CAPSTONE PROJECT
NLP CHATBOT INTERFACE
FINAL REPORT

Group 6

Team Members

Amit Pudkey

Arul Kumar Raman

Ananya Sathyanarayana

Kushal Voona

Shrey Rathi

Shobhit Verma

Table of Contents

Problem Statement

Introduction

In today's industrial world, keeping workers safe remains a pressing challenge. Even with modern safety measures, employees continue to face workplace accidents that can lead to injuries or worse. Our focus is on a major Brazilian industry where understanding why accidents happen - and preventing them - is crucial for protecting workers' lives. While companies collect extensive data about these incidents, making sense of this information quickly and effectively is difficult for safety professionals. This project introduces an intelligent chatbot that uses Natural Language Processing to help these professionals rapidly analyse accident reports and identify safety risks, ultimately working towards a safer workplace for everyone.

Project Objective

The primary objective is to design a machine learning (ML) based chatbot utility. This tool is intended to help safety professionals in identifying and highlighting safety risks based on incident descriptions. By analysing 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.

Data Description

The dataset is obtained from one of the biggest industries in Brazil and in the world. It is a record of accidents from 12 different plants in 3 different countries.

Data Components of the columns

- Time-based Information
  - Data: timestamp or time/date information
- Geographic Information
  - Countries: which country the accident occurred (anonymised)
  - Local: the city where the manufacturing plant is located (anonymised)
  - Industry sector classification

- Accident Classification
    - Accident level: Severity levels (I to V)
    - Potential accident levels: Severity levels (I to VI)
    - Critical Risk: some description of the risk involved in the accident
- Personnel Information
    - Genre: if the person is male of female
    - Employee or Third Party: if the injured person is an employee or a third party
- Descriptive Elements
    - Description: Detailed description of how the accident happened.

Solution approach

The solution was developed in two parts:

Milestone 1: Includes steps for Data cleaning, text preprocessing, EDA and applying traditional ML algorithms

Milestone 2: Includes experimenting with performance improvement techniques like SMOTE ad NLP Augmentation along with using Neural networks (specially transformer models from Hugging Face Hub) to solve the problem.

For more details, refer respective sections

MILESTONE 1

Data Preprocessing

Data Cleaning

- Initial data contained 425 rows and 11 columns
- Columns were renamed to follow a more standard convention like Data to Date, Countries to Country, Local to City, using '_' as a separator in column names
- Removed Unnamed:0 column as it was unique and more like a duplicate index
- Identified 7 duplicate rows after removing above column, these were dropped
- Further identified 7 rows with same Description, these were also dropped
- There was only one row with Potential Accident Level = VI, this was updated to the closest severity i.e. Level V
- Post data cleaning, dataset contains 411 Rows and 10 Columns

Feature Engineering

- Extract the Year, Month, Date, Day from the Date Columns
- Add number of words in Description as a new column -Word_Count
- Now Data set has 411 rows and 15 columns

Text Preprocessing

- Checked for non-ASCII and HTML tags, found no discrepancy
- Used WordNet and NLTK to perform below operations:
    - Removed special characters
    - Converted all to lower case
    - Trimmed white spaces
    - Removed stop words
    - Apply Lemmatizer

Exploratory Data Analysis

Summary of Data

Defined a method detailed_summary that can display all the statistics of numerical and categorical variables in the given data frame

Code snapshot:

```
def detailed_summary(df):
  '''
    Print a summary of the given dataframe to display no of records, columns,
    datatype and names of each column.
    Also list the Missing values, Duplicate values, if any along with other details


  '''

  print(f'Data set has {df.shape[0]} rows and {df.shape[1]} columns')
  print(f'Various datatypes present in the dataset are:\n{df.dtypes.value_counts()}')
  summary = pd.DataFrame(df.dtypes, columns = ['Datatypes'])
  summary = summary.reset_index()
  summary['Missing_values'] = df.isnull().sum()
  summary['Missing_values'].fillna(0,inplace=True)
  summary['Unique_values'] = df.nunique().values
  summary['Duplicate_values'] = df.duplicated().sum()
  summary['First_value'] = df.loc[0].values
  summary['Second_value'] = df.loc[1].values
  return summary
```

Sample output as below:

```
detailed_summary(df)

Data set has 425 rows and 11 columns
Various datatypes present in the dataset are:
object            9
int64             1
datetime64[ns]    1
Name: count, dtype: int64
```

	index	Datatypes	Missing_values	Unique_values	Duplicate_values	First_value	Second_value
0	Unnamed: 0	int64	0.0	425	0	0	1
1	Date	datetime64[ns]	0.0	287	0	2016-01-01 00:00:00	2016-01-02 00:00:00
2	Country	object	0.0	3	0	Country_01	Country_02
3	City	object	0.0	12	0	Local_01	Local_02
4	Industry_Sector	object	0.0	3	0	Mining	Mining
5	Accident_Level	object	0.0	5	0	I	I
6	Potential_Accident_Level	object	0.0	6	0	IV	IV
7	Gender	object	0.0	2	0	Male	Male
8	Employee_Type	object	0.0	3	0	Third Party	Employee
9	Critical_Risk	object	0.0	33	0	Pressed	Pressurized Systems
						While removing the drill rod of the	

Univariate Analysis

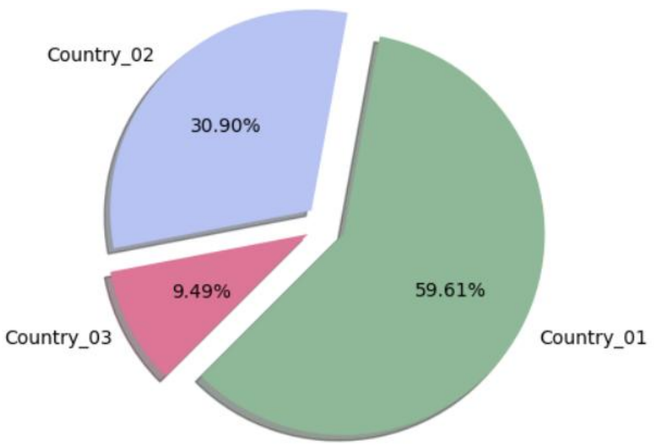
Helper method to plot distribution of categorical features

Defined a method to visualise either a plie chart or a count plot based on the number of unique values in the given categorical column

Snapshot of the code:

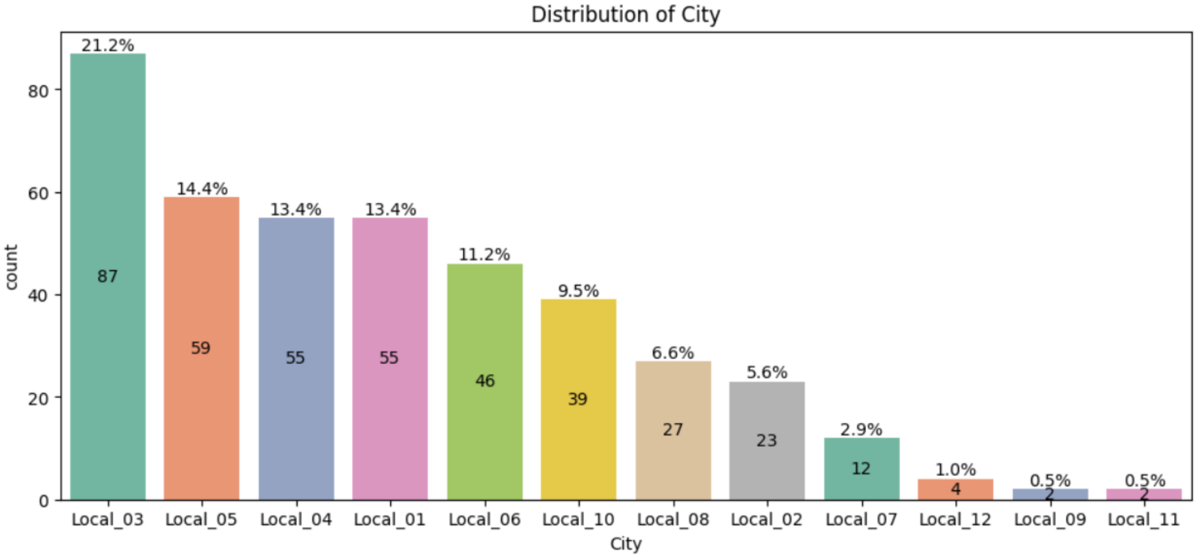
```
def visualise_cat_features(df,col):

    '''
      Plots a pie chart if feature has less than 5 unique values
      Plots a countplot if it has >=5 and <=20 unique values
      Prints an error message if it has >20 unique values

    '''

    num_values = df[col].nunique()
    values_count = df[col].value_counts()


    if num_values <= 4:

        #pie
        colors  = ['#8EB897', '#B7C3F3', '#DD7596', '#4F6272' ]
        plt.pie(values_count, labels = values_count.index, colors=colors, autopct='%.2f%%',
              shadow = True, explode = [0.1]*len(values_count.index), startangle = -135)
        plt.title(f"Distribution of {col}")
        plt.show()

    elif num_values <= 20:

        #countplot
        plt.figure(figsize = (12,5))
        ax = sns.countplot(df,x=col,order=values_count.index,palette='Set2')

        # Calculate percentages
        total = len(df[col])

        # Add labels on the bars
        for p in ax.patches:

            count = p.get_height()
```

Analysis by Country



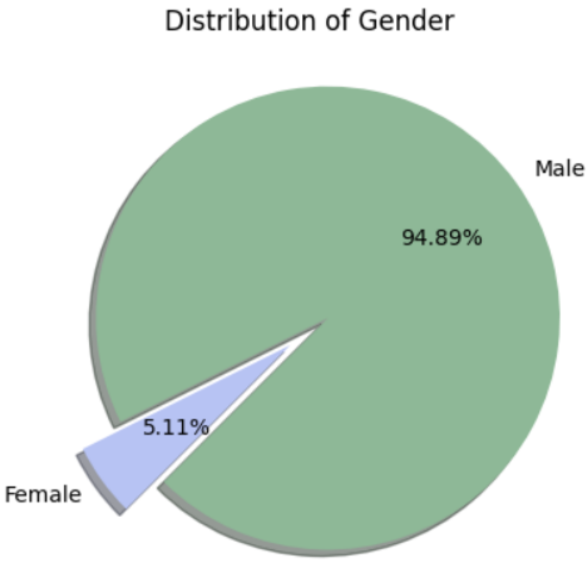
- Country 1 has reported almost 60% of accidents. Probably because there are more cities with industrial plants in this country. This needs to be verified in Bivariate analysis
- Country 2 and Country 3 report approx 30% and 10% accidents respectively

Analysis by City



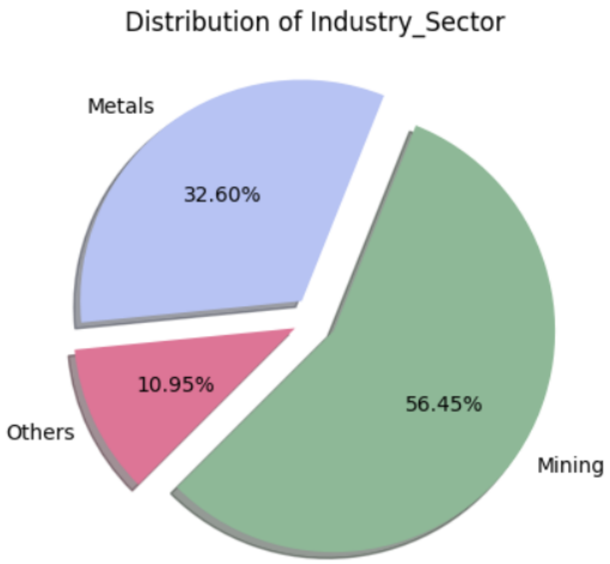
- 21% accidents of total accidents reported from Local 3
- Local 5, Local 4 and Local 1 are next major cities with accidents
- Local 2, Local 9 and Local 11 have reported least number of accidents (less than 2% together)

Analysis by Gender



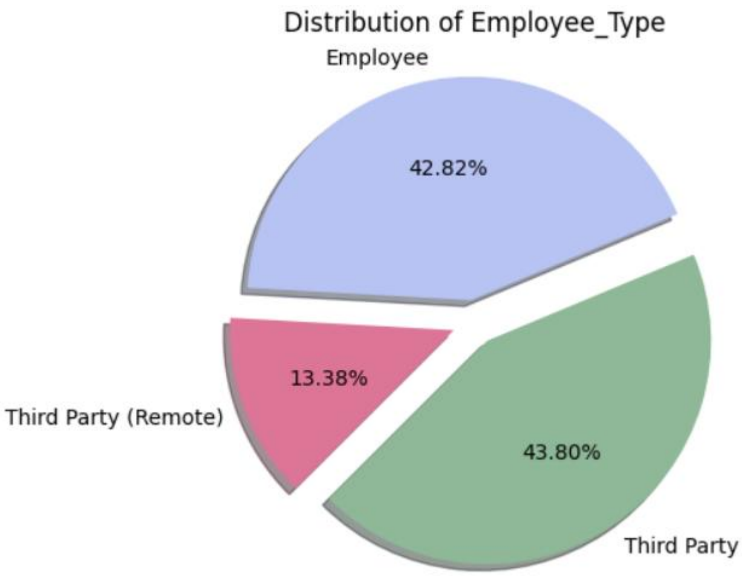
- 95% employees who reported accidents are Male compared to 5% Females

Analysis by Industry Sector



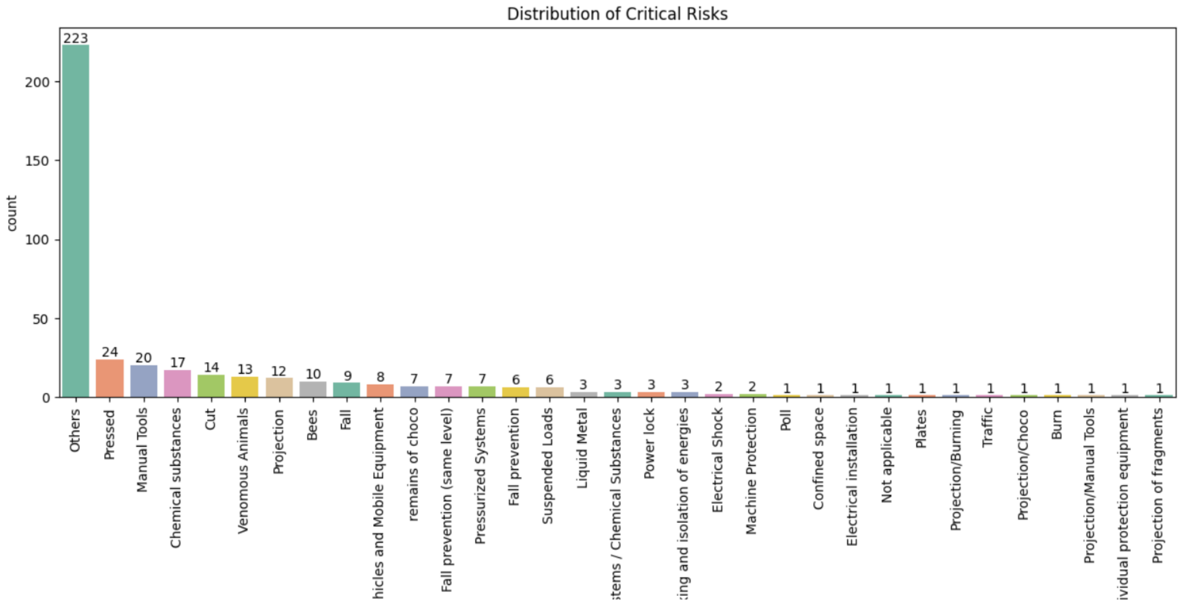
- Mining industry contributes to almost 56% accidents
- Metals and Others industries contribute 33% and 11% accidents (approx)

Analysis by Employee Type



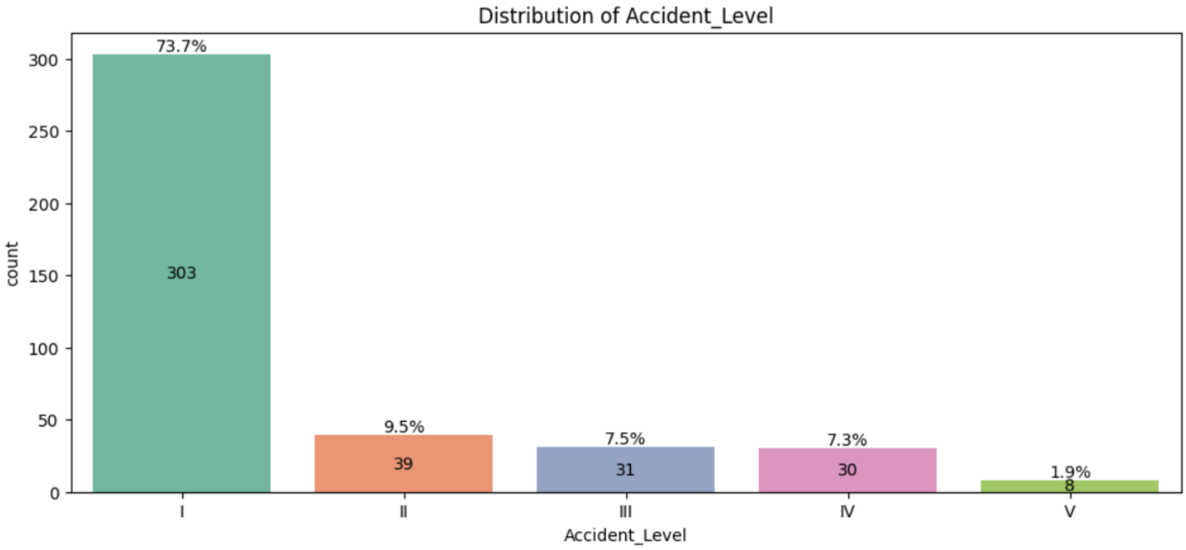
- 44% of accidents are recorded by Third party (possibly contract) employees
- 43% accidents are recorded by Direct employees
- Least accidents (13%) are recorded by Third Party Remote employees, possibly because they are not working on location in the plants

Analysis by Critical Risk



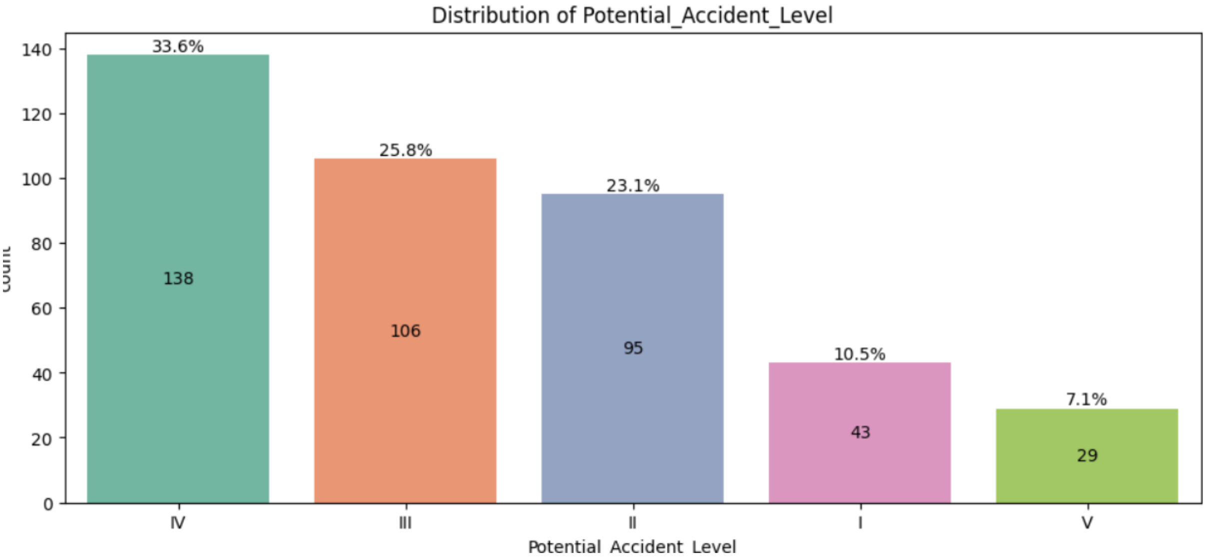
- More than 50% of accidents are categorised as Others.
- The other significant critical risks are Pressed, Manual Tools, Chemical Substances, Cut, Venomous Animals, Projection, Bees and Fall

Analysis by Accident Level



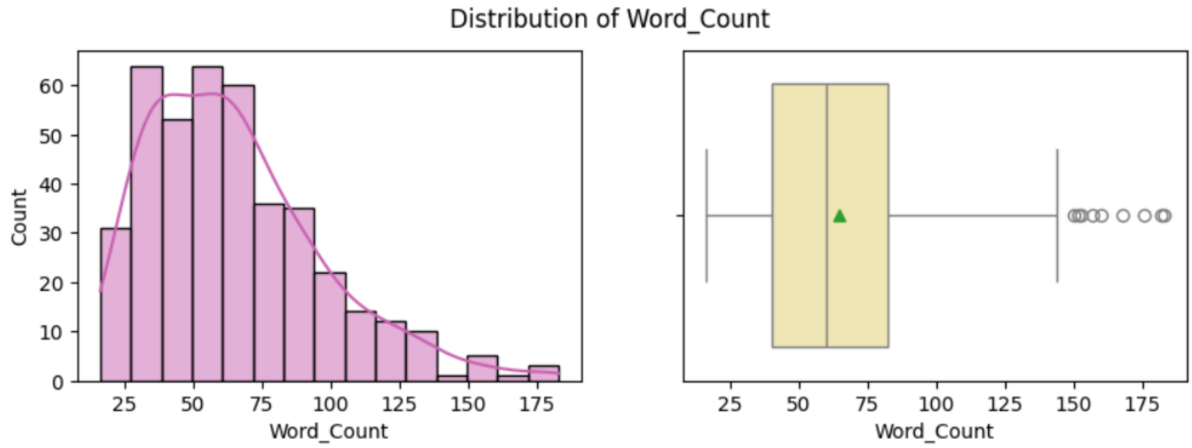
- 74% of the accidents are categorised as Level I accidents indicating a low severity
- 2% are classified as Level V (most severe)

Analysis by Potential Accident Level



- There is a mismatch observed between Accident Level and Potential Accident Level
- Most of the accidents categorised as Level I Accident are of higher severity, this can be seen from the reduced number of Level I accidents according to Potential Accident Level
- This indicates that Potential Accident Level will indicate a better categorisation of accidents as compared to Accident Level and hence a better target for predicting acccident severity

Analysis by Word Count



- Distribution of Word Count is right skewed, indicating most of the descriptions contain fewer than 75 words
- 50% of the descriptions are around 60 words or lesser
- Few descriptions contain more than 150 words

Word clouds

Word Cloud for Description

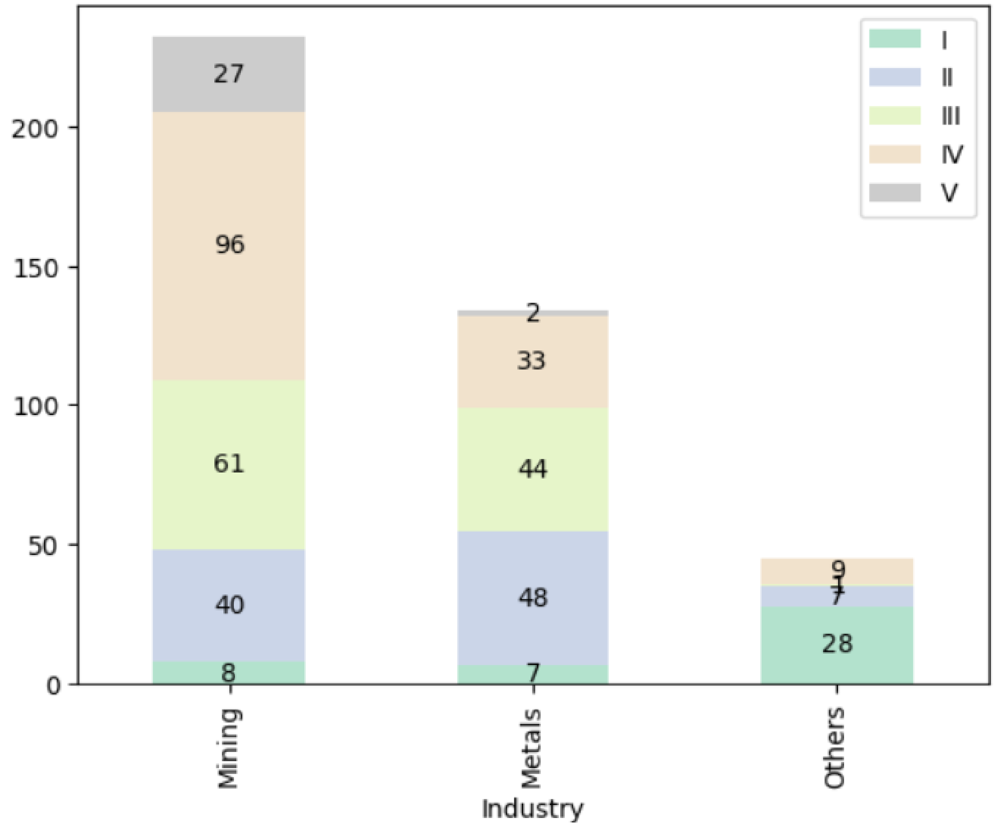
Word Cloud for Nouns used in Description

Word Cloud for Verbs used in Description

- Most of the descriptions contain nouns like operator, employee - people involved in the accident
- Most of the descriptions contain verbs like hit, carry, injured, fell, drill which is most likely the action performed when the accident occurred

Multivariate Analysis

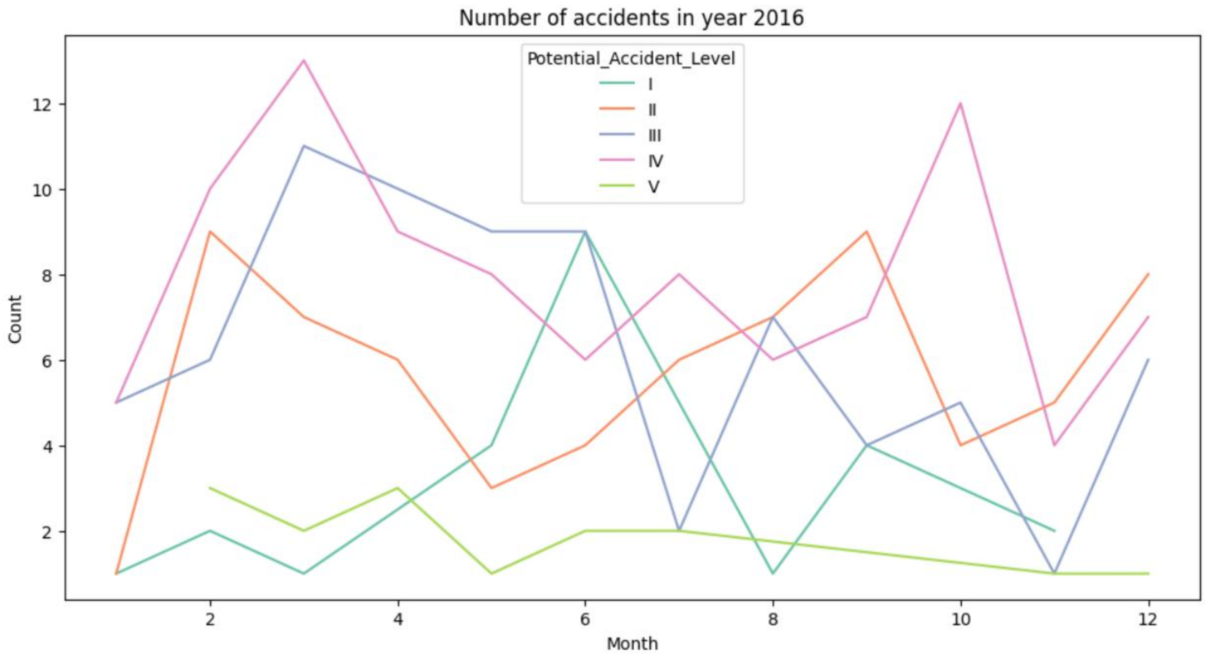
Analysis of industry Sector vs Potential Accident Level

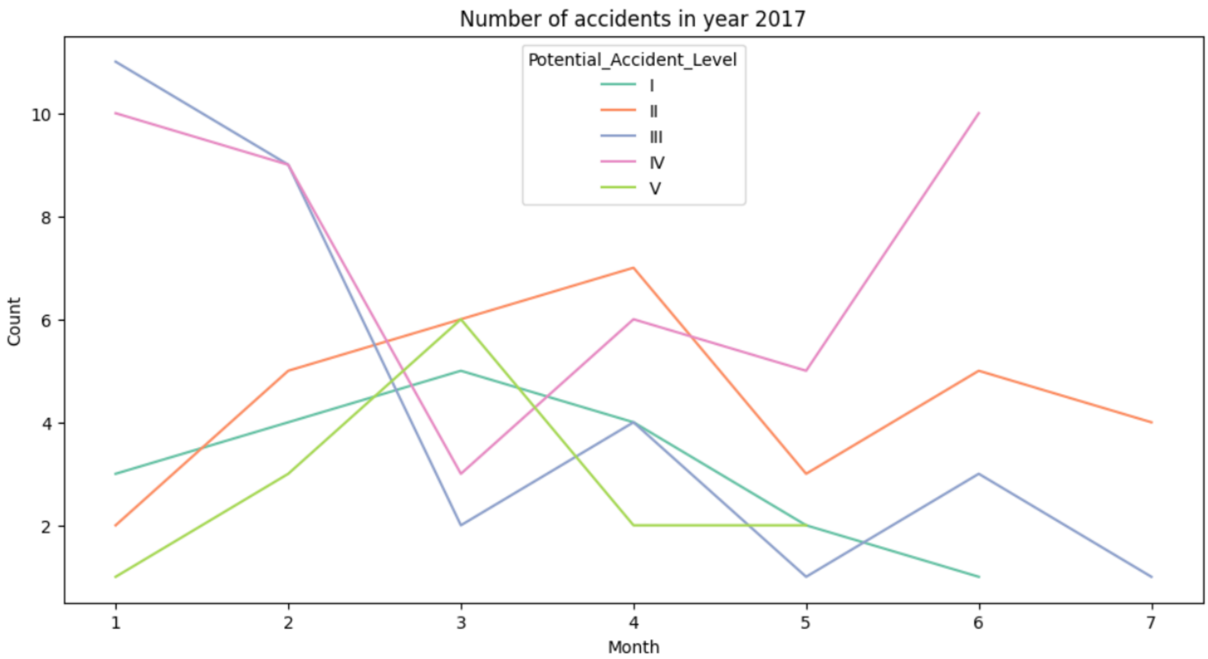


- The most severe accidents (Level V,4,3) are seen only in Mining & Metals industry sectors, majority in Mining and 1 in 'Others' industry
- The 'Others' industry sector has least accidents, with majority being the least severe ones (Level I).
- Accident Levels II accidents have occurred more in Mining and Metals

Analysis by Time period of accidents

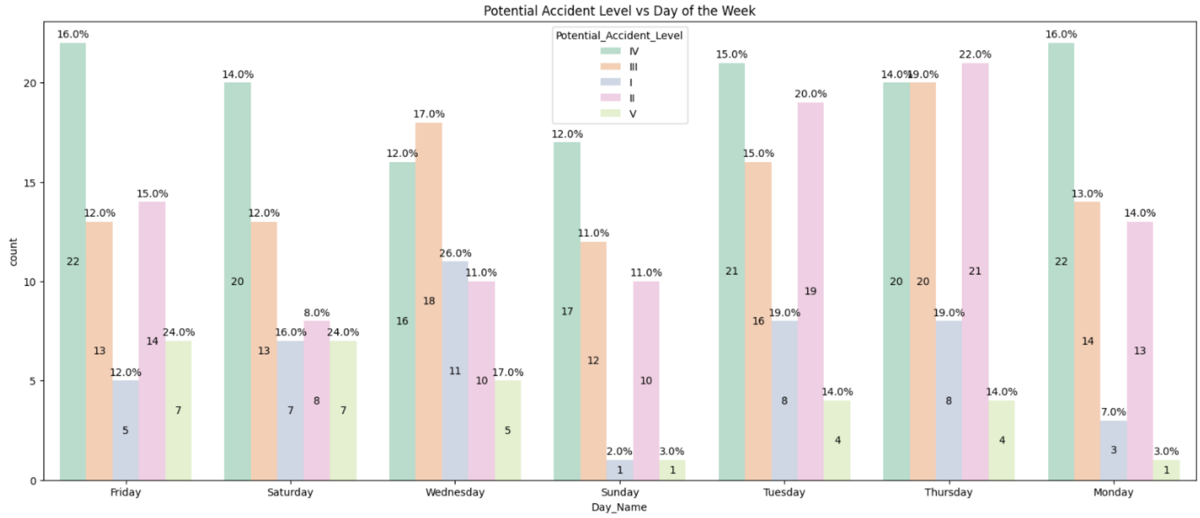
Since the data contained accident details for years 2016 and 2017, we plotted the temporal distribution separated by Potential Accident Level





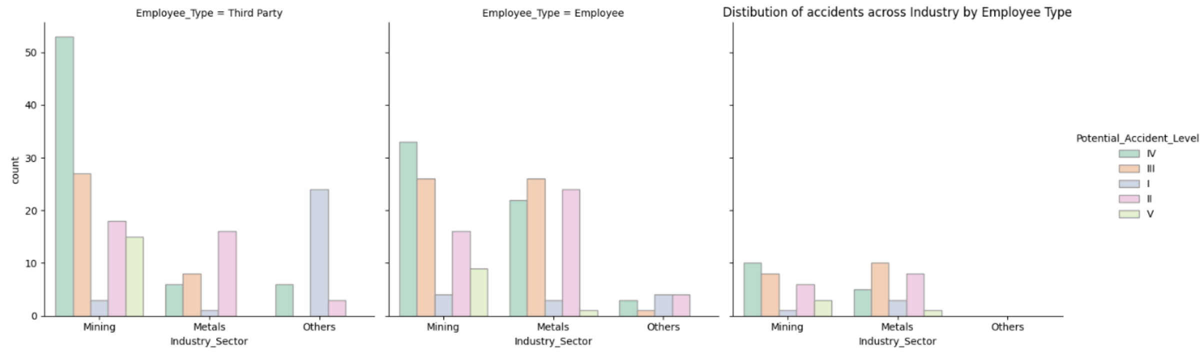
- Accident details are recorded for January 2016 to July 2017
- There is a decreasing trend in accidents for year 2017 as compared to year 2016
- Severe accidents are observed across different months in each year
- Most severe accidents (Level V) were observed in February and April in 2016 while in March for year 2017

Analysis of Potential Accident Level by Day of the week



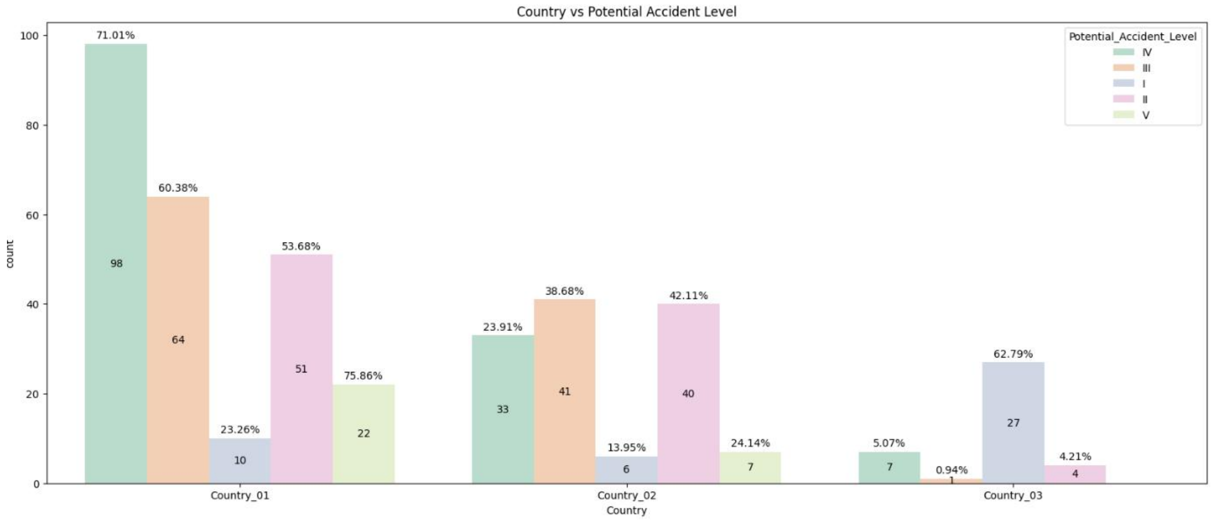
- Accidents are more prone to occur on Thursday and Friday
- Most severe (Level V) accidents are prone to occur on Friday or Saturday
- Least severe (Level I) accidents may occur on any day as per data given its distribution

Analysis of Potential Accident Level by Employee Type across Industry sectors



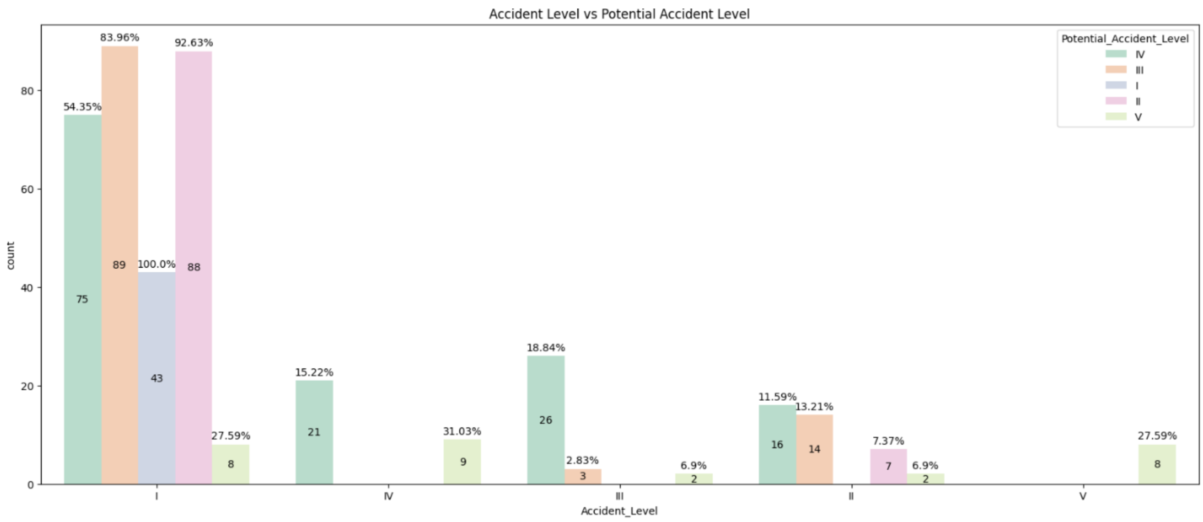
- Third Party employees reported many accidents compared to other employee types across all industries
- Others industries do not have any accidents reported by Remote Third-Party employees
- Most severe accidents are also reported by Third party employees

Analysis of Potential Accident Level by Country



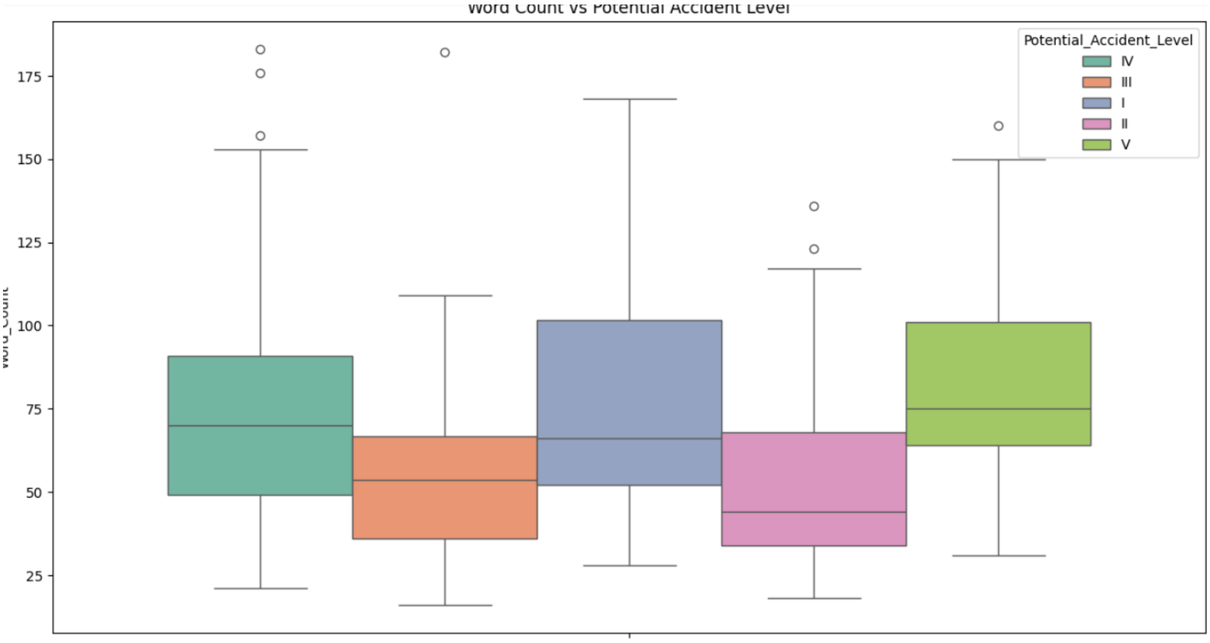
- Country 1 has reported most accidents with majority of Level IV accidents
- Country 3 has reported the least number of accidents of which most are low severity accidents
- Most severe accidents (Levels IV and V) are reported in Country 1 and Country 2

Analysis of Potential Accident Level by Accident Level



- Accident Level tends to categorise accidents on lower severity than Potential Accident Level
- Most of the accidents reported as Level I Accident are actually of a higher severity
- 92% of Potential Accident Level II are categorised as Level I accidents
- 84% of Potential Accident Level III are categorised as Level I accidents
- Only 8 accidents were categorised with highest severity in both Accident Level and Potential Accident Level

Analysis of Potential Accident Level by Word Count



- Description of Levels I, IV and V in general longer than those of Levels II and III
- Longest description has around 200 words and belongs to Potential Accident Levels III and IV

Model Building Experiment

Approach

The aim of the project is to predict the accident severity based on its description so that we can utilise NLP techniques and build a chatbot that can help in increasing safety pf industrial workers.

Key points

We will solve this problem using Traditional ML techniques.

This can be categorised as a multiclass classification to determine accident severity (Level I to Level V).

Based on the EDA, we know Accident Level is highly imbalanced (74% of accidents are classified as Level I) and hence may not capture the actual severity like Potential Accident Level. Hence, we will use Potential Accident Level as target.

Text embeddings:

We will generate embeddings for accident Description (after text preprocessing) using 4 different techniques: TF-IDF, Word2Vec, Glove and Sentence Transformer

ML Models and Model Training process:

We will explore the below traditional ML models on each of the above embeddings one at a time and compare the performance.

- Logistic Regression
- Decision Tree
- Random Forest
- Support Vector Classifier
- XGBoost Classifier

Each of the above models will be included as a part of Scikit-Learn pipeline.

During experimentation, we found models which include other relevant features in addition to the accident description perform slightly better. So the final training dataset will include the text embeddings along with selected features from the dataset.

The Dropped features include 'Date', 'Day_Name', 'Accident_Level',' Description' and 'Processed_Description'

The best performing 3 models will be selected for Hyper parameter tuning using GridSearchCV. Post tuning, we will record their performance on test set using classification report, metrics and confusion metrics.

Evaluation Metrics:

- Models will be evaluated on Accuracy, Precision, Recall and F1 -Score against Train and Validation data
- Stratified Cross Validation will be performed to record Mean F1-scores across different folds
- The key metric of evaluation will be F1-Score

Snapshot of Data split

```
[ ] #define dataframe containing select columns from orginal data

    feature_df = df.drop(['Date','Day_Name','Accident_Level','Potential_Accident_Level','Description','Processed_Description'],axis=1)
```

```
[ ] feature_df.head(2)
```

	Country	City	Industry_Sector	Gender	Employee_Type	Critical_Risk	Year	Month	Day	Word_Count
0	Country_01	Local_01	Mining	Male	Third Party	Pressed	2016	1	1	80
1	Country_02	Local_02	Mining	Male	Employee	Pressurized Systems	2016	1	2	54

Combine embeddings and other features

```
[ ] X_tfidf = tfidf_df.join(feature_df) #TF-IDF embeddings + relevant features

    #encode the target
    y = df['Potential_Accident_Level']

    encoder = OrdinalEncoder(handle_unknown='use_encoded_value',unknown_value=-1)
    y = encoder.fit_transform(y.values.reshape(-1,1))
```

```
[ ] #Split data into temp and test, further split temp to train and validation
    X_temp, X_test_tfidf, y_temp, y_test_tfidf = train_test_split(X_tfidf,y, random_state = SEED, stratify = y, test_size=TEST_SIZE)
    X_train_tfidf, X_valid_tfidf, y_train_tfidf, y_valid_tfidf = train_test_split(X_temp,y_temp, random_state = SEED,
                                                                    stratify = y_temp, test_size = TEST_SIZE)
```

Model Performance

Model performance on TF-IDF

	Accuracy Train	Precision Train	Recall Train	F1 Score Train	Accuracy Test	Precision Test	Recall Test	F1 Score Test	CV Score Mean
TFIDF Logistic Reg	1.000000	1.000000	1.000000	1.000000	0.348485	0.355556	0.298304	0.310333	0.336414
TFIDF DecisionTree	1.000000	1.000000	1.000000	1.000000	0.272727	0.219372	0.258065	0.229976	0.233550
TFIDF RandomForest	1.000000	1.000000	1.000000	1.000000	0.257576	0.352235	0.207772	0.223620	0.275571
TFIDF SVM	0.866412	0.917886	0.748193	0.780828	0.363636	0.231303	0.238930	0.201176	0.225583
TFIDF Xgboost	0.996183	0.997753	0.997059	0.997389	0.242424	0.205466	0.193705	0.193571	0.190359

Model performance on Word2Vec

	Accuracy Train	Precision Train	Recall Train	F1 Score Train	Accuracy Test	Precision Test	Recall Test	F1 Score Test	CV Score Mean
W2Vec DecisionTree	1.000000	1.000000	1.000000	1.000000	0.409091	0.370909	0.380081	0.367378	0.212932
W2Vec Xgboost	1.000000	1.000000	1.000000	1.000000	0.378788	0.275138	0.278197	0.263717	0.218080
W2Vec Logistic Reg	0.652672	0.711491	0.626925	0.657806	0.318182	0.251483	0.279226	0.258767	0.259717
W2Vec RandomForest	1.000000	1.000000	1.000000	1.000000	0.378788	0.247500	0.262959	0.247094	0.228404
W2Vec SVM	0.366412	0.246592	0.227547	0.163578	0.333333	0.133333	0.202674	0.118824	0.143684

Model performance on Glove

	Accuracy Train	Precision Train	Recall Train	F1 Score Train	Accuracy Test	Precision Test	Recall Test	F1 Score Test	CV Score Mean
Glove RandomForest	1.000000	1.000000	1.000000	1.000000	0.484848	0.427778	0.393522	0.396498	0.281966
Glove Logistic Reg	0.927481	0.946150	0.947447	0.946647	0.378788	0.329470	0.325643	0.326526	0.278342
Glove Xgboost	1.000000	1.000000	1.000000	1.000000	0.378788	0.321508	0.317622	0.306841	0.301549
Glove DecisionTree	1.000000	1.000000	1.000000	1.000000	0.333333	0.283843	0.283402	0.280356	0.243387
Glove SVM	0.553435	0.583214	0.428556	0.444570	0.363636	0.330926	0.241604	0.215775	0.183752

Model performance on Sentence Transformer

	Accuracy Train	Precision Train	Recall Train	F1 Score Train	Accuracy Test	Precision Test	Recall Test	F1 Score Test	CV Score Mean
ST Logistic Reg	1.000000	1.000000	1.000000	1.000000	0.469697	0.415556	0.421523	0.413483	0.364433
ST Xgboost	1.000000	1.000000	1.000000	1.000000	0.409091	0.390870	0.356552	0.360519	0.324161
ST SVM	0.832061	0.918094	0.734113	0.780773	0.424242	0.396364	0.353216	0.342424	0.287357
ST DecisionTree	1.000000	1.000000	1.000000	1.000000	0.378788	0.322381	0.338370	0.328360	0.220041
ST RandomForest	1.000000	1.000000	1.000000	1.000000	0.378788	0.420907	0.309300	0.322527	0.294620

Compare all model performance

Snapshot of best performing models:

	Accuracy Train	Precision Train	Recall Train	F1 Score Train	Accuracy Test	Precision Test	Recall Test	F1 Score Test	CV Score Mean
ST Logistic Reg	1.000000	1.000000	1.000000	1.000000	0.469697	0.415556	0.421523	0.413483	0.364433
Glove RandomForest	1.000000	1.000000	1.000000	1.000000	0.484848	0.427778	0.393522	0.396498	0.281966
ST Xgboost	1.000000	1.000000	1.000000	1.000000	0.409091	0.390870	0.356552	0.360519	0.324161
ST SVM	0.832061	0.918094	0.734113	0.780773	0.424242	0.396364	0.353216	0.342424	0.287357
ST DecisionTree	1.000000	1.000000	1.000000	1.000000	0.378788	0.322381	0.338370	0.328360	0.220041
Glove Logistic Reg	0.927481	0.946150	0.947447	0.946647	0.378788	0.329470	0.325643	0.326526	0.278342
ST RandomForest	1.000000	1.000000	1.000000	1.000000	0.378788	0.420907	0.309300	0.322527	0.294620
W2Vec Xgboost	1.000000	1.000000	1.000000	1.000000	0.393939	0.338629	0.318228	0.312261	0.244017
TFIDF Logistic Reg	1.000000	1.000000	1.000000	1.000000	0.348485	0.355556	0.298304	0.310333	0.336414
Glove Xgboost	1.000000	1.000000	1.000000	1.000000	0.378788	0.321508	0.317622	0.306841	0.301549
Glove DecisionTree	1.000000	1.000000	1.000000	1.000000	0.333333	0.283843	0.283402	0.280356	0.243387
W2Vec DecisionTree	1.000000	1.000000	1.000000	1.000000	0.333333	0.241832	0.250924	0.241103	0.226927
TFIDF DecisionTree	1.000000	1.000000	1.000000	1.000000	0.272727	0.219372	0.258065	0.229976	0.233550

Snapshot of least performing models:

TFIDF DecisionTree	1.000000	1.000000	1.000000	1.000000	0.272727	0.219372	0.258065	0.229976	0.233550
TFIDF RandomForest	1.000000	1.000000	1.000000	1.000000	0.257576	0.352235	0.207772	0.223620	0.275571
Glove SVM	0.553435	0.583214	0.428556	0.444570	0.363636	0.330926	0.241604	0.215775	0.183752
TFIDF SVM	0.866412	0.917886	0.748193	0.780828	0.363636	0.231303	0.238930	0.201176	0.225583
TFIDF Xgboost	0.996183	0.997753	0.997059	0.997389	0.242424	0.205466	0.193705	0.193571	0.190359
W2Vec SVM	0.366412	0.246592	0.227547	0.163578	0.333333	0.133333	0.202674	0.118824	0.143684

Observations:

- Sentence transformer embedding models are the best performing among all embeddings, followed by Glove
- Word2Vec and TF-IDF embeddings are least performing among the embeddings
- Logistic Regression, Random Forest and XGBoost models' performance is better than those of SVM and Decision Tree

- Best performing 3 models are:
    - Logistic Regression on Sentence Transformer
        - ✓ F1- Score (41%) and Mean CV score (36%)
    - Random Forest on Glove
        - ✓ F1- Score (39%) and Mean CV score (28%)
    - XGBoost on Sentence Transformer
        - ✓ F1- Score (36%) and Mean CV score (32%)

- Least performing models are:
    - Decision Tree on Word2Vec
        - ✓ F1- Score (18%) and Mean CV score (25%)
    - Logistic Regression on Sentence Transformer
        - ✓ F1- Score (11%) and Mean CV score (13%)

Hyperparameter Tuning

The Best performing 3 models are selected for Hyper param tuning using GridSearchCV.

This process was handled using a single method that could fit the pipeline with the given model, tune the parameters and display the metrics before and after tuning

Model performance of Logistic Regression on Sentence Transformer

	Accuracy	Precision	Recall	F1 Score
ST Logistic Regression Base Train	1.000000	1.000000	1.000000	1.000000
ST Logistic Regression Base Valid	0.469697	0.415556	0.421523	0.413483
ST Logistic Regression Best Train	0.996183	0.997753	0.996721	0.997217
ST Logistic Regression Best Valid	0.469697	0.408333	0.418849	0.409442

Model performance of Random Forest on Glove

	Accuracy	Precision	Recall	F1 Score
Glove RandomForest Base Train	1.000000	1.000000	1.000000	1.000000
Glove RandomForest Base Valid	0.439394	0.375579	0.378813	0.372893
Glove RandomForest Best Train	0.690840	0.601741	0.582583	0.584758
Glove RandomForest Best Valid	0.348485	0.268328	0.263153	0.257907

Model performance of XGBoost on Sentence Transformer

	Accuracy	Precision	Recall	F1 Score
ST XGBoost Base Train	1.000000	1.000000	1.000000	1.000000
ST XGBoost Base Valid	0.409091	0.390870	0.356552	0.360519
ST XGBoost Best Train	0.961832	0.976283	0.949013	0.961573
ST XGBoost Best Valid	0.318182	0.316667	0.264451	0.274190

Observations:

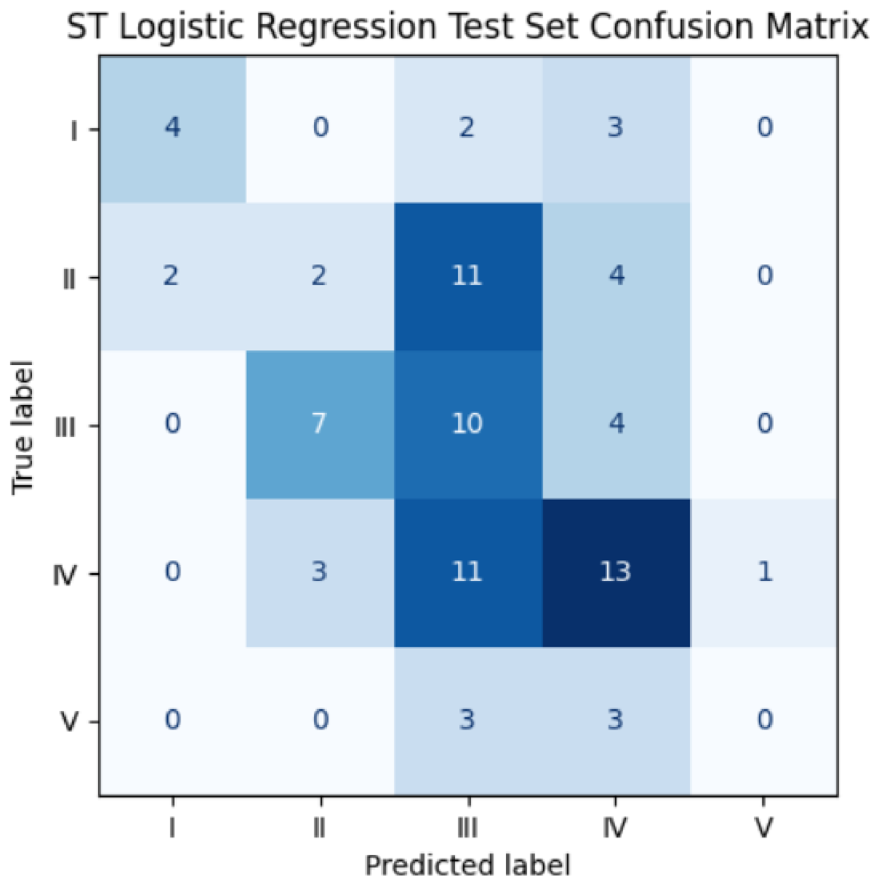
- Logistic Regression on Sentence Transformer embedding is still overfit after tuning
- XGBoost on ST performs better than Random Forest on Glove

Compare Model Performance post tuning

We will compare performance of each of the 3 top models after tuning using GridSearch

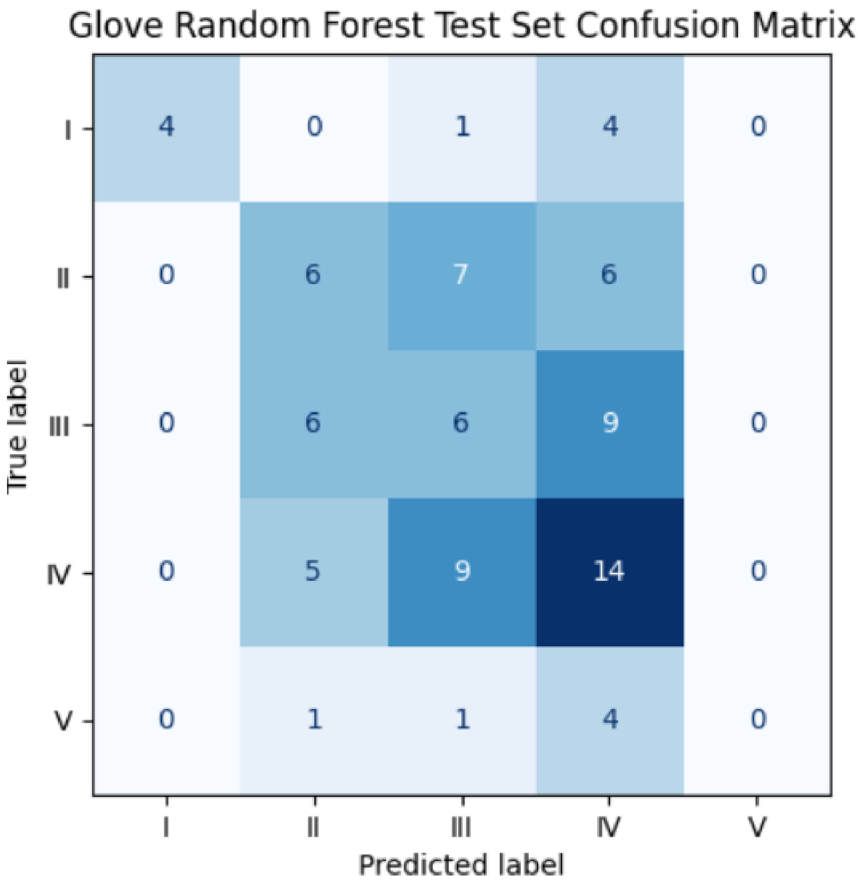
Model performance of Logistic Regression on Sentence Transformer

```
**********************************************************************

ST Logistic Regression Test Set Classification Report
**********************************************************************
              precision    recall  f1-score   support

         0.0       0.67      0.44      0.53         9
         1.0       0.17      0.11      0.13        19
         2.0       0.27      0.48      0.34        21
         3.0       0.48      0.46      0.47        28
         4.0       0.00      0.00      0.00         6

    accuracy                           0.35        83
   macro avg       0.32      0.30      0.30        83
weighted avg       0.34      0.35      0.33        83
```



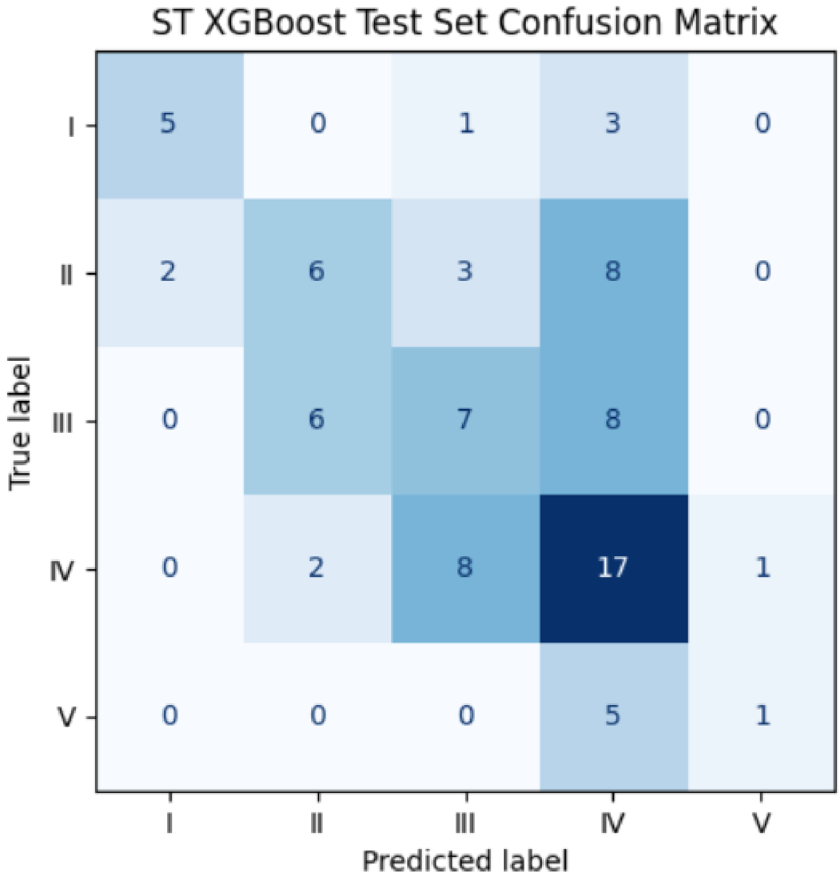
Model performance of Random Forest on Glove

```
**********************************************************************

Glove Random Forest Test Set Classification Report
**********************************************************************
              precision    recall  f1-score   support

         0.0       1.00      0.44      0.62         9
         1.0       0.33      0.32      0.32        19
         2.0       0.25      0.29      0.27        21
         3.0       0.38      0.50      0.43        28
         4.0       0.00      0.00      0.00         6

    accuracy                           0.36        83
   macro avg       0.39      0.31      0.33        83
weighted avg       0.38      0.36      0.35        83
```



Model performance of XGBoost on Sentence Transformer

```
************************************************************************

ST XGBoost Test Set Classification Report
************************************************************************
              precision    recall  f1-score   support

         0.0       0.71      0.56      0.63         9
         1.0       0.43      0.32      0.36        19
         2.0       0.37      0.33      0.35        21
         3.0       0.41      0.61      0.49        28
         4.0       0.50      0.17      0.25         6

    accuracy                           0.43        83
   macro avg       0.49      0.40      0.42        83
weighted avg       0.44      0.43      0.42        83
```



Conclusion from Milestone 1 tasks

```
**********************************************************************
Summary of performance after hyper param tuning
**********************************************************************
```

	Accuracy	Precision	Recall	F1 Score
ST Logistic Regression	0.349398	0.317017	0.298037	0.295984
Glove Random Forest	0.361446	0.392342	0.309190	0.327429
ST XGBoost	0.433735	0.485182	0.395698	0.416278

- XGBoost on Sentence Transformer embedding is the best performing model after hyper param tuning with F1 Score is 41% on the test set
- Random Forest on Glove embedding is next with F1 Score of 32%
- Least performing model is Logistic Regression on Sentence Transformer embedding with F1 Score of 29%
- Mis-classification is highest for Potential Accident Levels III and IV among all models but mostly in Logistic Regression on Sentence Transformer embedding
- Results can be improved by obtaining more training data (if feasible) and including advanced techniques like deep learning which can interpret the contextual information in accident description in a better way
- Another option is to use SMOTE to handle data imbalance or use data augmentation (NLP augmentation) to increase model performance

MILESTONE 2

Summary of problem statement, data and findings

Problem Statement:

The main objective of this project is to build a Chatbot utility that can classify the accident severity level based on the accident description. Please refer to the beginning of this document for a detailed problem statement.

Summary of Data

- The initial dataset consisted of 425 rows and 11 columns
- After data cleaning (removing duplicate rows, duplicate descriptions, unused columns), the dataset contained 411 rows and 10 columns
- Text preprocessing steps were applied on Accident Description and cleaned description is stored in Processed_Description column
- The cleaned dataset was stored in a CSV file

For detailed description of features refer Data Description section

Findings:

- The dataset contains accident reports from 3 countries with Country 1 accounting for 60% of accidents. More data is needed to identify if this is due to lack of reporting mechanism in other countries or Country 1 being a larger country with many industrial plants
- The accidents are recorded from January 2016 to July 2017. So annual trends cannot be captured as this timeline is too narrow to capture seasonal patterns if any
- 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
- The target Potential Accident Level contains severity levels I, II, II, IV or V.
  It is slightly imbalanced with most accidents belonging to Level IV

Overview of the final process

In Milestone 1, we have applied various traditional ML algorithms on different text embeddings and compared the model performance metrics.

Since the target Potential Accident Level is slightly imbalanced, F1 Score will be the deciding metric.

Based on the model performance comparison in Milestone 1, we have identified **XGBoost on Sentence Transformer embedding** as the best performing model after hyper param tuning with F1 Score is **41%** on the test set. This will be considered our baseline.

As part of milestone 2, we will explore advanced methods to predict the target Potential Accident Level using the accident description to identify a model that can exceed the baseline performance. We will then use the best performing model to build a Chatbot Utility.

The solution steps are classified in below categories:

- Traditional ML: Performance improvement techniques
- Neural network models
- Hugging Face Transformers
- Chatbot Utility

The detailed steps for each category are explained in below sections

Performance Improvement techniques on classic ML algorithms

The target Potential Accident Level is imbalanced. We will apply below techniques and compare against original data to see if there is any improvement in model performance

- **NLP Augmentation**: Providing synonymous accident descriptions
- **SMOTE** - Synthetic Minority Oversampling Technique: adding additional rows to handle class imbalance.

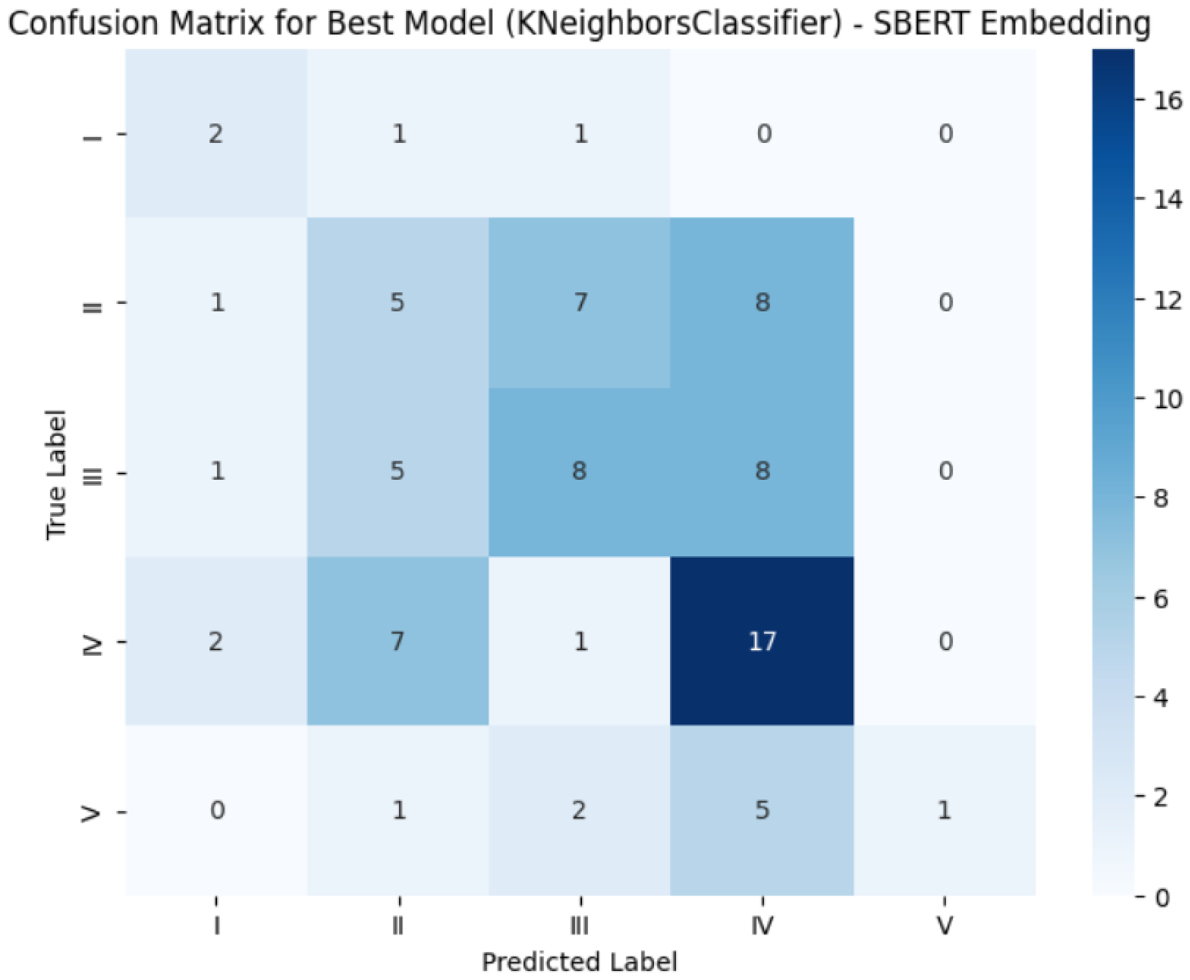
Below ML algorithms will be considered for comparison:

- K-Nearest Neighbour
- Random Forest
- Decision Tree
- Support Vector Machine (SVM)
- Bagging Classifier
- XGBoost

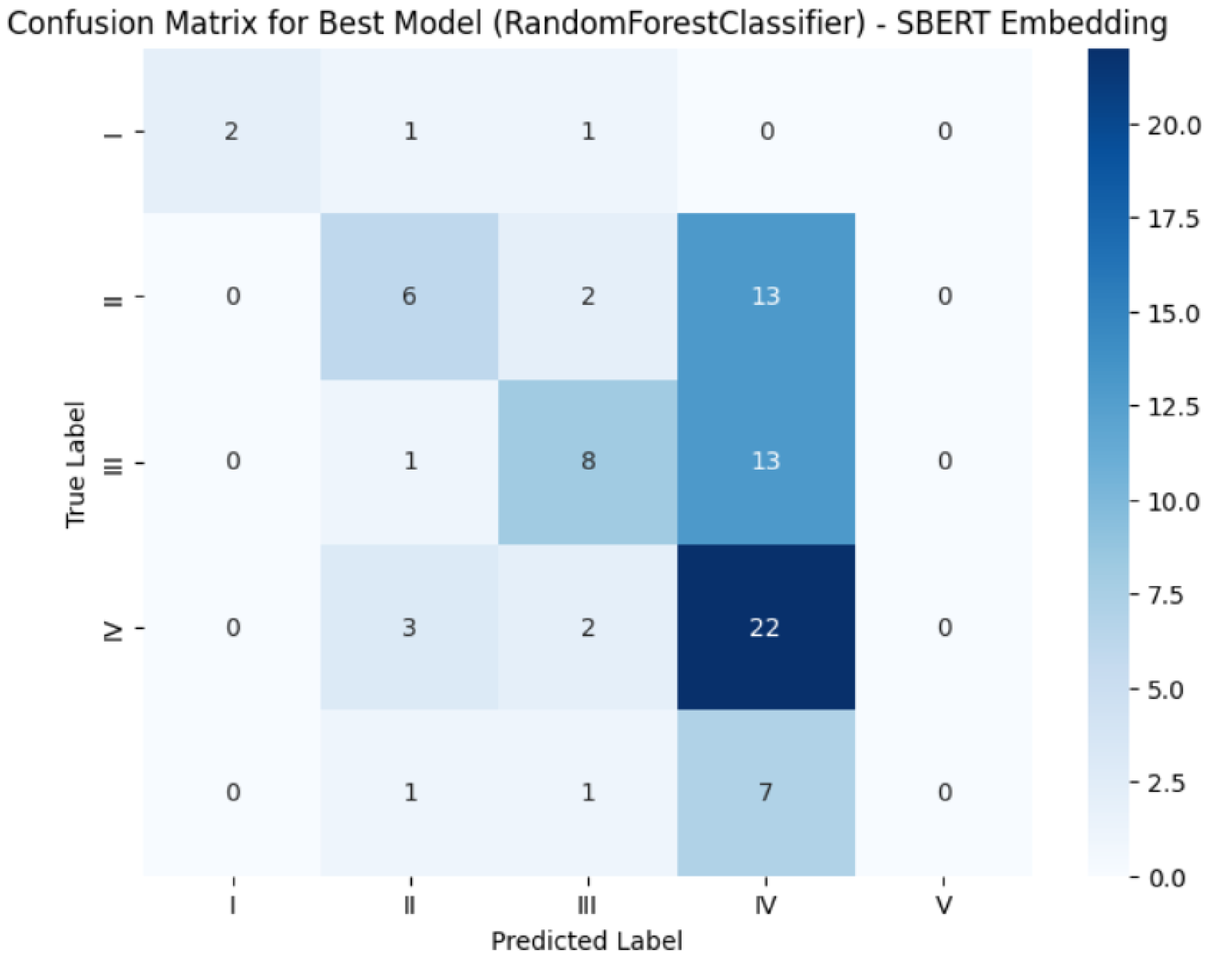
We will generate embeddings using Sentence Transformer and apply the above ML algorithms on original data, NLP augmented data and resampled SMOTE data.

Then we will compare the performance of all models across different strategies.

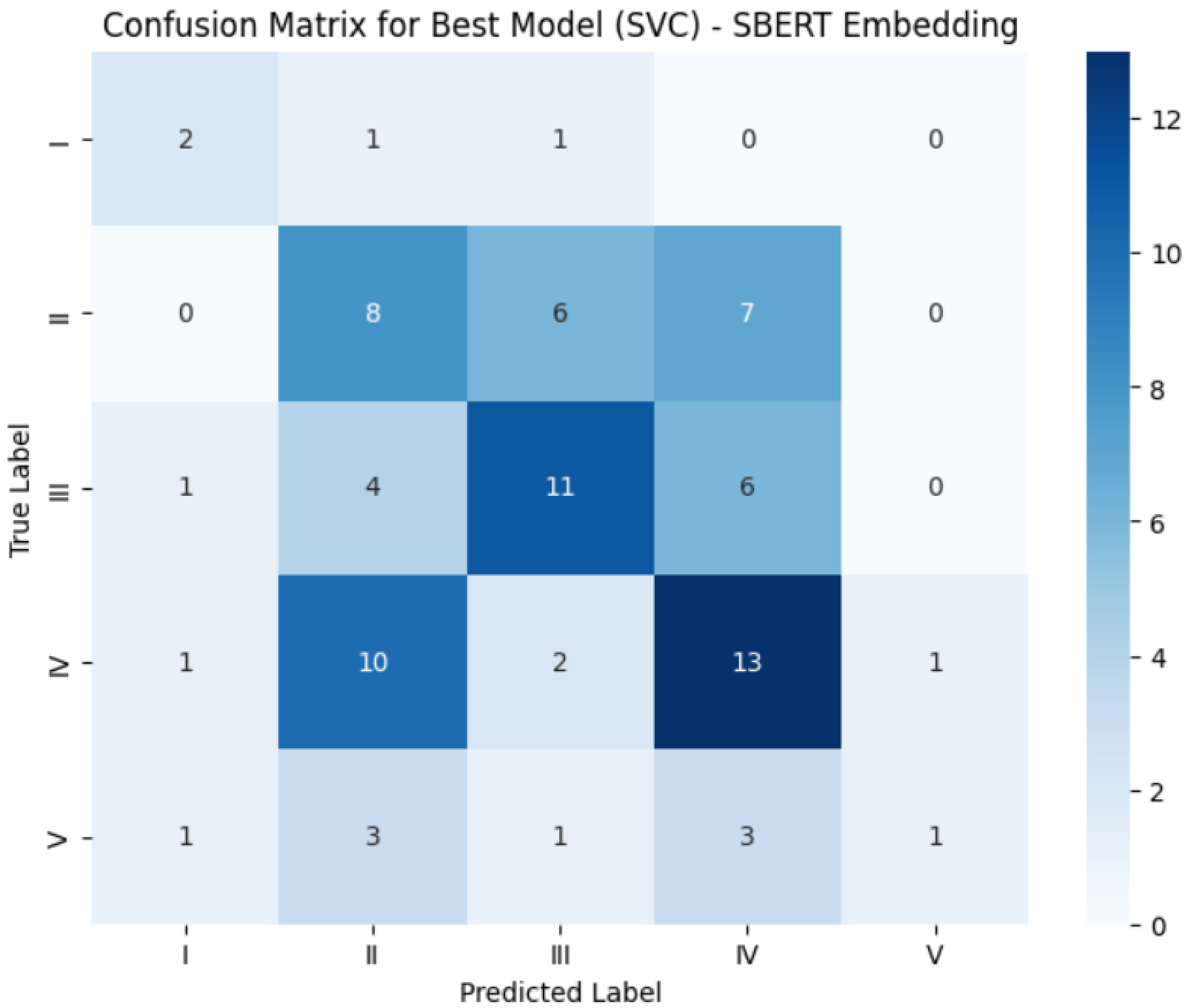
SBERT Base Best Model Confusion Matrix



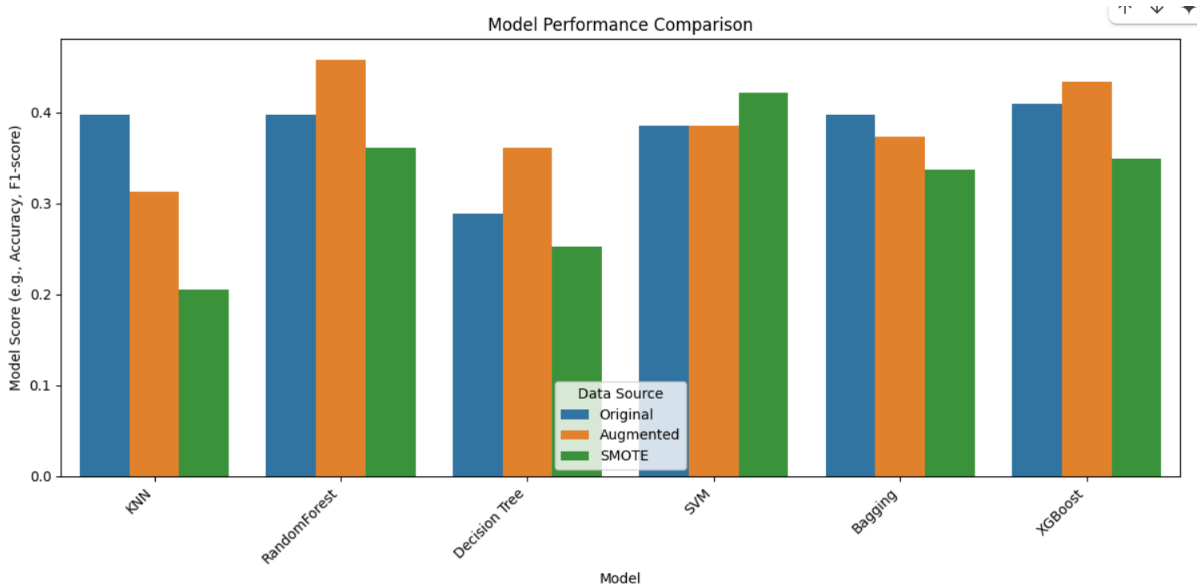
SBERT with NLP Aug: Best Model Confusion Matrix



SBERT with SMOTE: Best Model Confusion Matrix



Models comparison



Observation

- Different models show varying performance across the 3 methods. No decisive pattern is observed.
- Best performing models in each category:
    - Original: XGBoost
    - NLP Aug: Random Forest
    - SMOTE: SVM
- Worst performing models in each category:
    - Original: Decision Tree
    - NLP Aug: KNN
    - SMOTE: KNN
- Although some models are exceeding baseline F1 Score of 41%, there seems to be scope for improvement

Neural networks

In this section, we will predict Potential Accident Level using below 3 types of basic Neural Network architectures.

- Fully connected neural network
- Bi-directional Recurrent Neural Network (Bidirectional RNN)
- Long-Short Term Memory (LSTM)

We will implement the Neural Network using PyTorch and use the cleaned, preprocessed Accident Description in Sentence Transformer embeddings as input to each NN.

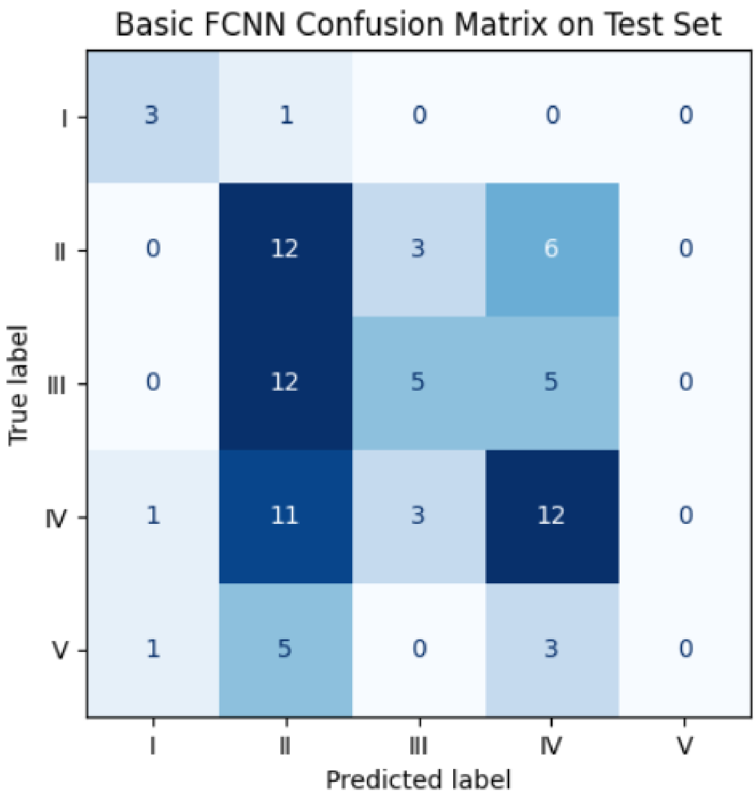
Note: Sentence Transformer embeddings are used as input since these were the best performing embeddings as identified in Milestone 1

Fully Connected Neural Network

We will build a fully connected NN with 2 hidden layers with ReLU activation and a Softmax activation on the output layer.

Note: All models in this section are trained with a default Learning Rate = 0.1, Adam Optimiser for 30 epochs

```
************************************************************************
Basic FCNN Classification Report on Test Set
************************************************************************
              precision    recall  f1-score   support

           I       0.60      0.75      0.67         4
          II       0.29      0.57      0.39        21
         III       0.45      0.23      0.30        22
          IV       0.46      0.44      0.45        27
           V       0.00      0.00      0.00         9

    accuracy                           0.39        83
   macro avg       0.36      0.40      0.36        83
weighted avg       0.37      0.39      0.36        83

************************************************************************
```



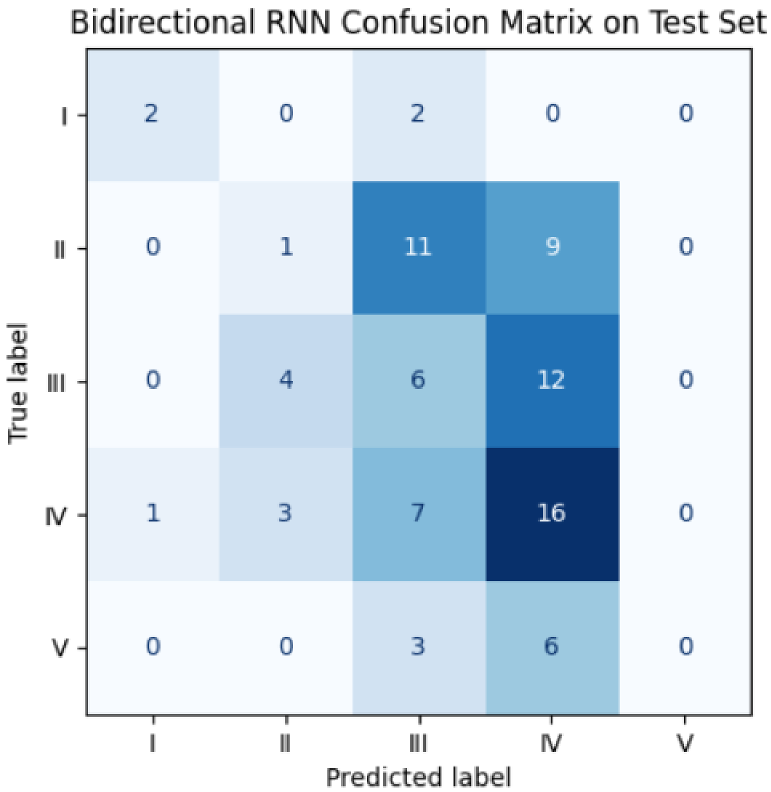
Bidirectional RNN

Recurrent Neural Network(RNN) is a type of neural network designed for sequential data, where outputs from previous steps are fed as inputs to the current step.

A bidirectional RNN can process sequences in both forward and backward directions.

We will build a NN consisting of 2 bidirectional RNN layers

```
****************************************************************
Bidirectional RNN Classification Report on Test Set
****************************************************************
              precision    recall  f1-score   support

           I       0.67      0.50      0.57         4
          II       0.12      0.05      0.07        21
         III       0.21      0.27      0.24        22
          IV       0.37      0.59      0.46        27
           V       0.00      0.00      0.00         9

    accuracy                           0.30        83
   macro avg       0.27      0.28      0.27        83
weighted avg       0.24      0.30      0.26        83

****************************************************************
```

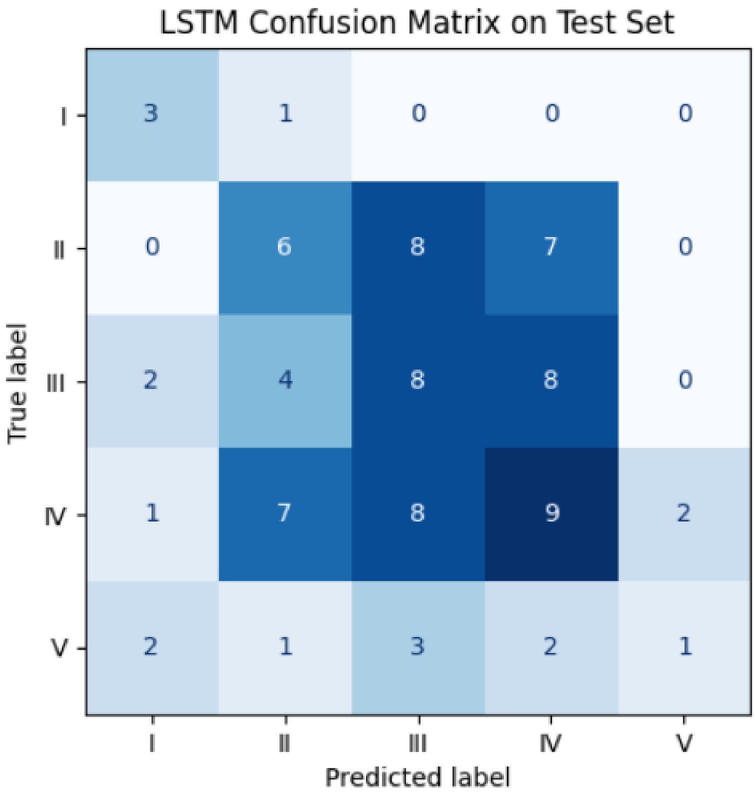


LSTM

Long Short-Term Memory (LSTM) is a special type of RNN designed to address vanishing/exploding gradient problems. It can capture long term dependencies in sequential data.

We will build a NN with 2 LSTM layers

```
***********************************************************
LSTM Classification Report on Test Set
***********************************************************
              precision    recall  f1-score   support

           I       0.38      0.75      0.50         4
          II       0.32      0.29      0.30        21
         III       0.30      0.36      0.33        22
          IV       0.35      0.33      0.34        27
           V       0.33      0.11      0.17         9

    accuracy                           0.33        83
   macro avg       0.33      0.37      0.33        83
weighted avg       0.33      0.33      0.32        83

***********************************************************
```



NN Models Comparison

	Accuracy	F1	Precision	Recall
Basic FCNN	0.385542	0.357696	0.373589	0.385542
LSTM	0.325301	0.315102	0.325256	0.325301
Bidirectional RNN	0.301205	0.256064	0.239637	0.301205

Observation:

- Fully connected NN is the top performing neural network
- It has an Accuracy of 38% and F1 score of 35%.

- LSTM is next best with accuracy of 32% and F1 of 31%
- Bidirectional RNN is least performing model with accuracy of 30% and F1 of 25%
- All are have poor F1 metrics as compared to the baseline model

Hugging Face Transformers

We will load below transformers from Hugging Face and finetune against the accident dataset to classify the Potential Accident Level.

- Deberta
- Distilbert
- XLNet
- Roberta
- Llama

Key terms/concepts:

Transformers:

A deep learning architecture based on self-attention mechanisms, introduced in the paper "Attention is All You Need." It Captures long-range dependencies effectively. It consists of encoder-decoder or encoder-only/decoder-only structures. It is used in Natural language processing (NLP) tasks like translation, text generation, and classification.

Hugging Face Hub:

It is a platform for sharing, discovering, and using pre-trained models, datasets, and demos. It hosts thousands of open-source models (e.g., transformers, diffusers). It provides tools for fine-tuning, inference, and collaboration. It integrates with the transformers library for easy model loading and deployment.

Quantisation:

Most transformers are huge in size and have billions of parameters. Finetuning them requires a high GPU capacity and availability. To overcome this, we can use quantisation.

It is a technique to reduce the precision of model weights and activations (e.g., from 32-bit to 8-bit or 4-bit). It reduces memory usage and computational cost, enabling faster inference and deployment on resource-constrained devices.

BitsandBytes library:

It is a library for efficient 8-bit and 4-bit quantization of deep learning models. This will help in finetuning huge transformers (LLMs) even with limited GPU resources

LoRA:

LoRA (Low-Rank Adaptation) is a PEFT (Parameter-Efficient Fine-Tuning) technique to fine-tune large models by updating only a small subset of parameters. This enables us to train large pre-trained models to new tasks with minimal additional parameters. Advantage: Maintains most of the pre-trained weights frozen, reducing memory and compute requirements.

Weights and Biases(wandb):

It is a library that can track and visualise the training process of machine learning models.

Fine-tuning steps:

Below are the general steps followed for training the transformers against the accident dataset.

- Prepare the train and test dataset with 80/20 split
- Use transformers library to load the model (AutoModelForSequenceClassification or specific variations) and tokenizer (AutoTokenizer or specific variations) from Hugging Face by using the model's name
- For large models, use bitsandbytes library to download 4-bit quantised models
- Setup training arguments by specifying the output directiory, evaluation strategy, number of epochs, logging directory
- Start training, use wandb to track the process, as applicable
- Post training, evaluate the finetuned transformer and document the performance metrics
- Save the finetuned model in Google Drive

Please refer below the different transformers and variations that were used in finetuning experiment. We will look at the evaluation metrics on the test set.

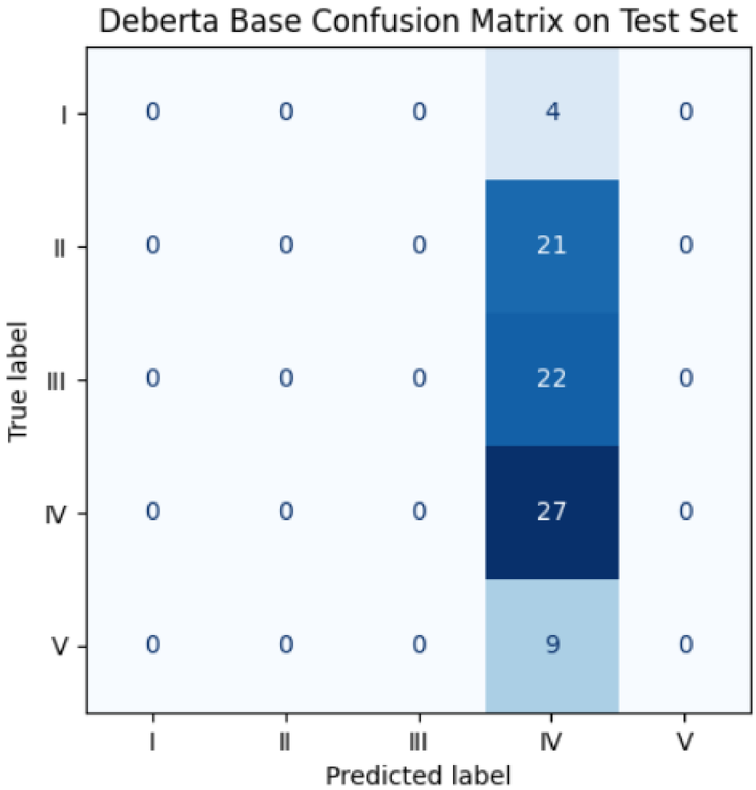
Deberta

In this section, we will finetune the Deberta transformer against Orginal data, with NLP Aug and with SMOTE

Hugging Face Hub Model Name: **microsoft/deberta-v3-base**

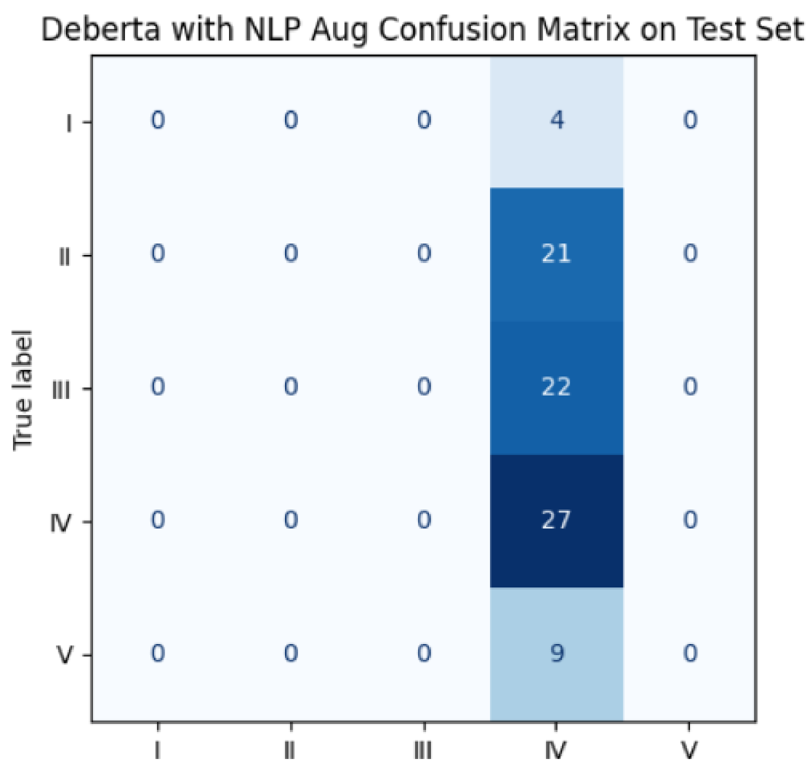
Deberta Base

```
****************************************************************
Deberta Base Classification Report on Test Set
****************************************************************
              precision    recall  f1-score   support

           I     0.0000    0.0000    0.0000         4
          II     0.0000    0.0000    0.0000        21
         III     0.0000    0.0000    0.0000        22
          IV     0.3253    1.0000    0.4909        27
           V     0.0000    0.0000    0.0000         9

    accuracy                         0.3253        83
   macro avg     0.0651    0.2000    0.0982        83
weighted avg     0.1058    0.3253    0.1597        83

****************************************************************
```



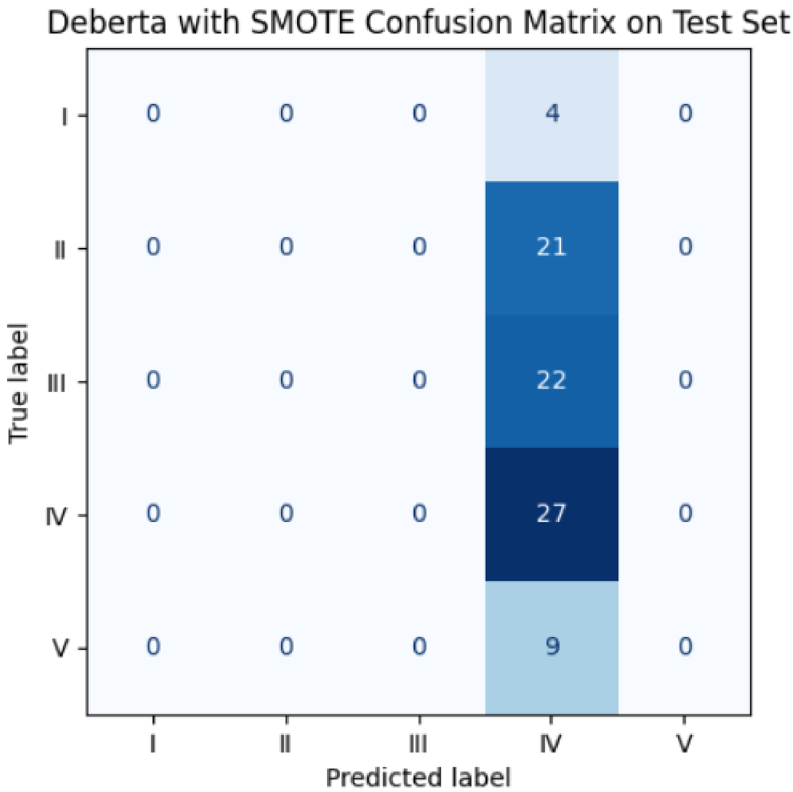
Deberta with NLP Aug

```
*************************************************************
Deberta with NLP Aug Classification Report on Test Set
*************************************************************
              precision    recall  f1-score   support

           I     0.0000    0.0000    0.0000         4
          II     0.0000    0.0000    0.0000        21
         III     0.0000    0.0000    0.0000        22
          IV     0.3253    1.0000    0.4909        27
           V     0.0000    0.0000    0.0000         9

    accuracy                         0.3253        83
   macro avg     0.0651    0.2000    0.0982        83
weighted avg     0.1058    0.3253    0.1597        83

*************************************************************
```



Deberta with SMOTE

```
**********************************************************
Deberta with SMOTE Classification Report on Test Set
**********************************************************
              precision    recall  f1-score   support

           I     0.0000    0.0000    0.0000         4
          II     0.0000    0.0000    0.0000        21
         III     0.0000    0.0000    0.0000        22
          IV     0.3253    1.0000    0.4909        27
           V     0.0000    0.0000    0.0000         9

    accuracy                         0.3253        83
   macro avg     0.0651    0.2000    0.0982        83
weighted avg     0.1058    0.3253    0.1597        83

**********************************************************
```



Deberta Models Comparison

	Accuracy	F1	Precision	Recall
Deberta Base	0.325301	0.159693	0.105821	0.325301
Deberta with NLP Aug	0.325301	0.159693	0.105821	0.325301
Deberta with SMOTE	0.325301	0.159693	0.105821	0.325301

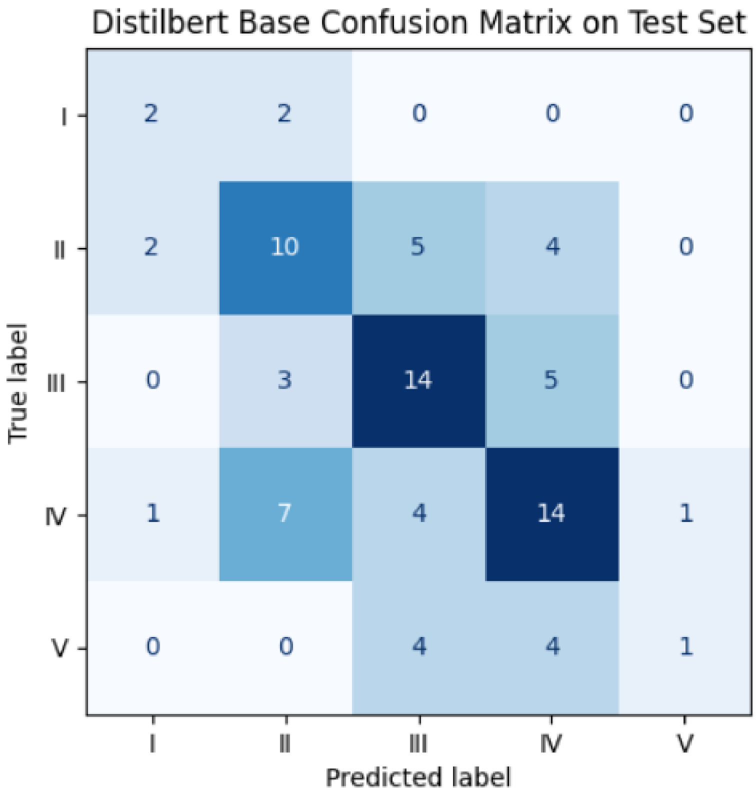
Deberta performance is same across all 3 model strategies indicating the model is unable to learn from the accident data

Accuracy: 32% F1 Score: 16%

Distilbert

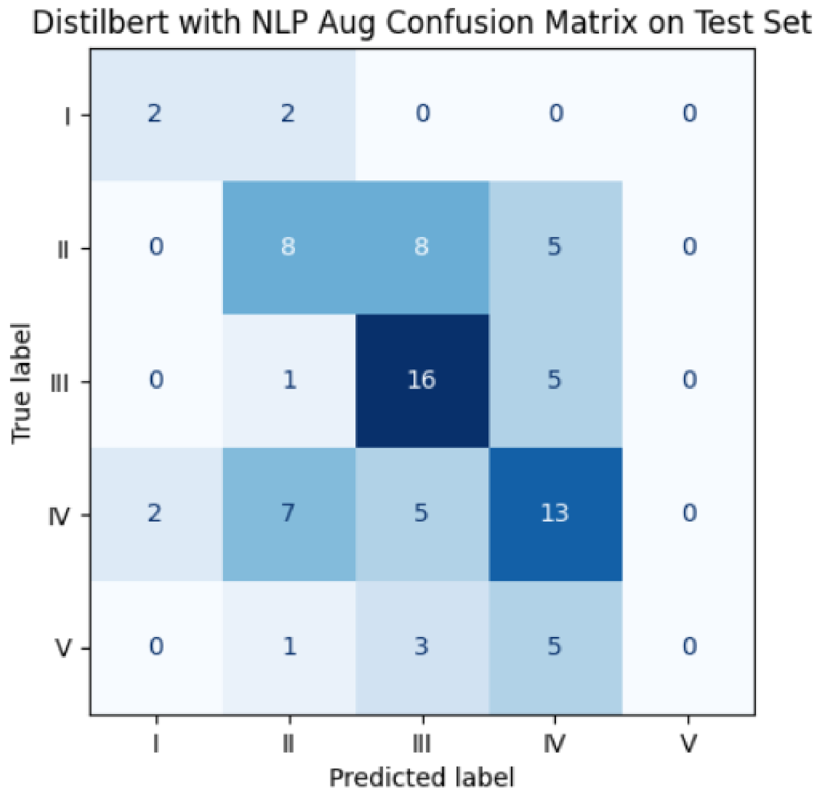
Distilbert Base

```
********************************************************************
Distilbert Base Classification Report on Test Set
********************************************************************
              precision    recall  f1-score   support

           I     0.4000    0.5000    0.4444         4
          II     0.4545    0.4762    0.4651        21
         III     0.5185    0.6364    0.5714        22
          IV     0.5185    0.5185    0.5185        27
           V     0.5000    0.1111    0.1818         9

    accuracy                         0.4940        83
   macro avg     0.4783    0.4484    0.4363        83
weighted avg     0.4946    0.4940    0.4790        83

********************************************************************
```



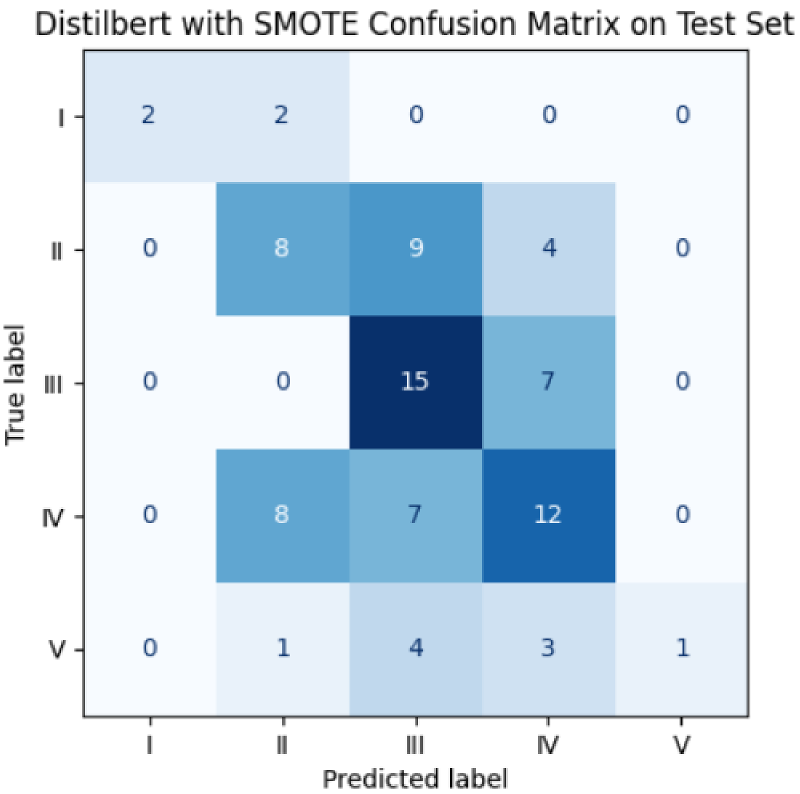
Distilbert with NLP Aug

```
************************************************************
Distilbert with NLP Aug Classification Report on Test Set
************************************************************
              precision    recall  f1-score   support

           I     0.5000    0.5000    0.5000         4
          II     0.4211    0.3810    0.4000        21
         III     0.5000    0.7273    0.5926        22
          IV     0.4643    0.4815    0.4727        27
           V     0.0000    0.0000    0.0000         9

    accuracy                         0.4699        83
   macro avg     0.3771    0.4179    0.3931        83
weighted avg     0.4142    0.4699    0.4362        83

************************************************************
```



Distilbert with SMOTE

```
************************************************************
Distilbert with SMOTE Classification Report on Test Set
************************************************************
              precision    recall  f1-score   support

           I     1.0000    0.5000    0.6667         4
          II     0.4211    0.3810    0.4000        21
         III     0.4286    0.6818    0.5263        22
          IV     0.4615    0.4444    0.4528        27
           V     1.0000    0.1111    0.2000         9

    accuracy                         0.4578        83
   macro avg     0.6622    0.4237    0.4492        83
weighted avg     0.5269    0.4578    0.4418        83

************************************************************
```



DistilBert Models Comparison

	Accuracy	F1	Precision	Recall
Distilbert Base	0.493976	0.478952	0.494613	0.493976
Distilbert with NLP Aug	0.469880	0.436153	0.414191	0.469880
Distilbert with SMOTE	0.457831	0.441832	0.526894	0.457831

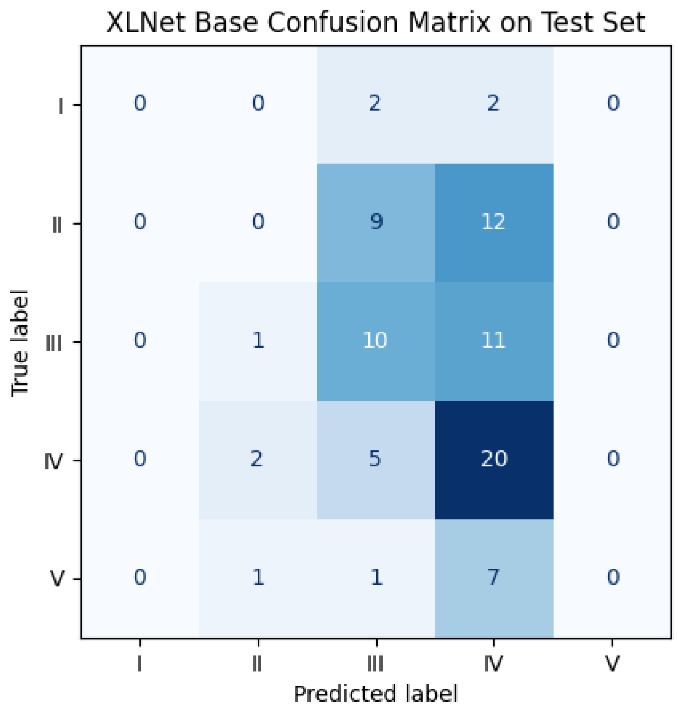
- DistilBert Base has the best performance scores
  - Accuracy: 49% F1 Score: 47%

- DistilBert with NLP Aug is second
    - Accuracy: 47% F1 Score: 43%
- DistilBert with SMOTE has the least performance scores
    - Accuracy: 45% F1 Score: 44%

XLNet

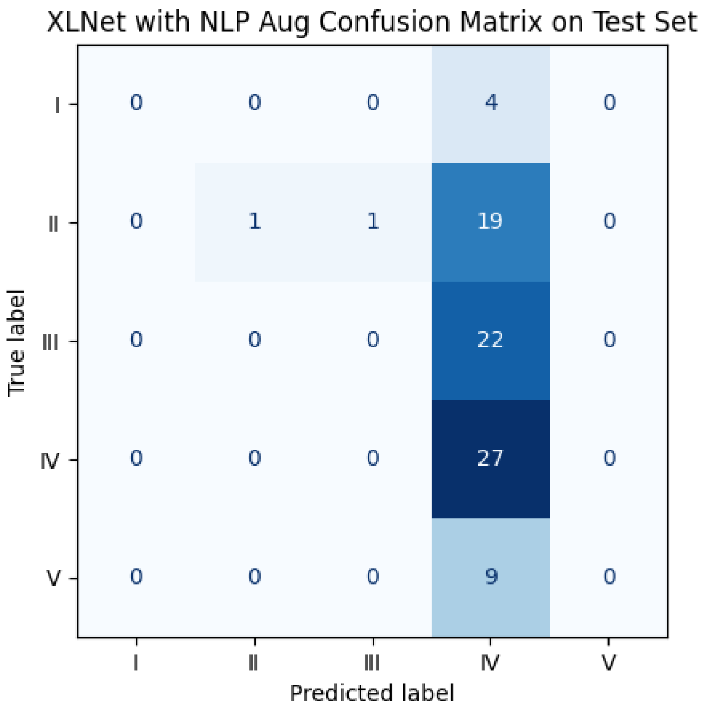
XLNet Base

```
************************************************************
XLNet Base Classification Report on Test Set
************************************************************
              precision    recall  f1-score   support

           I     0.0000    0.0000    0.0000         4
          II     0.0000    0.0000    0.0000        21
         III     0.3704    0.4545    0.4082        22
          IV     0.3846    0.7407    0.5063        27
           V     0.0000    0.0000    0.0000         9

    accuracy                         0.3614        83
   macro avg     0.1510    0.2391    0.1829        83
weighted avg     0.2233    0.3614    0.2729        83

************************************************************
```



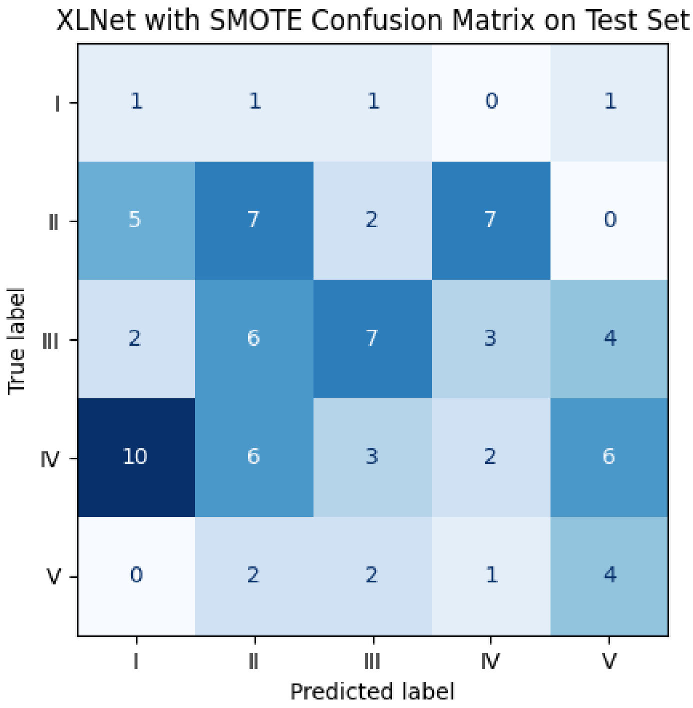
XLNet with NLP Aug

```
*************************************************************
XLNet with NLP Aug Classification Report on Test Set
*************************************************************
              precision    recall  f1-score   support

           I     0.0000    0.0000    0.0000         4
          II     1.0000    0.0476    0.0909        21
         III     0.0000    0.0000    0.0000        22
          IV     0.3333    1.0000    0.5000        27
           V     0.0000    0.0000    0.0000         9

    accuracy                         0.3373        83
   macro avg     0.2667    0.2095    0.1182        83
weighted avg     0.3614    0.3373    0.1857        83

*************************************************************
```



XLNet with SMOTE

```
************************************************************
XLNet with SMOTE Classification Report on Test Set
************************************************************
              precision    recall  f1-score   support

           I     0.0556    0.2500    0.0909         4
          II     0.3182    0.3333    0.3256        21
         III     0.4667    0.3182    0.3784        22
          IV     0.1538    0.0741    0.1000        27
           V     0.2667    0.4444    0.3333         9

    accuracy                         0.2530        83
   macro avg     0.2522    0.2840    0.2456        83
weighted avg     0.2858    0.2530    0.2557        83

************************************************************
```



XLNet Models Comparison

	Accuracy	F1	Precision	Recall
XLNet Base	0.361446	0.272897	0.223286	0.361446
XLNet with NLP Aug	0.337349	0.185652	0.361446	0.337349
XLNet with SMOTE	0.253012	0.255725	0.285838	0.253012

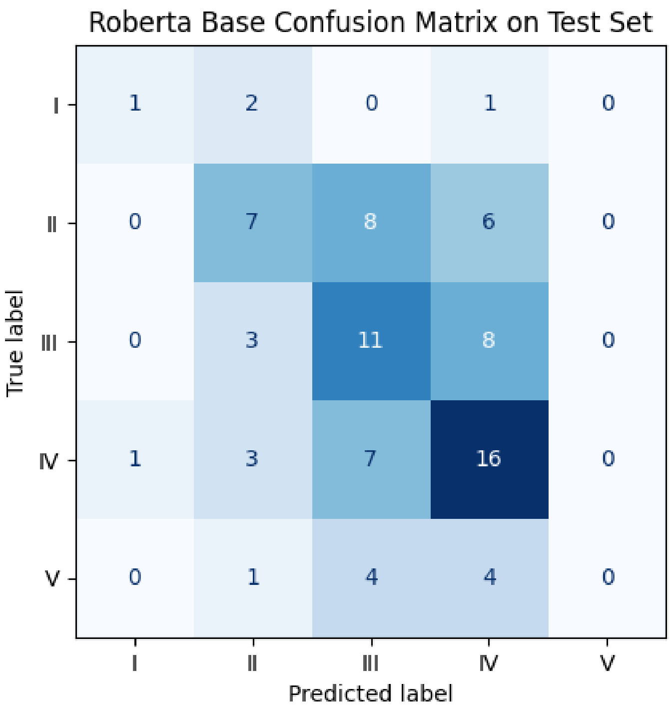
Observation

- XLNet Base has the best performance scores
  - Accuracy: 36% F1 Score: 27%
- XLNet with NLP Aug is second
  - Accuracy: 33% F1 Score: 18%
- XLNet with SMOTE has the least performance scores
  - Accuracy: 25% F1 Score: 25%

Roberta

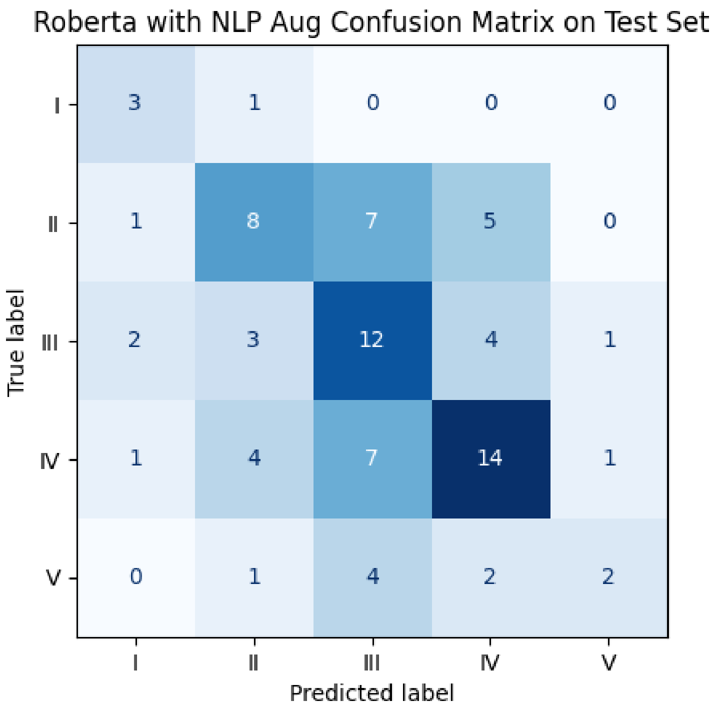
Roberta Base

```
*****************************************************************
Roberta Base Classification Report on Test Set
*****************************************************************
              precision    recall  f1-score   support

           I     0.5000    0.2500    0.3333         4
          II     0.4375    0.3333    0.3784        21
         III     0.3667    0.5000    0.4231        22
          IV     0.4571    0.5926    0.5161        27
           V     0.0000    0.0000    0.0000         9

    accuracy                         0.4217        83
   macro avg     0.3523    0.3352    0.3302        83
weighted avg     0.3807    0.4217    0.3918        83

*****************************************************************
```



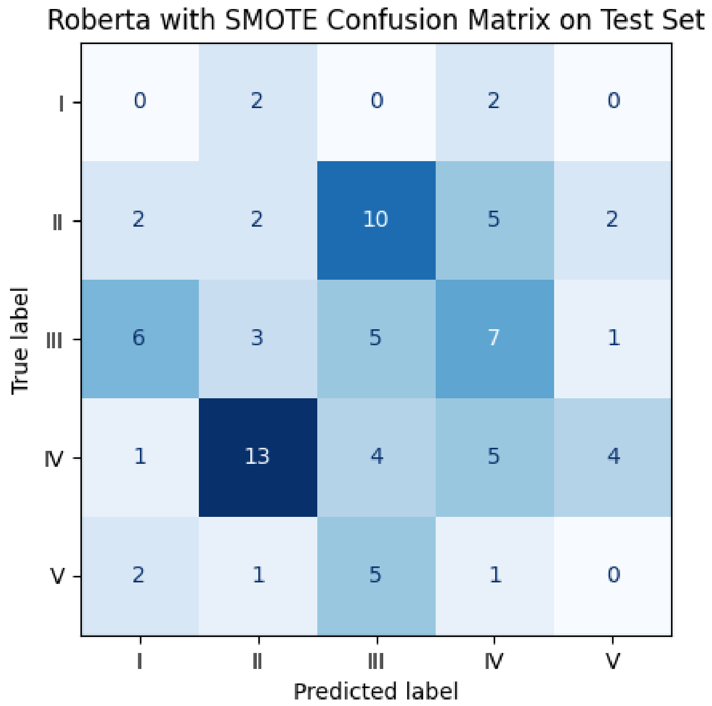
Roberta with NLP Aug

```
************************************************************
Roberta with NLP Aug Classification Report on Test Set
************************************************************
              precision    recall  f1-score   support

           I     0.4286    0.7500    0.5455         4
          II     0.4706    0.3810    0.4211        21
         III     0.4000    0.5455    0.4615        22
          IV     0.5600    0.5185    0.5385        27
           V     0.5000    0.2222    0.3077         9

    accuracy                         0.4699        83
   macro avg     0.4718    0.4834    0.4548        83
weighted avg     0.4821    0.4699    0.4637        83

************************************************************
```



Roberta with SMOTE

```
************************************************************
Roberta with SMOTE Classification Report on Test Set
************************************************************
              precision    recall  f1-score   support

           I     0.0000    0.0000    0.0000         4
          II     0.0952    0.0952    0.0952        21
         III     0.2083    0.2273    0.2174        22
          IV     0.2500    0.1852    0.2128        27
           V     0.0000    0.0000    0.0000         9

    accuracy                         0.1446        83
   macro avg     0.1107    0.1015    0.1051        83
weighted avg     0.1606    0.1446    0.1509        83

************************************************************
```



Roberta Models Comparison

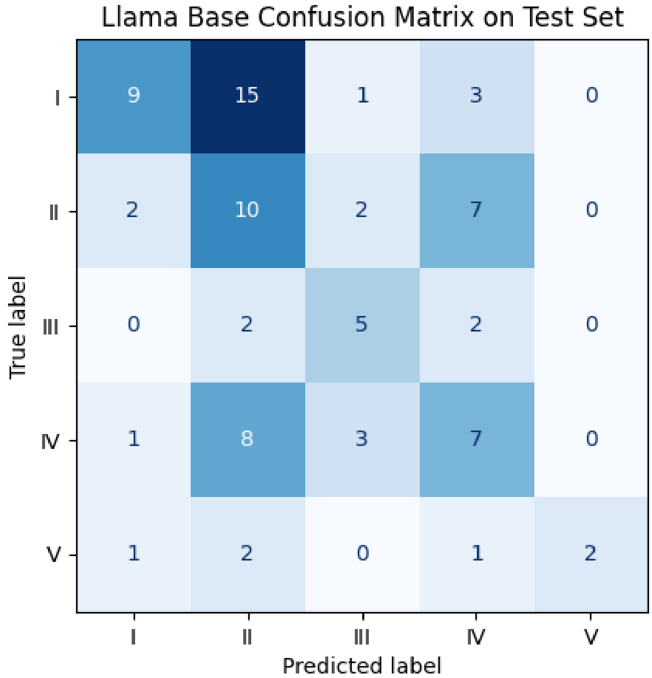
	Accuracy	F1	Precision	Recall
Roberta Base	0.421687	0.391837	0.380687	0.421687
Roberta with NLP Aug	0.469880	0.463680	0.482128	0.469880
Roberta with SMOTE	0.144578	0.150931	0.160643	0.144578

Observation

- Roberta with NLP Aug has the best performance scores
  - Accuracy: 47% F1 Score: 46%
- Roberta Base is second
  - Accuracy: 42% F1 Score: 39%
- Roberta with SMOTE has the least performance scores
  - Accuracy: 14% F1 Score: 15%

Llama

```
******************************************************************************
Llama Base Classification Report on Test Set
******************************************************************************
              precision    recall  f1-score   support

           I       0.69      0.32      0.44        28
          II       0.27      0.48      0.34        21
         III       0.45      0.56      0.50         9
          IV       0.35      0.37      0.36        19
           V       1.00      0.33      0.50         6

    accuracy                           0.40        83
   macro avg       0.55      0.41      0.43        83
weighted avg       0.50      0.40      0.41        83

******************************************************************************
```



Llama Model Performance

	Accuracy	F1	Precision	Recall
Llama Base	0.39759	0.407886	0.503629	0.39759

Observation

- Llama has below performance scores
  - Accuracy: 39% F1 Score: 40%

Models Comparison

Tabular Comparison

	Accuracy	F1	Precision	Recall	Model
0	0.493976	0.478952	0.494613	0.493976	Distilbert Base
7	0.469880	0.463680	0.482128	0.469880	Roberta with NLP Aug
2	0.457831	0.441832	0.526894	0.457831	Distilbert with SMOTE
1	0.469880	0.436153	0.414191	0.469880	Distilbert with NLP Aug
12	0.397590	0.407886	0.503629	0.397590	Llama Base
6	0.421687	0.391837	0.380687	0.421687	Roberta Base
3	0.361446	0.272897	0.223286	0.361446	XLNet Base
5	0.253012	0.255725	0.285838	0.253012	XLNet with SMOTE
4	0.337349	0.185652	0.361446	0.337349	XLNet with NLP Aug
9	0.325301	0.159693	0.105821	0.325301	Deberta Base
10	0.325301	0.159693	0.105821	0.325301	Deberta with NLP Aug
11	0.325301	0.159693	0.105821	0.325301	Deberta with SMOTE
8	0.144578	0.150931	0.160643	0.144578	Roberta with SMOTE

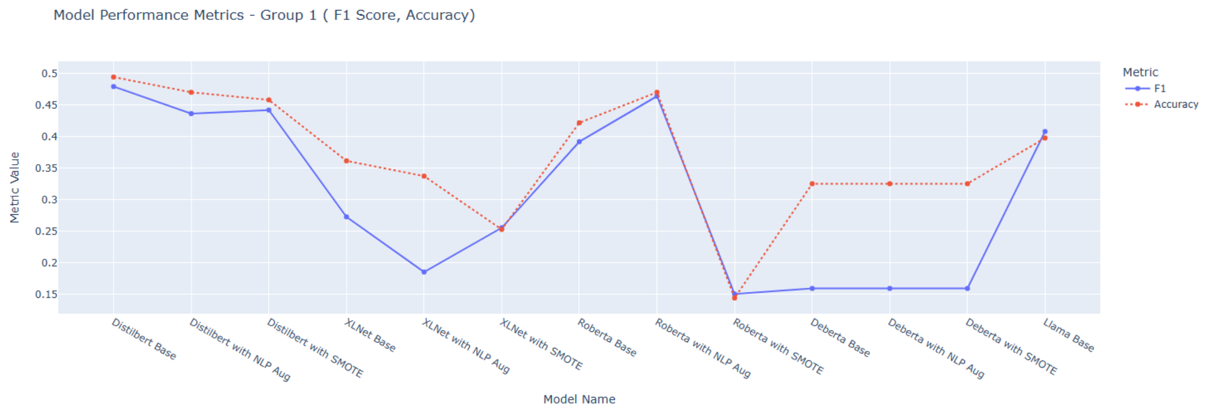
Best performing 3 transformers:

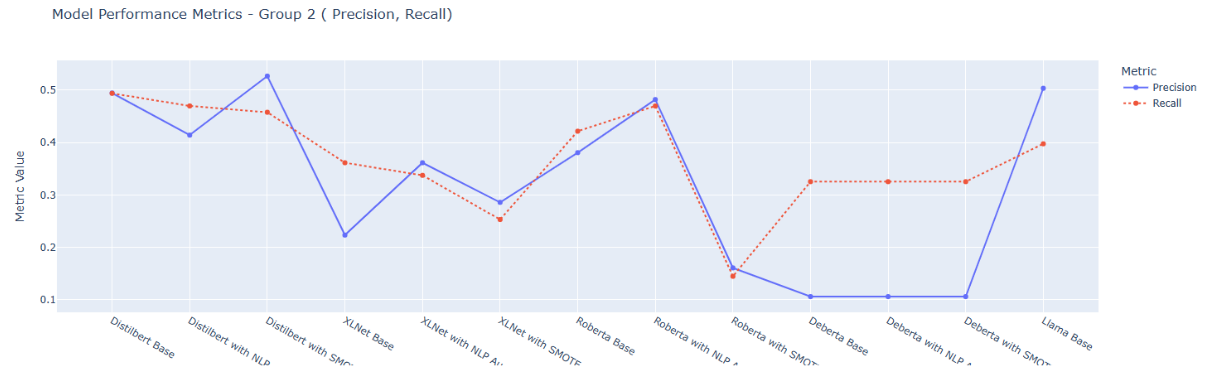
- Distibert Base
  - Accuracy:49% F1:47%
- Roberta with NLP Aug
  - Accuracy:47% F1:46%
- Distibert with SMOTE
  - Accuracy:45% F1:44%

Least performing 3 transformers:

- Deberta with NLP Aug
  - Accuracy:32% F1:16%
- Deberta with SMOTE
  - Accuracy:32% F1:16%
- Roberta with SMOTE
  - Accuracy:14% F1:15%

Graphical Comparison





Observation

- All models except Deberta and XLNet with NLP Aug show similar Accuracy and F1 Scores
- All models except Deberta and XLNet Base show similar Precision and Recall values

Demo on Prompting techniques

In this section, we will demonstrate the use of different prompting techniques against a sample finetuned transformer: (Llama)

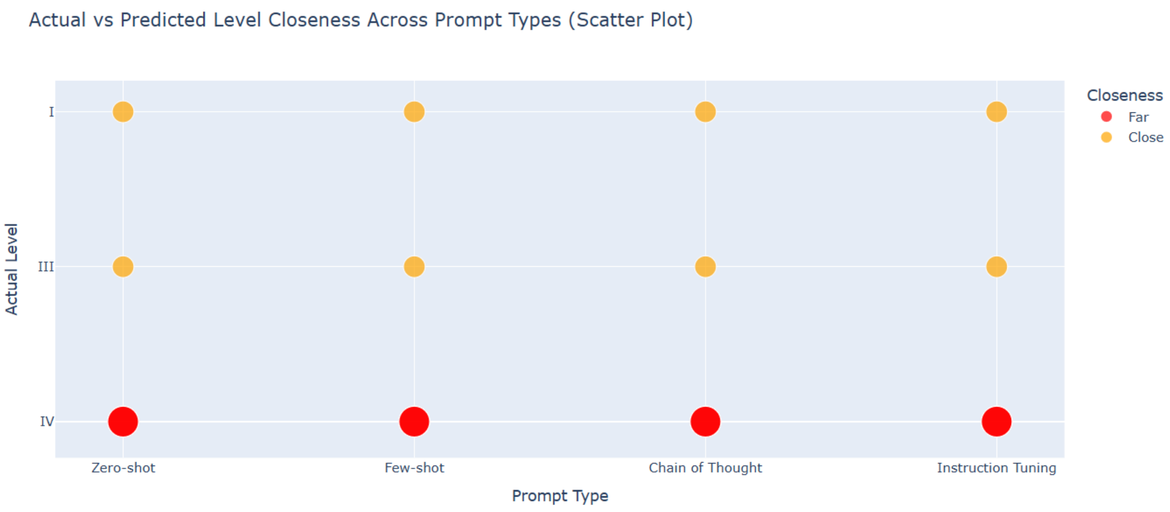
We will use below prompt styles and record the response for the first five accident records

- Zero-Shot prompt
- Few-shot prompt
- Chain of Thought
- Instruction tuning prompt

Different prompts

```
# Define different prompt strategies
def zero_shot_prompt(description):
    return f"Accident Description: {description}\nTask: Predict the potential accident level (I to V)."


def few_shot_prompt(description):
    return (
        "Example 1:\n"
        "Accident Description: A worker slipped on a wet floor and sprained their ankle.\nPredicted Level: I\n"
        "Example 2:\n"
        "Accident Description: A chemical spill caused a minor fire, requiring local evacuation.\nPredicted Level: III\n"
        "Now your turn:\n"
        f"Accident Description: {description}\nPredicted Level:"
    )


def chain_of_thought_prompt(description):
    return (
        f"Accident Description: {description}\n"
        "Analyze the accident severity based on the likelihood of harm, number of people affected, and required response measures. "
        "Then predict the accident level (I to V)."
    )


def instruction_tuning_prompt(description):
    return (
        "You are a safety analyst trained to assess accident descriptions and classify them into levels I to V, where I is minor and V is catastrophic.\n"
        f"Accident Description: {description}\nAccident Level:"
    )
```

Actual vs Predicted values on Sample on first 5 rows of dataset



Observation:

The red dots indicate the cases where the predicted level was further away than true level; the yellow dots indicate the predicted and true levels are closer

Chatbot Utility

We have built a Chatbot Utility for the end user. This will act as an interface for users to use our best model i.e. **DistilBert Base** to predict the **Potential Accident Level** for given **Industrial Accident description.**

Packages Used

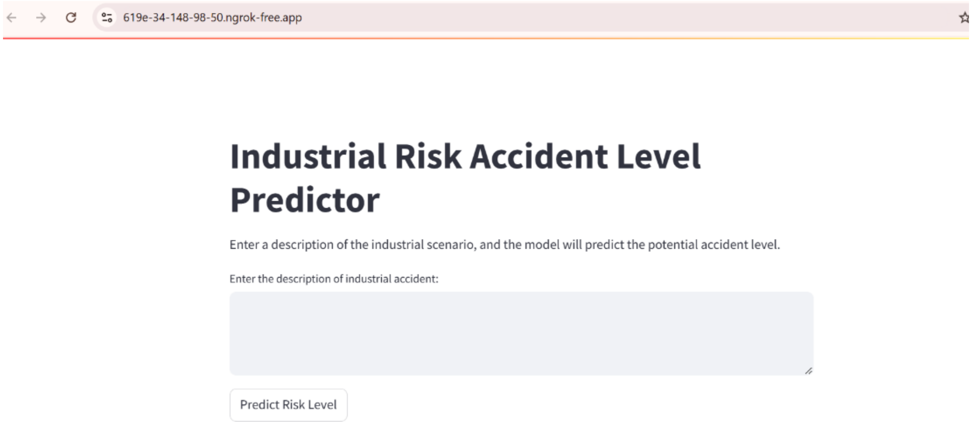
- STREAM LIT
- NGROK
- TORCH
- TRANSFORMERS

Perform following steps to build it

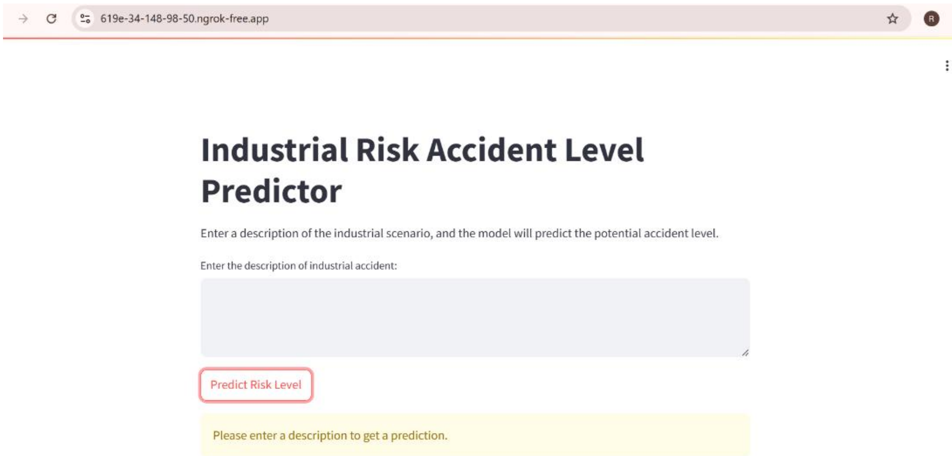
- Save the pre trained DistilBert model and generate following files
    - model.safetensors
    - vocab.txt
    - training_args.bin
    - tokenizer_config.json
    - special_tokens_map.json
    - config.json
- Load the saved model
- Start the streamlit server in background
- Host the app on NGROK and start the server
- Built the Stream Lit UI where user has to enter accident description **prompt** under text box with label i.e. "**Enter the description of industrial accident:**"
- User press the button with label **"Predict Risk Level"** to predict **Potential Accident Level.**

Screen Navigation

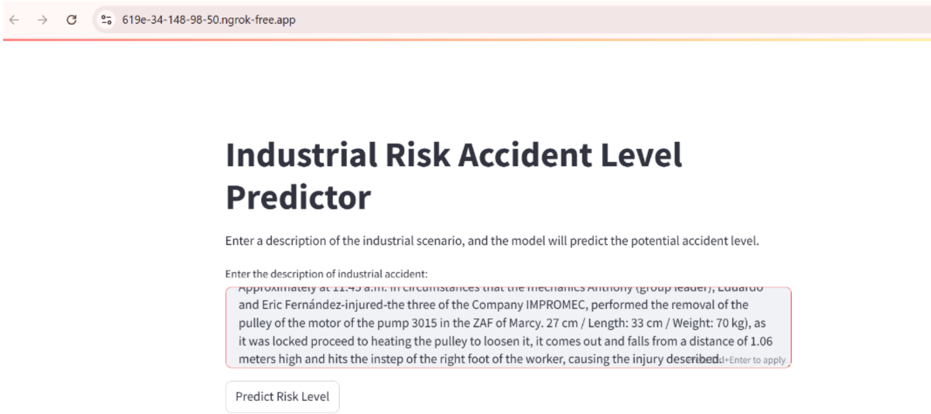
- **Chatbot Utility Screen**



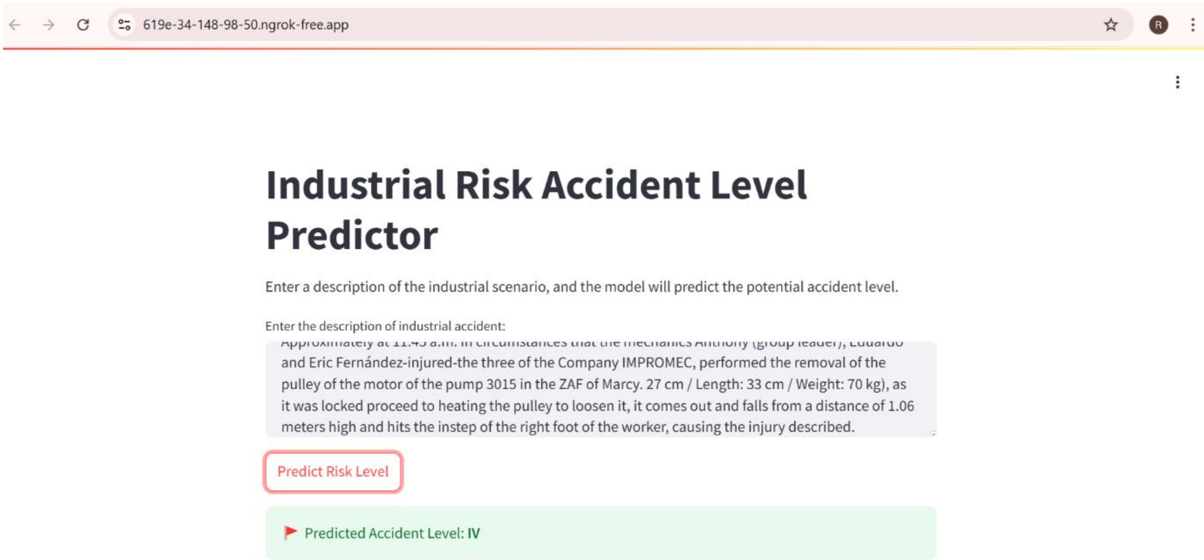
- **If User will press Button without entering prompt then it will give error**



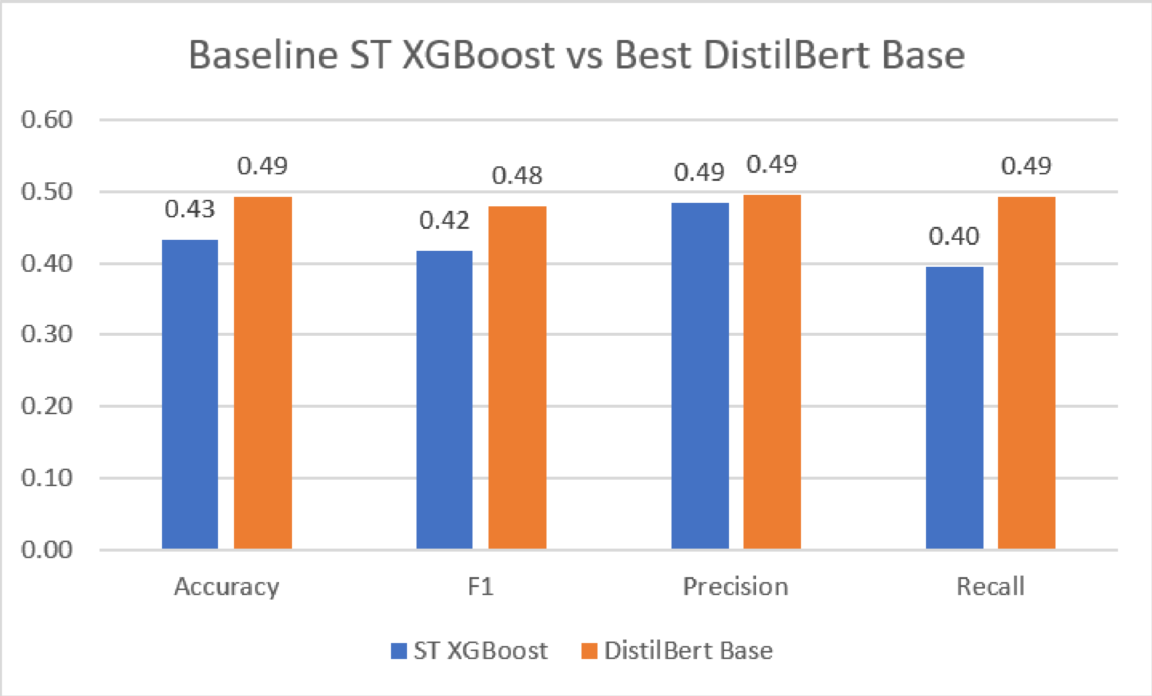
- **User enter the prompt (Description of Industrial Accident)**



- **Once hit the button, it will predict the Potential Accident Level**



Compare to the benchmark



- DistilBert-Base transformer (after finetuning) has achieved a F1 Score of 48% and Accuracy of 49%.
- This is a +6% increase on each metric as compared to Base XGBoost on Sentence Transformer embeddings
- The improvement is most likely because of the BeRT (Bidirectional Encoder Representations from Transformers) architecture which is the base for DistilBert. It enables the transformer to capture long term context in the accident descriptions more accurately than the baseline ML model
- Precision is almost similar but DistilBert achieves +9% on the Recall metric as well

Visualisations

Most of the data related analysis was completed as part of Milestone 1. Please refer the section Exploratory Data Analysis

Another important observation was a high rate of misclassification across different accident levels while using different strategies. So, further investigation was conducted to identify if there are common words that are used across different levels.

We will verify this by performing below steps:

- We will identify the Top 20 words for each Accident Level based on their occurrence count.
- Then we will identify how many of these are recurring across levels.

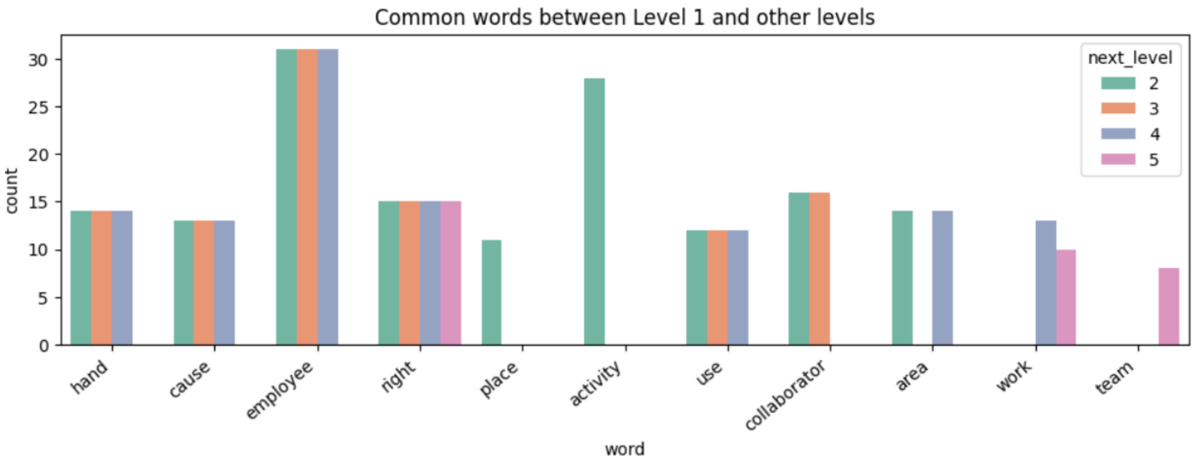
Code snippet to find Top 20 words for each accident level

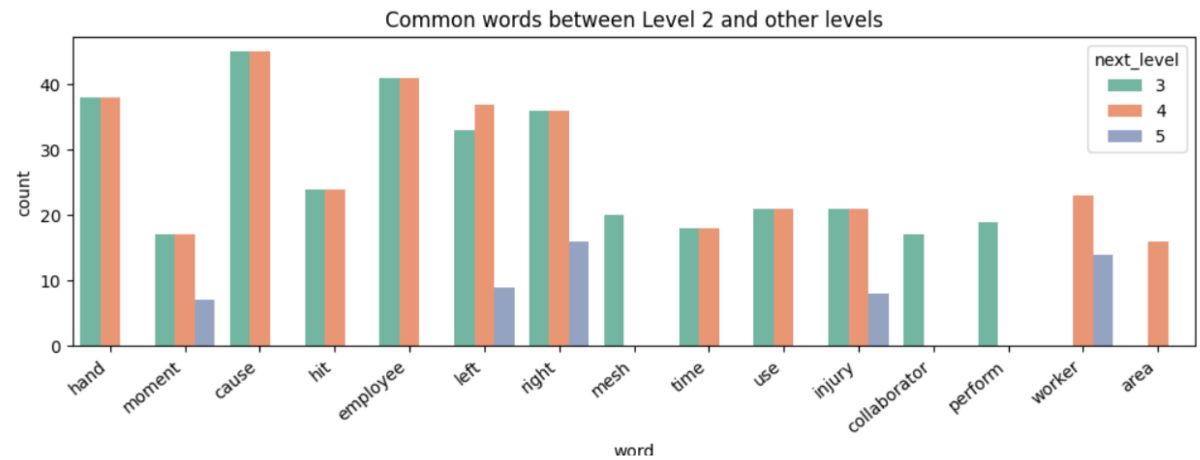
```
top_words_dict = {}
for level in range(5):

    level_df = df[df['labels'] == level]

    # Flatten the list of lists into a single list of words
    all_words = [word for sublist in level_df['Tokenized_Description'] for word in sublist]

    # Count the frequency of each word
    word_counts = Counter(all_words)

    # Select the top 20 most frequent words in each level
    top_words = dict(word_counts.most_common(20))
    top_words_dict[level] = top_words
```

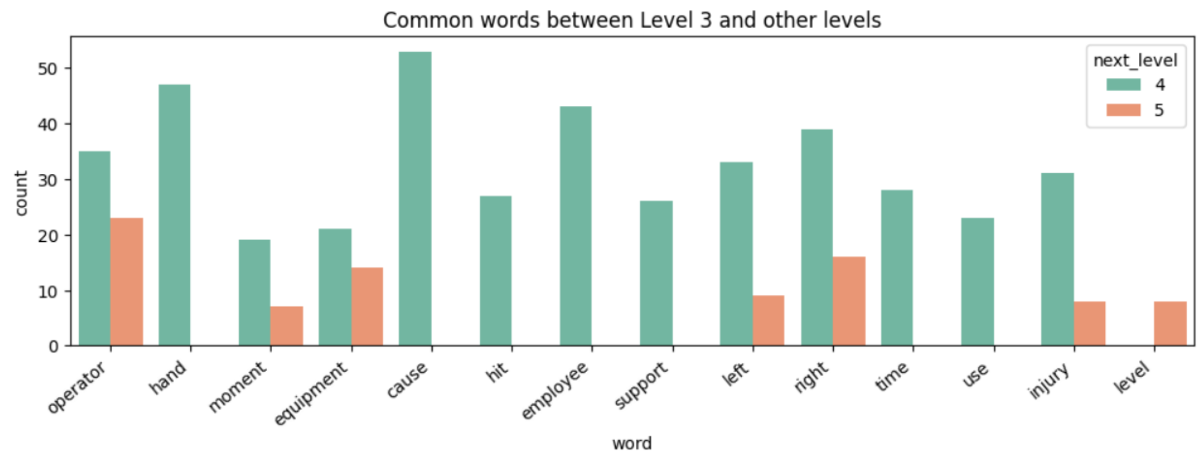
```
#find the common words between two levels
level_wise_words = pd.DataFrame()
for level in range(5):

    for next_level in range(level + 1, 5):
        common_words = set(top_words_dict[level].keys()) & set(top_words_dict[next_level].keys())


        for word in common_words:
            new_row = pd.DataFrame({'word': [word], 'level': [level+1], 'next_level': [next_level+1],
                                    'count': [min(top_words_dict[level].get(word),top_words_dict[next_level].get(word))]})
            level_wise_words = pd.concat([level_wise_words, new_row], ignore_index=True)
```

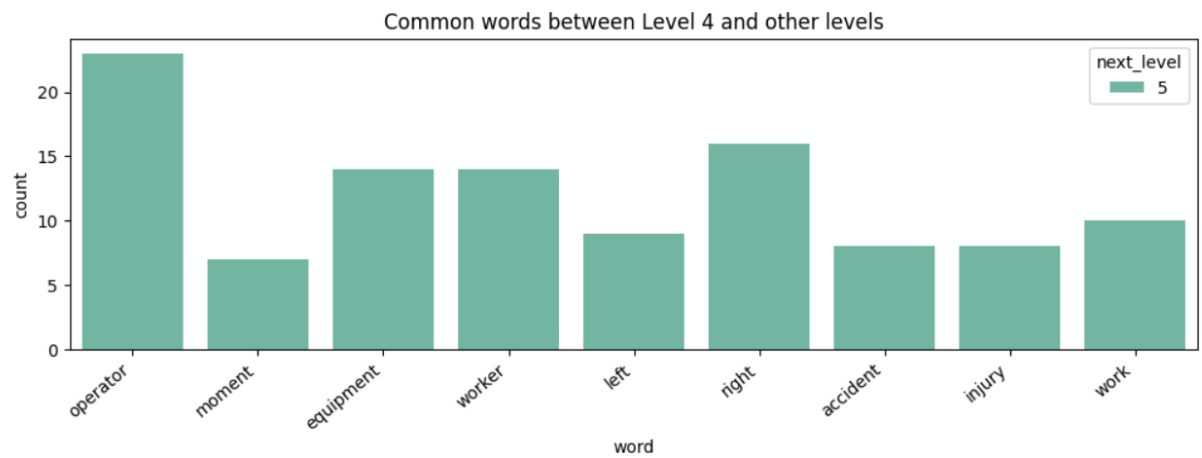
Code snippet to find common words among the Top 20:

```
#find the common words between two levels
level_wise_words = pd.DataFrame()
for level in range(5):

    for next_level in range(level + 1, 5):
        common_words = set(top_words_dict[level].keys()) & set(top_words_dict[next_level].keys())


        for word in common_words:
            new_row = pd.DataFrame({'word': [word], 'level': [level+1], 'next_level': [next_level+1],
                                    'count': [min(top_words_dict[level].get(word),top_words_dict[next_level].get(word))]})
            level_wise_words = pd.concat([level_wise_words, new_row], ignore_index=True)
```









Observations

- Each level has several words that are also common in other levels
- Level I has 11 words that are also commonly used in other levels, specially Level II
- Level II has 15 words that are also commonly used in other levels, specially Level III
- Level III has 14 words that are also commonly used in other levels, specially Level IV
- Level IV has 9 words that are also commonly used in Level V

The above overlap and overuse of similar words make it difficult to identify the exact Potential Accident Level using only the Accident Description. Suggestion is to use a standard template for reporting accident that can capture the nature of accident, impact, extent of injury/damage and relevant details with proper context.

Implications

The Chatbot utility can act as a first point of reporting to record any accident across industries, especially where manual labour is involved.

This will help the safety professionals to quickly analyse and validate the safety incidents thereby saving people and property from injury or damage.

Business Recommendation

- Prepare a standard template for reporting accidents with the inputs of industrial safety experts. This will ensure all accident descriptions maintain similar context without losing any necessary information
- To improve prediction accuracy, we recommend incorporating additional granular features that explicitly capture specific accident characteristics. For example, features such as "Was hazardous material present?", "Was heavy machinery in use?" or "Was the incident indoors or outdoors?". This can help the model better differentiate between high-risk and low-risk incidents. By structuring this information alongside the textual description, we can enhance the model's ability to recognize patterns in accident severity
- The current dataset is limited to January 2016 to July 2017. If feasible, more accident details need to be recorded for earlier or recent timelines
- The current dataset contains more accident observations from Country 1. Further investigation or data needs to be collected to check if Country 1 is more prone to accidents or it is just a larger country with many industrial plants. Appropriate action can then be recommended based on the findings

Limitations

- Due to higher cost of API calls involved in using well known transformer/LLMs like ChatGPT / DeepSeek and lack of high capacity GPU, the current implementation utilises smaller, open source models. This may compromise model performance to an extent. As of now, the maximum achieved F1 is around 50% and hence there is scope for improvement
- Another drawback is the limited size of the original dataset. The data needs to be recorded for significant time period and across many industries and locations to be generally useful
- A ML pipeline needs to be implemented which can help to track model performance and retrain the transformer when performance metrics decrease.

Closing Reflections

- The project highlights the importance of NLP in identifying safety risks across industries
- Lessons learned include the need for extensive relevant data and the iterative process of building models by applying different strategies and evaluate model performance
- We have also learnt to balance the technical feasibility and practical implementation of a solution
- In future:
    - we will plan to gather more relevant data from all feasible sources
    - use a higher capacity GPU or paid API to access top quality LLM for this task
    - build an end-to-end pipeline from data preprocessing through model deployment and model maintenance