manual Tools, Chemical Substances, Venomous Animals, Projection & Cut.
4. The dataset is highly imbalanced with 74% accidents only corresponding to Accident Level I, which are the least severe accidents. This has impacted the model's performances largely.

Implications:

1. We need to collect more data and from other countries to resolve the data issues.
2. Combined Classifier using Bi-directional LSTM model is working the best for the existing dataset, with 80% precision and 58% recall. This model combines two different Bi-directional LSTM classifiers – one for binary classification of Class I & Others (includes all the minority classes II-V) and the other for multi-class classification only among the minority classes Class II, III, IV & V.

Overview of the final process

In this section, we delve into the methodology applied in the development of our machine learning models, aimed at addressing the complexities of industrial safety through NLP.

SALIENT FEATURES OF DATA: We have a dataset of 425 accidents, which are classified among 5 levels of Accident (from least to most severe) – I, II, III, IV, V, based on the accident description and the associated risks. Majority (74%) of the accidents were least severe, across all the countries and industry sectors. Majority of Level I & II accidents could've potentially been Level III & IV accidents, hence the risks need to be carefully mitigated.

DATA PRE-PROCESSING: Below is the overview of steps taken to pre-process the data ('Description' column in the dataset):

1. We first performed a couple of checks – If there are any characters beyond the range of ASCII character set and HTML tags or URLs in the data. Since there were no such characters, no action was taken.
2. The numbers in the text are used for measurements, time, etc. hence provide meaningful information, but will not be helpful during vectorization and modelling, as it would add more noise and sparsity. Hence, we decided to retain only alphabetic characters.
3. The pre-processing function performs the following tasks – Lowercasing the entire data, retaining only alphabetic characters (as explained in pt. 2), removing English stopwords, lemmatizing the text using WordNetLemmatizer.

DATA PREPARATION: To prepare the data for training & testing classification models, below steps were taken:

1. We first tokenized the 'Description' using SpaCy library.
2. Using these tokens, we have generated N-Grams to identify the right number of words capturing richer semantic meaning and context, for Chatbot to recognize the sequences of words. The most frequent Bi-grams provide useful information regarding the cause of accident, while the unigrams seem to get covered in bi-grams and the tri-grams do not seem to be adding any useful information on top of bi-grams. For Example: 'injury', 'causing injury' & 'causing injury described'.

3. The last step was to generate word vectors, where we tried two different techniques TDF-IDF and Word2Vec, as TF-IDF uses a weighting scheme to understand the importance of a word while Wrod2Vec uses unsupervised learning approach to consider the context of the words.

ALGORITHMS & TECHNIQUES:

1. We initially started with basic machine learning classifiers – Logistic regression, Random Forest, K-Nearest Neighbors (KNN), Support Vector Machine (SVM) and AdaBoost. To improve the model's performance, we trained them over the vectors instead of words, and then also performed Grid Search to fine tune the models. Below is the summary of their performance at the end:

	Name	Accuracy (in %)	Precision (in %)	Recall (in %)	F1-Score (in %)
0	Logistic Regression	74.21875	55.0	74.0	63.0
1	Random Forest	74.21875	55.0	74.0	63.0
2	K-Nearest Neighbor	74.21875	55.0	74.0	63.0
3	Support Vector Machine	74.21875	55.0	74.0	63.0
4	AdaBoost	71.87500	55.0	72.0	62.0

2. The next obvious choice was Neural Network models: Neural Networks (NN), Recurrent Neural Networks (RNN), Long Short-Term Memory networks (LSTM), and Bidirectional Long Short-Term Memory networks (BiLSTM). Each model was created with an architecture having specific layer details to ensure proper and robust interpretation of the data.
3. For the NN, we used a combination of dense layers with activation functions tailored to our classification needs. The RNN was designed to capture temporal dependencies, with layers fine-tuned for sequence data processing. The LSTM model, known for its efficiency in handling long-term dependencies, included layers that were optimized for memory retention across longer sequences of text. Lastly, the BiLSTM expanded on the LSTM architecture by processing data in both forward and reverse directions to better understand context.
4. A notable aspect of our methodology was the comparison of model outputs using two different word embedding techniques: TF-IDF and GloVe. The NN models showed a

considerable level of improvement with TF-IDF, here's the performance summary:

	Model	Avg Training Accuracy	Test Accuracy	Precision	Recall
0	NN	0.713072	0.752941	0.566920	0.752941
1	RNN	0.881046	0.611765	0.543791	0.611765
2	LSTM	0.580882	0.752941	0.566920	0.752941
3	BiLSTM	0.857680	0.658824	0.633619	0.658824

5. The performance was not significantly good due to the data imbalance issue, hence two data augmentation techniques were combined to increase the training data in minority classes: Synonym Augmentation, which replaces some random words with their synonyms while duplicating the minority class records, and Contextual Word Embeddings Augmentation using BERT Generative Mode, which uses a pre-trained transformer-based model (BERT) to generate semantically similar but different sentences by considering the bidirectional context of words in a sentence.
6. Now, TF-IDF Vectorization technique focuses on the importance of terms in documents, which will not work with the above Data Augmentation techniques. Hence, we chose GLOVE EMBEDDINGS as the Vectorization technique, which is more effective for use cases involving semantic understanding, word similarity, and capturing relationships between words.
7. All the NN models were trained and tested on the augmented data in the form of vectors and here's the performance summary:

	Model	Avg Training Accuracy	Test Accuracy	Precision	Recall
0	NN	0.578450	0.671875	0.622106	0.671875
1	RNN	0.998789	0.468750	0.637802	0.468750
2	LSTM	0.670218	0.640625	0.629281	0.640625
3	BiLSTM	0.788741	0.593750	0.644931	0.593750

   As is evident, the Testing Accuracy, Precision & Recall, all have dropped after data augmentation.
8. As a further improvement to the model architecture, we built a Combined Classifier based on Bi-directional LSTM model, which is working the best, with 80% precision and 58% recall. This model combines two different Bi-directional LSTM classifiers – one for binary classification of Class I & Others (includes all the minority classes II-V) and the other for multi-class classification only among the minority classes Class II, III, IV & V. The details of this model will be addressed in the future questions.

Step-by-step walk through the solution

Milestone I steps are in details described in the Interim Report section above, hence the below only covers Milestone – II.

This milestone is for designing, training, and testing Neural Networks, RNN (Recurrent Neural Networks), LSTM (Long Short-Term Memory) models, and a Bi-directional LSTM model for a classification task related to industrial safety.

The goal is to predict the accident level based on text descriptions and vectorized representations of the incident.

Neural Network Model:

- The process begins by setting parameters such as input data (X for text, X_V for vectorized form), labels (Y), and other parameters.
- A simple neural network model is defined with a vector input layer, two dense layers (64 and 16 nodes), and a SoftMax output layer.
- The model is compiled using the Adam optimizer and sparse categorical cross-entropy loss.

RNN Model:

- RNN model where both text and vectors are considered as input branches.
- The text branch involves an embedding layer, a Simple RNN layer with 64 nodes, and an additional dense layer.
- The vector branch is a simple dense layer.
- The branches are concatenated, and a final dense layer with SoftMax activation is added.
- The model is compiled similarly to the Neural Network model.

LSTM Model:

- Like the RNN model, but it uses LSTM layers for the text branch.
- The text branch includes two LSTM layers with dropout, and the vector branch is a simple dense layer.
- The branches are concatenated, and a final dense layer with SoftMax activation is added.
- The model is compiled with the Adam optimizer, a specific learning rate, and L2 regularization.

Bi-Directional LSTM Model:

- This section defines a Bi-directional LSTM model where the text branch involves two Bi-directional LSTM layers.
- The vector branch and concatenation of branches are like the LSTM model.
- The model is compiled similarly to the previous models.

Data pre-processing

- Tokenization and padding are performed on the text data using the Keras Tokenizer.
- Vector data and labels are prepared.

Train and test data split

- text_data: This variable contains the pre-processed and padded text data (sequences of word indices) for the machine learning model.
- vector_data: This variable contains the vectorized representations of the incident descriptions.
- labels: This variable contains the corresponding labels or target values (accident levels) for the given data.
- test_size: This parameter specifies the proportion of the dataset to include in the test split. In this case, 20% of the data will be used for testing.

- random_state: This parameter is used to ensure reproducibility. Setting a random seed (random_state) to a specific value ensures that the split is the same every time you run the code. This is helpful for reproducibility and consistent results, especially during development and testing.

Train and evaluate the model

function train_and_evaluate that trains and evaluates different types of models, specifically a Simple Neural Network, an RNN (Recurrent Neural Network), an LSTM (Long Short-Term Memory) model, and a Bidirectional LSTM model. The function takes the model and the necessary data (training and testing sets) as input and returns various performance metrics.

this script trains and evaluates different models on the provided data, computes various performance metrics (such as accuracy, precision, and recall), and prints confusion matrices for each model. It allows for a comparative analysis of the models' performances. The choice of the number of epochs, batch size, and other hyperparameters can be adjusted based on the specific characteristics of your data and task.

Performance of various models

	Model	Avg Training Accuracy	Test Accuracy	Precision	Recall
0	NN	0.713072	0.752941	0.566920	0.752941
1	RNN	0.881046	0.611765	0.543791	0.611765
2	LSTM	0.580882	0.752941	0.566920	0.752941
3	BiLSTM	0.857680	0.658824	0.633619	0.658824

The data frame summarizes the performance of various models, revealing key observations:

1. The Training accuracy is highest but testing accuracy has dropped with RNN model.
2. The Simple Neural Network model is performing the best for testing data but has lowest accuracy on Training data.
3. As the training accuracy is dropping for models, the testing accuracy is increasing respectively.
4. Precision is almost similar with all the models, with highest being in LSTM model.

5. Compared to basic ML classifiers, Neural Network models are performing better on minority classes but it's significantly low, hence **we need to perform Data Augmentation to balance the dataset.**

Balancing the class distribution in the dataset using NLP Augmentation techniques

Balancing the class distribution through NLP augmentation techniques is a crucial step in developing robust and fair machine learning models, especially in domains where certain classes are rare but highly significant. It promotes better model performance, generalization, and accuracy across all classes, contributing to the reliability and effectiveness of the model in real-world applications.

This section focuses on balancing the class distribution in a dataset using Natural Language Processing (NLP) augmentation techniques and showcases a comprehensive approach to address class imbalance by augmenting the dataset through synonym replacement and contextual word embeddings. It further demonstrates the utilization of different vectorization techniques and training of neural network models on the augmented data.

The steps include:

- Synonym Augmentation:
    a. Synonym augmentation is performed on the training data for the minority classes ('II', 'III', 'IV', 'V').
    b. Synonym augmentation involves replacing words with their synonyms to increase data diversity.
- Contextual Word Embeddings Augmentation using BERT Generative Model:
    a. Augmentation is performed using a contextual word embeddings model based on BERT.
    b. The process involves substituting words in the descriptions to generate augmented data for minority classes.

<u>Outcome of data augmentation:</u>

1. Before augmentation, the dataset exhibits a significant class imbalance, with class I being the majority class (216 instances), and classes II, III, IV, and V being minority classes with 28, 24, 24, and 5 instances, respectively.
2. After performing synonym augmentation, the dataset is expanded, and the instances for each class increase. Class II, III, IV, and V now have 56, 48, 48, and 10 instances, respectively, reflecting an attempt to balance the class distribution by generating additional instances through synonym replacement.
3. Further augmentation using contextual word embeddings with a BERT generative model results in a more pronounced increase in the instances of the minority classes. Class II, III, IV, and V now have 84, 72, 72, and 15 instances, respectively.
4. This additional augmentation contributes to further addressing the class imbalance in the dataset, enhancing the representation of the minority classes to improve the model's ability to recognize and generalize well for all classes.

Performing Vectorization over the augmented training data and testing data

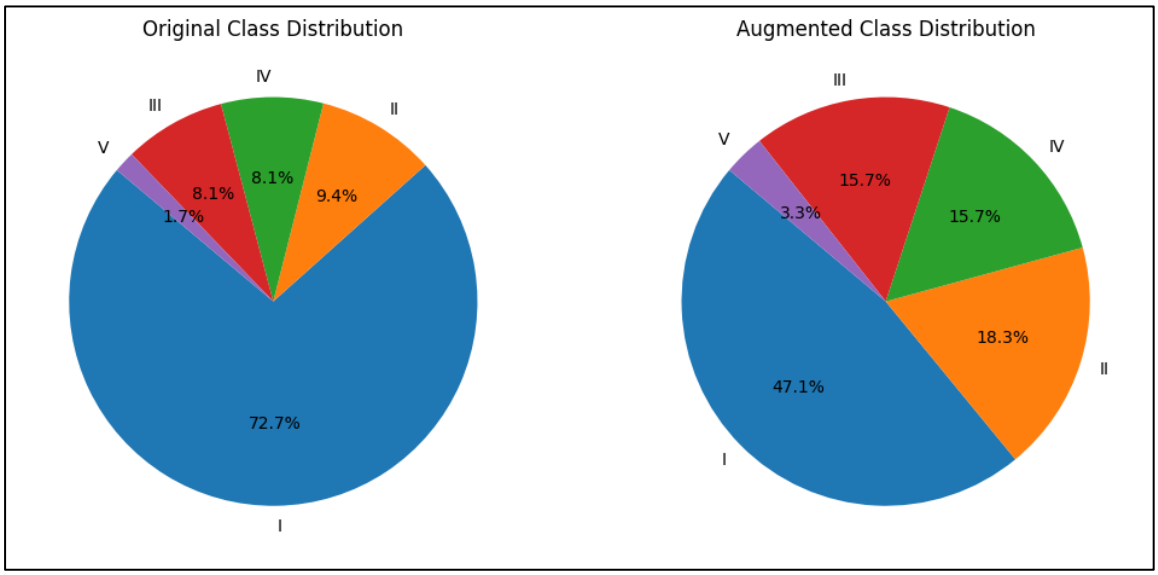
- Vectorization of Augmented Training and Testing Data:
    a. GloVe embeddings and TF-IDF vectorization techniques are applied to the augmented training data, generating GloVe embedding vectors and TF-IDF vectors, respectively.
- Training Neural Network Models on Augmented Data:
    a. The goal is to prepares the augmented data for training by tokenizing the text data and converting it into sequences.
    b. The GloVe embedding vectors and TF-IDF vectors are used as input features for training the neural network models.

Outcome of Vectorization:

The shapes of the training and testing datasets after applying GloVe embeddings and TF-IDF vectors are as follows:

1. Training Dataset with GloVe Embeddings: (459 samples, 2 features)
2. Testing Dataset with GloVe Embeddings: (128 samples, 2 features)

3. Training Dataset with TF-IDF Vectors: (459 samples, 50 features)
4. Testing Dataset with TF-IDF Vectors: (128 samples, 50 features)



Grid search for Hyperparameter configuration

This section is to perform a grid search for hyperparameter tuning on two types of neural network models: Simple Neural Network (NN) and Recurrent Neural Network (RNN). The hyperparameter being tuned is the learning rate of the Adam optimizer.

Simple NN Grid Search:

1. A function perform_grid_search_nn is defined for grid search on the Simple NN model. It takes the base NN model, a parameter grid with different learning rates, and training/testing data.
2. The grid search is performed for various learning rates using the Adam optimizer.
3. The results, including average training accuracy, test accuracy, precision, and recall for each configuration, are stored in a DataFrame.

RNN Grid Search:

1. A similar function perform_grid_search_rnn is defined for grid search on the RNN model. It takes the base RNN model, a parameter grid with different learning rates, and training/testing data.
2. The grid search is performed for various learning rates using the Adam optimizer.

3. The results, including average training accuracy, test accuracy, precision, and recall for each configuration, are stored in a DataFrame.

Combining Results:

1. The results of both grid searches (Simple NN and RNN) are combined into a single DataFrame (grid_search_results_combined).
2. The combined DataFrame is then formatted for easy interpretation, separating parameters into columns.
3. The final DataFrame (grid_search_results_df) provides a summary of the grid search results, indicating whether each row corresponds to an NN or RNN model, the learning rate, average training accuracy, test accuracy, precision, and recall.
4. The resulting DataFrame can be used to identify the optimal hyperparameters for the given models and dataset.

Evaluation Summary:

	Model	Avg Training Accuracy	Test Accuracy	Precision	Recall
0	NN	0.578450	0.671875	0.622106	0.671875
1	RNN	0.998789	0.468750	0.637802	0.468750
2	LSTM	0.670218	0.640625	0.629281	0.640625
3	BiLSTM	0.788741	0.593750	0.644931	0.593750

Observations:

1. The Simple Neural Network model shows relatively balanced performance between training and testing, with a decent accuracy and precision.
2. The Recurrent Neural Network (RNN) model, on the other hand, demonstrates high training accuracy but struggles with generalization, as seen in the lower test accuracy.
3. The LSTM and Bidirectional LSTM models fall in between, showing better generalization compared to the RNN but not as strong as the Simple NN

Recommendations:

1. Further tuning and exploration of model architectures and hyperparameters may improve the overall performance, especially for the RNN-based models.
2. Consideration of additional techniques such as dropout and regularization might help prevent overfitting in high-capacity models.
3. This summary provides insights into the relative performance of different neural network architectures on the given dataset.

Use of Grid Search and Combined Classification Process on GloVe Embeddings and Augmented Data

This section is about implementing a two-level classification strategy, where the first level classifies whether an instance belongs to Class 1 or not (binary classification), and the second level focuses on further classifying instances belonging to classes 2, 3, 4, and 5. The model used for both levels is a Bidirectional LSTM (BiLSTM). This strategy is particularly useful when dealing with imbalanced multiclass classification problems, allowing for a more targeted approach to handling different classes. Let's break down the key steps:

Binary Classification (First Level):

- Convert the original labels (y_train and y_test) into binary labels: 1 for Class 1 and 0 for Not Class 1.
- Define and train a BiLSTM model (model_bilstm_2) for binary classification using the converted binary labels.
- Evaluate the performance of the first level classifier using metrics such as accuracy, precision, recall, and the confusion matrix.

Filtering Data for Second Level:

- Filter the training and testing data to include only instances belonging to classes 2, 3, 4, and 5. This is done by creating boolean masks (mask_train and mask_test).
- Create new datasets (X_train_text_second_level, X_train_vector_second_level, y_train_second_level, X_test_text_second_level, X_test_vector_second_level, y_test_second_level) containing instances for classes 2, 3, 4, and 5.

Multiclass Classification (Second Level):

- Define and train another BiLSTM model (model_bilstm_3) for multiclass classification using the filtered datasets.
- Evaluate the performance of the second level classifier using metrics similar to the first level.
- Printing Results:

Observations:

```
Combined Classifier Performance:
              precision    recall  f1-score   support

           0       1.00      0.66      0.80       100
           1       0.11      0.42      0.17        12
           2       0.19      0.43      0.26         7
           3       0.00      0.00      0.00         6
           4       0.00      0.00      0.00         3

    accuracy                           0.58       128
   macro avg       0.26      0.30      0.25       128
weighted avg       0.80      0.58      0.65       128

Confusion Matrix:
 [[66 34  0  0  0]
 [ 0  5  7  0  0]
 [ 0  4  3  0  0]
 [ 0  2  4  0  0]
 [ 0  1  2  0  0]]
```

- This has the best combination of our performance metrics: Recall (58%) & Precision (80%), with 100% accuracy of Accident Level I class.
- The two-level approach allows for a more focused and nuanced classification strategy, addressing the complexity of multiclass problems.
- The filtering of data for the second level ensures that the second classifier focuses only on the relevant classes.

Model evaluation

Our final model is a nested classifier that combines the results from two classifiers. The first model classifies whether a given description belongs to the majority class (Accident Severity I) or not. The second model classifies the other classes (Severity II, III, IV and V).

Thus, the data is prepared separately for the first classifier and the second classifier. For the first classifier the target variable is modified to reflect only two classes i.e. Severity I or Not Severity I. For the second classifier, the dataset is filtered to contain only classes other than Severity I i.e. Severity II, III, IV and V.

The base model used here is GloVe-BiLSTM (Bi Directional LSTM using GloVe embeddings) for both the classifiers. The results of both the classifiers are combined, as it is a nested classifier and the performance metrics for the test data was generated.

The following is the architecture of the base model (GloVe-BiLSTM):

- Text Branch:
    - Input Layer: Handles text input with a maximum length defined by max_text_length.
    - Embedding Layer: Transforms the text input into dense vectors of size embedding_dim.
    - Bi Directional LSTM Layers: Two Bidirectional LSTM layers with 128 and 64 units, respectively, featuring dropout and recurrent dropout to prevent overfitting. The first BiLSTM layer returns sequences, allowing the next BiLSTM layer to capture longer dependencies in the text data.The use of bidirectionality allows the model to capture context from both directions in the sequence.

- Vector Branch:
    - Input Layer: Designed to process additional vector-based inputs of dimension vector_dim.
    - Dense Layer: A fully connected layer with 64 units and ReLU activation, processing the vector data.

- Concatenation and Final Layers:
    - Concatenate Layer: Merges the outputs of the text and vector branches.
    - Additional Dense Layer: A fully connected layer with 16 units and ReLU activation, further processing the concatenated data.
    - Output Layer: A Dense layer with num_classes units and a softmax activation function, regularized with L2 regularization.

Following are few prominent parameters in the Bi-LSTM model:

- Layer Units and Structure:
  - The number of units in the LSTM layers (128 and 64 units respectively) is a crucial parameter. These units determine the capacity of the model to learn and capture patterns in the data.
  - The architecture's bidirectional approach allows the model to gather context from both past and future points in the sequence, enhancing its ability to understand the textual data more comprehensively.
- Dropout and Recurrent Dropout:
  - These parameters, set at 0.2 and 0.3 for different layers, play a significant role in preventing overfitting. They randomly deactivate a fraction of neurons during training, forcing the model to learn more robust features.
- Embedding Dimensions:
  - The embedding layer's dimension (embedding_dim) is a key parameter, as it transforms input text into a specified vector space, impacting how textual information is represented internally.
- Regularization:
  - The L2 regularization in the output layer helps in preventing overfitting by penalizing large weights, ensuring a more generalized model.
- Optimization Parameters:
  - The Adam optimizer and the learning rate influence the speed and quality of the model's training process. The loss function 'sparse_categorical_crossentropy' is well-suited for multi-class classification tasks.

The objective was to create a model that takes the ‘Accident Description’ as input and provides the ‘Accident Severity’ class as output. The objective was to make the model be as accurate as possible, considering even recall, precision and F1 score as additional metrics.

Following are the performance metrics used to evaluate the models:

- Accuracy: Measures the proportion of correctly predicted instances among the total instances.
- Precision: Assesses the accuracy of the model in predicting severe accidents.
- Recall: Evaluates the model's ability to correctly identify all severe accidents.
- F1-Score: Provides a balance between precision and recall, crucial for imbalanced datasets.
- Confusion Matrix: Offers a detailed view of the model's performance, distinguishing between true positives, false positives, true negatives, and false negatives.

Class	Precision	Recall	F1-Score
0	1.00	0.66	0.80
1	0.11	0.42	0.17
2	0.19	0.43	0.26
3	0.00	0.00	0.00
4	0.00	0.00	0.00

Score	Precision	Recall	F1-Score	Accuracy
Weighted Average	0.80	0.58	0.65	0.58

Comparison to benchmark

Our selected benchmark was the least performing model, which turned out to be AdaBoost algorithm, with 55% Precision.
Below is the comparison of performance metrics against the Benchmark (all values are in %):

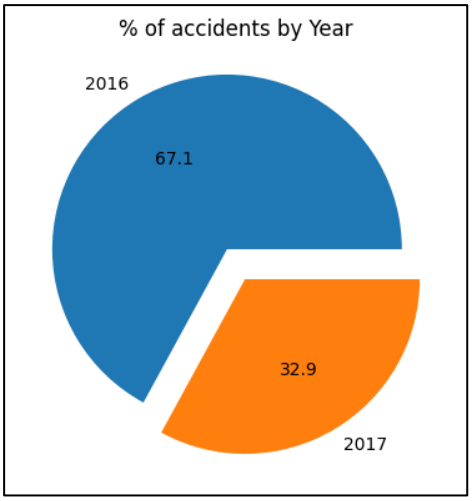
Model	Vectorization Method	Precision	Precision v/s Benchmark	Recall	Recall v/s Benchmark	Test Accuracy	Accuracy v/s Benchmark
BENCHMARK MODEL							
AdaBoost	TF-IDF, Word2Vec	55.0		72.0		71.9	
Improving Performance with other BASIC ML CLASSIFIERS, Word2Vec Vectorization and Grid Search Hyperparameter Tuning							
Logistic regression	TF-IDF, Word2Vec	55.0	0.0	74.0	2.0	74.2	2.3
Random Forest	TF-IDF, Word2Vec	55.0	0.0	74.0	2.0	74.2	2.3
K-Nearest Neighbors	TF-IDF, Word2Vec	55.0	0.0	74.0	2.0	74.2	2.3
Support Vector Machine	TF-IDF, Word2Vec	55.0	0.0	74.0	2.0	74.2	2.3
AdaBoost	TF-IDF, Word2Vec	55.0	0.0	72.0	0.0	71.9	0.0
Improving Performance with NEURAL NETWORK MODELS							
Simple NN	TF-IDF	56.7	1.7	75.3	3.3	75.3	3.4
Recurrent NN	TF-IDF	54.5	-0.5	61.2	-10.8	61.2	-10.7
Long Short-Term Memory NN	TF-IDF	56.7	1.7	75.3	3.3	75.3	3.4
Bi-Directional LSTM NN	TF-IDF	63.3	8.3	65.9	-6.1	65.9	-6.0
Improving Performance on NEURAL NETWORK MODELS with Data Augmentation & Grid Search							
Simple NN	GloVe Embeddings	62.2	7.2	67.2	-4.8	67.2	-4.7
Recurrent NN	GloVe Embeddings	63.8	8.8	46.9	-25.1	46.9	-25.0
Long Short-Term Memory NN	GloVe Embeddings	62.9	7.9	64.1	-7.9	64.1	-7.8
Bi-Directional LSTM NN	GloVe Embeddings	64.4	9.4	59.4	-12.6	59.4	-12.5
Improving Performance with COMBINED CLASSIFIER using Bi-LSTM NN Model							
Combined Classifier using Bi-LSTM NN	GloVe Embeddings	80.0	25.0	58.0	-14.0	65.0	-6.9

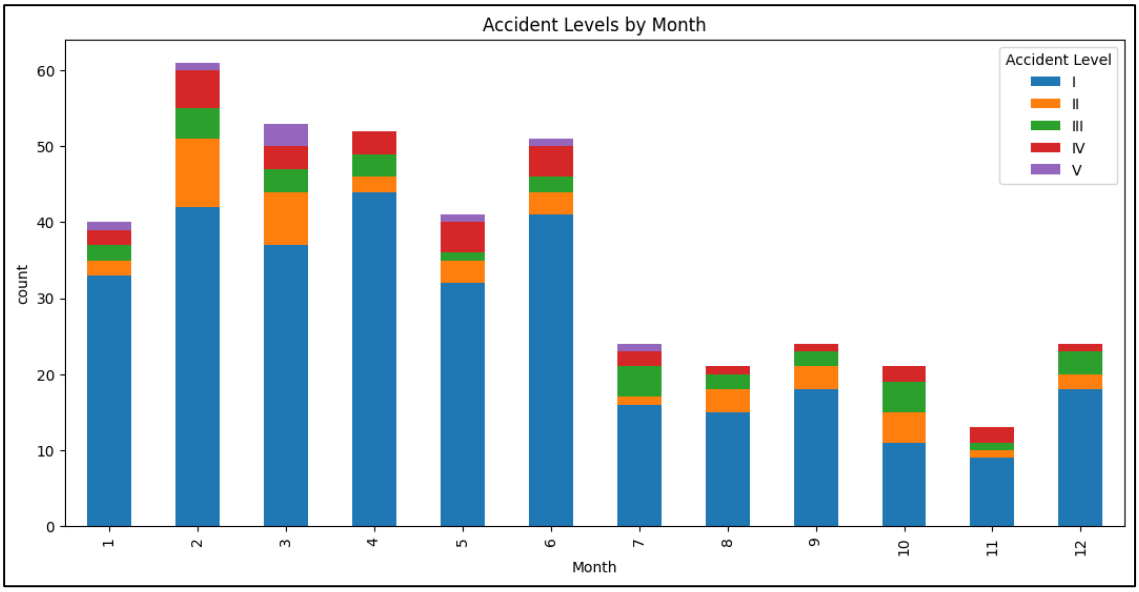
Different performance improvement methods have been tried as shown in the above table:

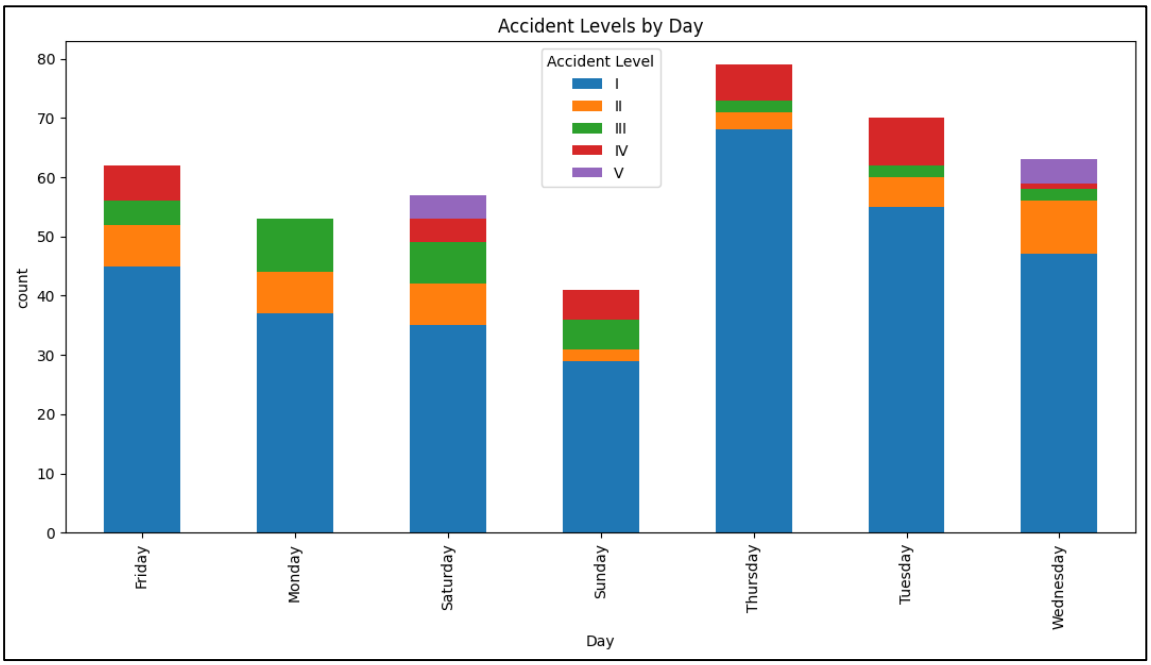
1. Other than AdaBoost, we implemented 4 Basic Machine Learning Classifier models – Logistic regression, Random Forest, KNN & SVM.
2. Two different Vectorization techniques were used for improving the performance of Basic ML Classifier models, but there was no change in the performance.
3. Grid Search was used to identify the best hyperparameters for each of these models, but there still was no improvement in the performance of any of these models.
4. Taking it a step ahead, we implemented 4 different Neural Network models – Simple NN, Recurrent NN, Long Short-Term Memory and Bi-directional LSTM. The performance metrics improved considerably with Simple NN, LSTM & Bi-directional LSTM model, but RNN model exhibited degraded performance.
5. To mitigate the imbalanced dataset issue, we performed Data Augmentation techniques and also used Grid Search to fine tune the model hyperparameters, after which all the models' performance was decreased.
6. Hence, we tried to split the majority & minority classes and classified them separately using Bi-Directional LSTM model. The we combined the two classifiers and ran the Combined Classifier model, which displayed significant improvement over the standard models above. Hence, we chose this as the best model.

Visualizations

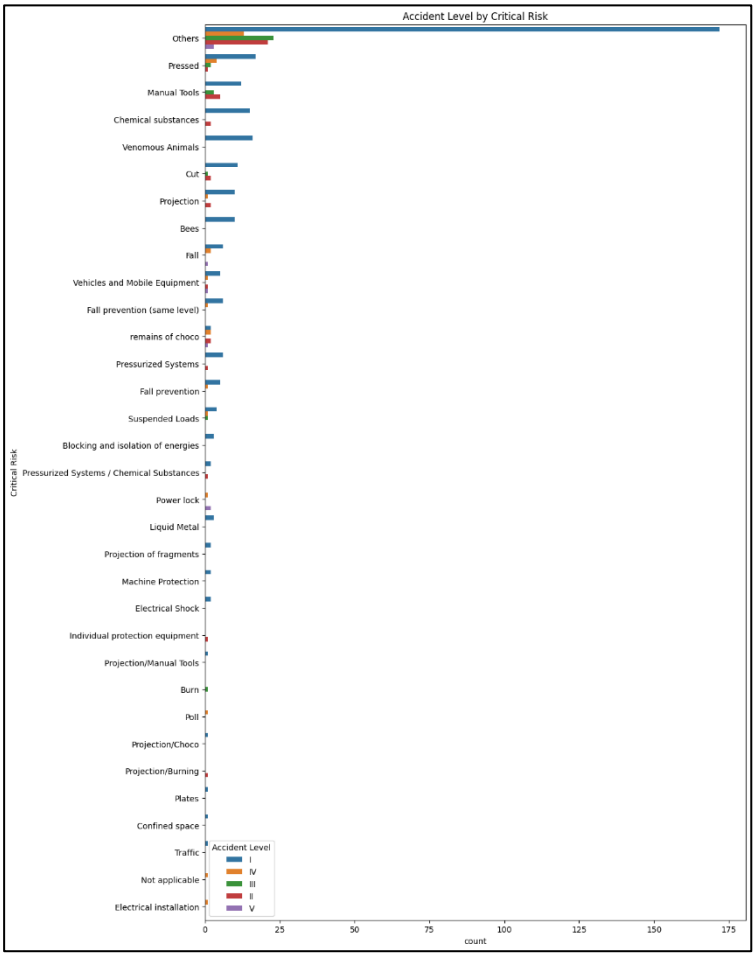
1. The Dataset contains records from 1st Jan 2016 – 9th July 2017, which helped us to identify only partial trends, such as:
    a. For Both 2016 & 2017, about 70-80% accidents have happened during the weekdays. People travelling for work on weekdays could be the possible reason for this.
    b. For every month from Feb - Jun, the number of accidents have reduced in 2017. There is not enough data to know the reason behind this and if this trend would continue.



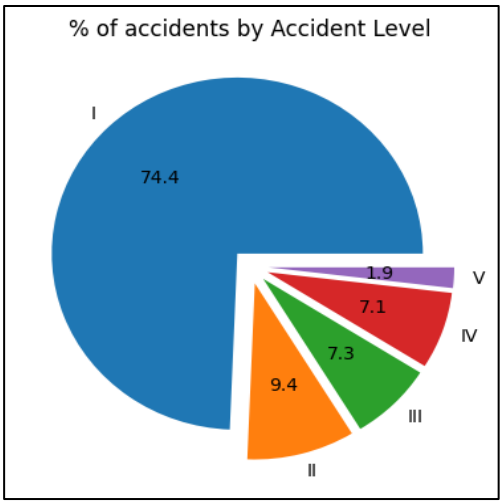




2. Most of the ‘Critical Risks’ are grouped under ‘Others’ which is about 50% of the dataset. The more common risks observed in the accidents are: Pressed, Manual Tools, Chemical Substances, Venomous Animals, Projection & Cut.

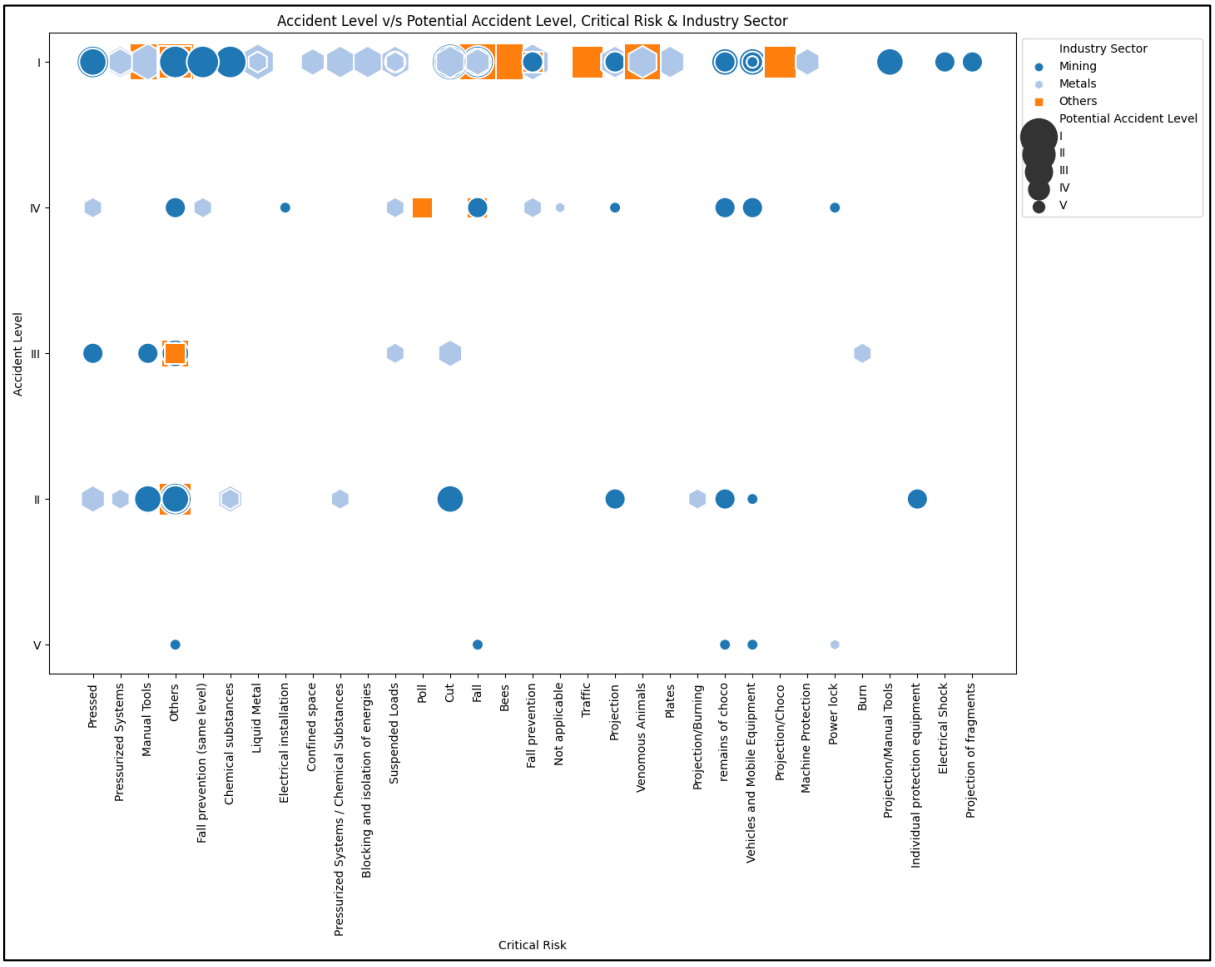


3. The dataset is highly imbalanced with 74% accidents only corresponding to Accident Level I, which are the least severe accidents. This has impacted the model's performances largely.

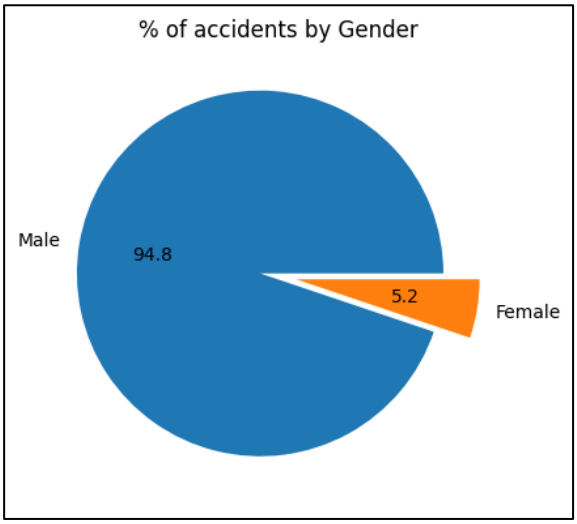


4. Majority of the risks have been recorded from only Country_01, which could either mean there are lesser accidents due other countries or lesser population, but there is not enough data to conclude. This could've led to an action-oriented recommendation.
5. Although 'Fall' is the most common risk across industries but there are industry specific risks as well, which can be focussed on to improve the ways of working in

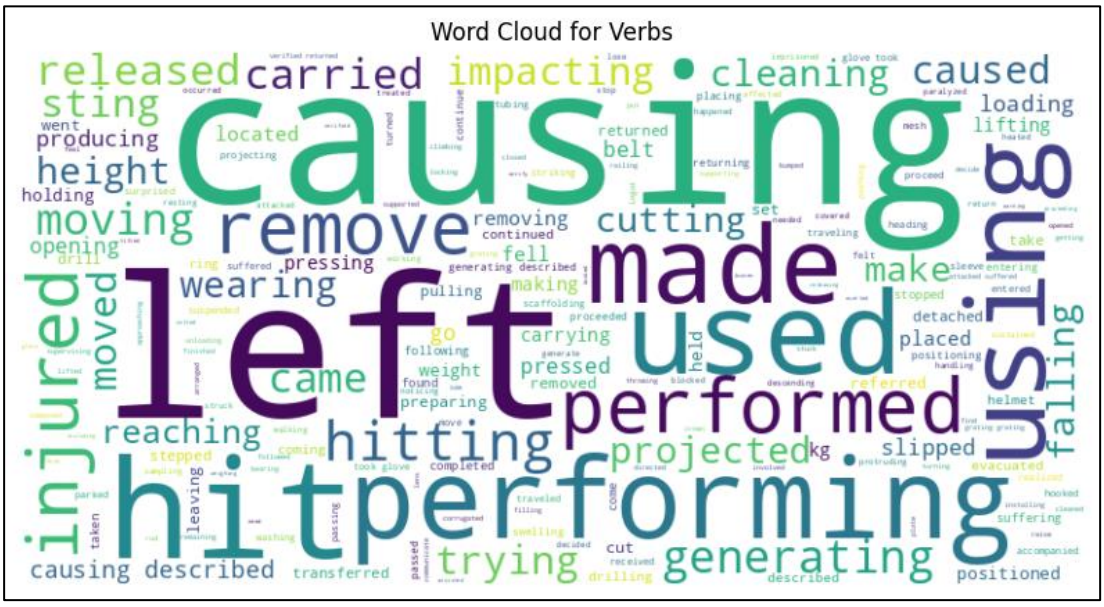
order to prevent accidents.



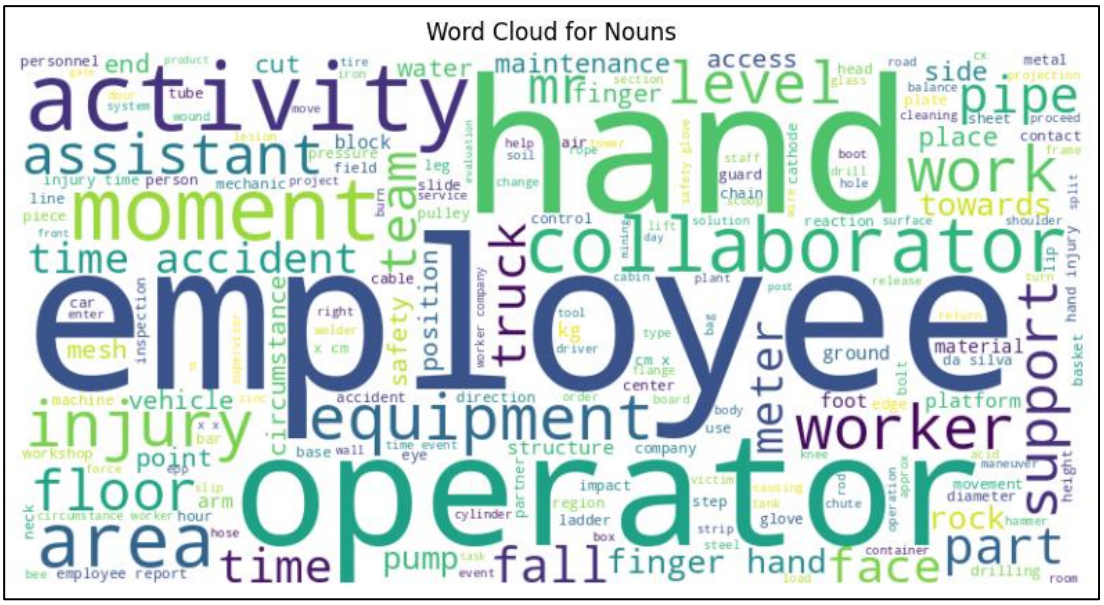
6. We can see the only ~5% Females have encountered accidents. This could be due to a general assumption of majority of industrial jobs being taken up by Males.



7. Using POS Tagging on the 'Description', we found action words like: falling, cutting, drilling, hitting, cleaning, slipped, injured, carried, moving, pressing, etc., which must have led to the accident.



We also found nouns which can also help us understand the object causing the accident, like: truck, pipe, bee, vehicle, tube, floor, support mesh, ladder, helmet, metal, bolt, rock, water, tool, sting, equipment, etc.



Implications

Industrial safety is a major concern in all industries and particularly in those where manual labor is involved with exposure to working hazards. Handling and responding to accidents, understanding the historical trends and classifying accidents to properly make risk assessments and implement preventive measures is critical for continuous improvement of

operations and processes. A key focus of this project was to provide the accident severity level for immediate response to an accident based on the accident description. A chatbot would accept the accident description from anyone with access to it, and the system can alert the concerned medical staff or other responders to aid in the medical treatment required and other protocols that may be in place. Another aspect is the improvement of preventive measures that can be adopted based on the classification report of various incidents. Implementing this chatbot could lead to a safer working environment, better response towards incidents, potentially reducing accidents and improving overall safety protocols.

Based on the data analysis and the model building, we can conclude that accident description is likely a very useful indicator for determining the severity level and subsequent response to an accident. AI based models have the potential to capture and effectively classify the severity based on the description alone. However, other factors such as gender, location, country, industry etc. can make the prediction more focussed and targeted.

Our current model would be capable of fairly classifying a ‘Level-I’ accident as opposed to ‘Higher Than Level-1’ accident with a very high level of confidence but, the gradation of other risk levels is not reliable as of now. We recommend implementation of this model on a test basis, aiding decision makers to help guide their responses.

Limitations

The solution's limitations are primarily around the quality and diversity of the input data; the data set that was used to train the models was limited in size and also suffered class imbalance. Due to this the AI models may suffer overfitting during training and may not generalize well in production. Another drawback is the class imbalance, where more severe accidents are fewer in number compared to more routine ones. Thus, the critical aspect of identifying severe accidents is missed in the model which is a major drawback and can be rectified only with a well balanced and large dataset. Further enhancements could include expanding the dataset, either in-house or through secondary sources like other accident databases and improving the algorithm's ability to understand nuanced language and complex scenarios through a larger neural network. A text specific analysis on the description to create novel preprocessing steps designed to capture important information for accidents could be a key area of exploration. Another way to mitigate the drawback is by the use of pre-trained language models that are capable of understanding text better and re-training them to customize it to our problem statement.

Overall, the key issue with the model is that it is limited by its inability to discern complex, non-obvious patterns in accident reports, due to imbalanced data and its limited size. Analysis

using four AI models—NN, RNN, LSTM, BiLSTM—revealed these constraints despite nuanced NLP architectures.

Closing Reflections

The project highlighted the importance of robust data pre-processing and the potential of NLP in safety management. Lessons learned include the need for comprehensive data and the value of iterative testing across various models, architectures, strategies and preprocessing steps to arrive at a feasible AI model. Future projects will benefit from a more diverse dataset, use of pre-trained models, use of automation pipelines throughout the development process, including continuous and periodic retraining of the models. Development of the various processes using multi-thread architecture with execution blocks, could make the model deployment ready in any platform. The process has been a learning curve in balancing technical feasibility with practical utility. We've learned the importance of iterative development in creating AI solutions.

In future projects, we would allocate more effort, time and resources towards concrete goals, objectives, planning, architecture and expected outcomes. Also, more focus will be placed on the ethical implications of AI, ensuring our models are fair and unbiased and don't mislead decision makers to downplay accident levels. Since this model is a basic one and does not account for the possibility that the classification can lead to certain accidents being taken lightly, this should be used with caution.

Future Scope:

1. We can create a pipeline which includes all the steps from data pre-processing to Model training, so this model is ready to deploy.
2. We can build models using pre-trained neural network architecture, which are trained on larger dataset.