AIML
Capstone Project

Industrial Safety: NLP Based Chatbot

Team: Group - 10 - NLP1

Batch: AIML ONLINE SEPTEMBER 23 A

By

Mitesh Waghela
Raghavendra Sriram Vempaty
Sridhar G R
Shaurya Bhagat
Rethina Durai S J
Karan Raj Sharma

Great Learning
POWER AHEAD

NATURAL LANGUAGE PROCESSING

NLP Chatbot - Milestone-1

Version 1.0

August 24, 2024

Contents

About This Capstone Milestone Report

History

Version No.	Issue Date	Author	Status	Reason for Change
1.0	24-Aug-2024	Group 10	Initial Version	NA

Review

Reviewers	Version No.	Date
Mitesh Waghela	1.0	24-Aug-2024
Raghavendra Sriram Vempaty	1.0	24-Aug-2024
Shaurya Bhagat	1.0	24-Aug-2024
Rethina Durai S J	1.0	24-Aug-2024
Karan Raj Sharma	1.0	24-Aug-2024
Sridhar G.R.	1.0	24-Aug-2024

Approvers

Approvers	Version No.	Date
Jayanth – Great Learning	1.0	24-Aug-2024

1 Problem Statement

1.1 Domain and Context

DOMAIN: Industrial safety - NLP based Chatbot.

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

1.2 Objective

To design a ML/DL based chatbot utility which can help the professionals to highlight the safety risk as per the incident description

1.3 Data Description

The database is basically records of accidents from 12 different plants in three different countries where every line in the data is an occurrence of an accident. Different columns with its description are as follows:

- **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.

Link to download the dataset:

https://www.kaggle.com/ihmstefanini/industrial-safety-and-health-analytics-database
[for your reference only]

`

2 Approach and Timelines

2.1 Project Approach and Timelines

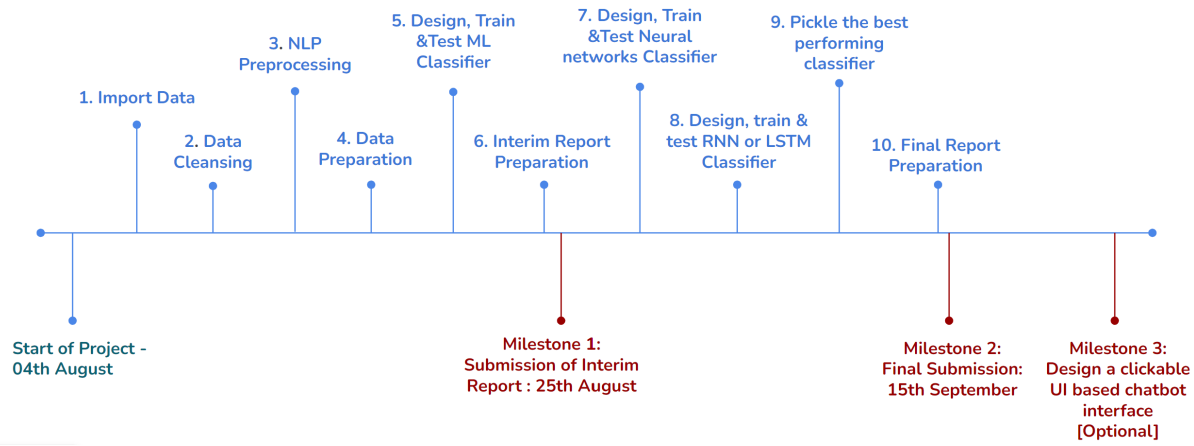


Figure 1 Project Approach and Timelines

2.2 Assumptions

List of assumptions agreed upon with mentor (Jayanth) for Group-10.

1. Utilize the dataset provided in Capstone project details link for NLP
2. Report need not include all the steps mentioned in the Google Collab notebook used to develop
3. To capture both Test & Train data to showcase which is the best classifier
4. Include only relevant visualizations

2.3 Acknowledgement

Group-10 would like to acknowledge Jayanth's support for each and every phase of this project and thank the Great learning team for their diligent support throughout this phase of the project.

`

3 Data Import and Exploration

The main objective of this step is to import the data set and check for the type of data present within each attribute.

	Unnamed: 0	Data	Countries	Local	Industry Sector	Accident Level	Potential Accident Level	Genre	Employee or Third Party	Critical Risk	Description
0	0	2016-01-01	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 for maintenance, the supervisor proceeds to loosen the support of the intermediate centralizer to facilitate the removal, seeing this the mechanic supports one end on the drill of the equipment to pull with both hands the bar and accelerate the removal from this, at this moment the bar slides from its point of support and tightens the fingers of the mechanic between the drilling bar and the beam of the jumbo.
1	1	2016-01-02	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pump, the piping was uncoupled and the sulfide solution was designed in the area to reach the maid. Immediately she made use of the emergency shower and was directed to the ambulatory doctor and later to the hospital. Note: of sulphide solution = 48 grams / liter.
2	2	2016-01-06	Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170 when the collaborator was doing the excavation work with a pick (hand tool), hitting a rock with the flat part of the beak, it bounces off hitting the steel tip of the safety shoe and then the metatarsal area of the left foot of the collaborator causing the injury.
3	3	2016-01-08	Country_01	Local_04	Mining	I	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 CX-695 OB7, the personnel begins the task of unlocking the Soquet bolts of the BHB machine, when they were in the penultimate bolt they identified that the hexagonal head was worn, proceeding Mr. Cristóbal - Auxiliary assistant to climb to the platform to exert pressure with your hand on the "DADO" key, to prevent it from coming out of the bolt; in those moments two collaborators rotate with the lever in anti-clockwise direction, leaving the key of the bolt, hitting the palm of the left hand, causing the injury.
4	4	2016-01-10	Country_01	Local_04	Mining	IV	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances that the mechanics Anthony (group leader), Eduardo and Eric Fernández-injured-the three of the Company IMPROMEC, performed the removal of the pulley of the motor of the pump 3015 in the ZAF of Marcy. 27 cm / Length: 33 cm / Weight: 70 kg), as it was locked proceed to heating the pulley to loosen it, it comes out and falls from a distance of 1.06 meters high and hits the instep of the right foot of the worker, causing the injury described.

Shape and Datatype of each attribute:

```
[ ]  print("Number of rows = {0} and Number of Columns = {1} in the Data frame".format(ISH_df.shape[0], ISH_df.shape[1]))
```

```
Number of rows = 425 and Number of Columns = 11 in the Data frame
```

Datatype of each attribute:

```
[ ]  # Check datatypes
     ISH_df.dtypes
```

	0
Unnamed: 0	int64
Data	datetime64[ns]
Countries	object
Local	object
Industry Sector	object
Accident Level	object
Potential Accident Level	object
Genre	object
Employee or Third Party	object
Critical Risk	object
Description	object

dtype: object

Checking for Null and Missing Values:

Unnamed: 0	0
Data	0
Countries	0
Local	0
Industry Sector	0
Accident Level	0
Potential Accident Level	0
Genre	0
Employee or Third Party	0
Critical Risk	0
Description	0

dtype: int64

`

Checking for Duplicate Values:

```
Duplicate_Rows = ISH_df.duplicated().sum()
print("Number of duplicate rows:", Duplicate_Rows)
```

```
Number of duplicate rows: 7
```

```
# View Duplicate records
Duplicates = ISH_df.duplicated()

ISH_df[Duplicates]
```

	Date	Country	City	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee Type	Critical Risk	Description
77	2016-04-01	Country_01	City_01	Mining	I	V	Male	Third Party (Remote)	Others	In circumstances that two workers of the Abratech company were doing putty work inside the conditioning tank (5 meters deep and covered by platforms) of metal gratings - grating- in the upper part), two other employees of the HyT company carried out maneuvers transfer of a pump with the help of a manual tick - which worked hooked to a beam H, dragging the pump on the metal gratings (grating), suddenly the pump is hooked with a metal grate (grating) and when trying to release it, the metal grid (grating - 13.0 Kg. (60 cm x 92 cm)) falls inside the tank, hits a diagonal channel inside the tank and then impacts the right arm of one of the workers and rubs the helmet of the second worker that he was crouching. The area where the bomb was being moved was marked with tape and did not have a lookout.
262	2016-12-01	Country_01	City_03	Mining	I	IV	Male	Employee	Others	During the activity of chuteo of ore in hopper OP5; the operator of the locomotive parks his equipment under the hopper to fill the first car, it is at this moment that when it was blowing out to release the load, a mud flow suddenly appears with the presence of rock fragments; the personnel that was in the direction of the flow was covered with mud.
303	2017-01-21	Country_02	City_02	Mining	I	I	Male	Third Party (Remote)	Others	Employees engaged in the removal of material from the excavation of the well 2 of level 265, using shovel and placing it in the bucket. During the day some of this material fell into the pipes of the employees' boots and the friction between the boot and the calf caused a superficial injury to the legs.
345	2017-03-02	Country_03	City_10	Others	I	I	Male	Third Party	Venomous Animals	On 02/03/17 during the soil sampling in the region of Sta. the employees Rafael and Danillo da Silva were attacked by a bee test. They rushed away from the place, but the employee Rafael took 4 bites, one on the chin, one on the chest, one on the neck and one on the hand over the glove. The employee took 4 bites, one in his hand over his glove and the other in the head, and the employee Danillo took 2 bites in the left arm over his uniform. At first no one sketched allergy, just swelling at the sting site. The activity was stopped to evaluate the site, after verifying that the test had remained in the line, they left the site.
346	2017-03-02	Country_03	City_10	Others	I	I	Male	Third Party	Venomous Animals	On 02/03/17 during the soil sampling in the region of Sta. the employees Rafael and Danillo da Silva were attacked by a bee test. They rushed away from the place, but the employee Rafael took 4 bites, one on the chin, one on the chest, one on the neck and one on the hand over the glove. The employee took 4 bites, one in his hand over his glove and the other in the head, and the employee Danillo took 2 bites in the left arm over his uniform. At first no one sketched allergy, just swelling at the sting site. The activity was stopped to evaluate the site, after verifying that the test had remained in the line, they left the site.
355	2017-03-15	Country_03	City_10	Others	I	I	Male	Third Party	Venomous Animals	Team of the VMS Project performed soil collection on the Xixás target with 3 members. When the teams were moving from one collection point to another, Mr. Fabio was ahead of the team, stinging behind Robson and Manoel da Silva. near the collection point were surprised by a swarm of bees that was inside a I play near the ground, with no visibility in the woods and no hissing noise. Fabio passed by the stump, but Robson and Manoel da Silva were attacked by the bees. Robson had a sting in his left arm over his uniform and Manoel da Silva had a prick in his lip as his screen ripped as he tangled in the branches during the escape.
397	2017-05-23	Country_01	City_04	Mining	I	IV	Male	Third Party	Projection of fragments	In moments when the 02 collaborators carried out the inspection of the conveyor belt No. 3 from the tail pulley when they were at the height of the load polymer No. 372, the Maslucan collaborator heard a noise where note that the belt was moving towards the tail pulley, 4 "fragmentos mineral fragments are projected towards the access of the ramp impacting the 2 collaborators, being evacuated to the medical post.

Number of duplicate rows identified are 7.

Data Exploration Observations:

Overall:

1. The dataset contains information on industrial accidents across different countries, cities, and industry sectors.
2. The time frame of the accidents is captured in the 'Date' column.
3. The severity of accidents is categorized into levels from I to VI.
4. Information about the gender, employee type, critical risk, and a detailed description of the accident is provided.

Specific Observations:

- **Country:** Most accidents occurred in Country_01, followed by Country_02 and Country_03.
- **City:** The distribution of accidents across cities varies, with some cities having a higher number of incidents than others.
- **Industry Sector:** The 'Mining' sector has the highest number of accidents, indicating a potentially higher risk in this industry.
- **Accident Level:** The majority of accidents fall under levels I and II, suggesting that most accidents are relatively minor in severity.
- **Potential Accident Level:** There's a notable difference between the actual accident level and the potential accident level, highlighting the importance of preventive measures.
- **Gender:** Male employees are involved in a significantly higher number of accidents compared to females.
- **Employee Type:** Most accidents involve employees rather than third parties.
- **Critical Risk:** 'Others' is the most frequent category in critical risk, which might indicate a need for more specific categorization.
- **Description:** The description column provides detailed narratives of the accidents, which can be valuable for further text analysis and understanding the circumstances leading to accidents.

`

Potential Areas for Further Analysis:

1. Investigate the reasons behind the higher number of accidents in specific countries, cities, and industry sectors.
2. Analyze the factors contributing to the difference between actual and potential accident levels.
3. Explore the reasons for the gender disparity in accident involvement.
4. Deep dive into the 'Others' category in critical risk to identify potential subcategories.
5. Perform text analysis on the 'Description' column to extract insights and patterns related to accident causes.

`

4 Data Cleansing

The main objective of this step is to fix any inconsistency found in the data file shared by great learning.

4.1 Cleansing steps

- Dropping Duplicate Records
- Dropping irrelevant columns
- Aligning Attribute names as per data available
- Checking the resulting dataframe and unique values for each attribute.

Dropping the duplicates

```
# Check for Duplicate rows in the dataset

Duplicate_Rows = ISH_df.duplicated().sum()
print('Number of duplicate rows:', Duplicate_Rows)
```

```
Number of duplicate rows: 7
```

```
# View Duplicate records
Duplicates = ISH_df.duplicated()

ISH_df[Duplicates]
```

```
# Remove duplicate rows and save the deduplicated dataset

ISH_df_cleaned = ISH_df.drop_duplicates()

# Save the deduplicated dataset to a new file
ISH_df_cleaned.to_csv('ISH_df_cleaned.csv', index=False)

# Print the number of rows before and after deduplication
print('Number of rows before deduplication:', len(ISH_df))
print('Number of rows after deduplication:', len(ISH_df_cleaned))
```

```
Number of rows before deduplication: 425
Number of rows after deduplication: 418
```

Number of duplicate rows identified are 7. Dropping these rows reduces the dataframe from 425 rows to 418 rows.

New dataframe shape is (418, 10)

Dropping irrelevant columns

```
# Dropping Unnecessary Columns:

ISH_df.drop("Unnamed: 0", axis=1, inplace=True)
```

Unnamed: 0: This column appears to be an index column and does not provide any useful information for analysis.

Updated dataframe shape is (418, 10)

`

Aligning Attribute names as per data available

1. Column name spelling fixed. Example: Data and Genre were misspelled, were changed to **Date** and **Gender**
2. Column names were changed to match appropriate nomenclature. e.g. Countries changed to **Country**, Local to **City**, Employee or Third Party to **Employee Type**

```
[ ]  # Renaming the columns as per available Data and Description
     ISH_df.rename(columns={
         "Data": "Date",
         "Countries": "Country",
         "Local": "City",
         "Genre": "Gender",
         "Employee or Third Party":"Employee Type",
     }, inplace=True)

     # Modify 'City' column values
     ISH_df['City'] = ISH_df['City'].str.replace('Local_', 'City_')

     ISH_df.head()
```

Checking the updated dataframe post data cleansing:

	Date	Country	City	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee Type	Critical Risk	Description
0	2016-01-01	Country_01	City_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 for maintenance, the supervisor proceeds to loosen the support of the intermediate centralizer to facilitate the removal, seeing this the mechanic supports one end on the drill of the equipment to pull with both hands the bar and accelerate the removal from this, at this moment the bar slides from its point of support and tightens the fingers of the mechanic between the drilling bar and the beam of the jumbo.
1	2016-01-02	Country_02	City_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pump, the piping was uncoupled and the sulfide solution was designed in the area to reach the maid. Immediately she made use of the emergency shower and was directed to the ambulatory doctor and later to the hospital. Note: of sulphide solution = 48 grams / liter.
2	2016-01-06	Country_01	City_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170 when the collaborator was doing the excavation work with a pick (hand tool), hitting a rock with the flat part of the beak, it bounces off hitting the steel tip of the safety shoe and then the metatarsal area of the left foot of the collaborator causing the injury.
3	2016-01-08	Country_01	City_04	Mining	I	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 CX-695 OB7, the personnel begins the task of unlocking the Soquet bolts of the BHB machine, when they were in the penultimate bolt they identified that the hexagonal head was worn, proceeding Mr. Cristóbal - Auxiliary assistant to climb to the platform to exert pressure with your hand on the "DADO" key, to prevent it from coming out of the bolt; in those moments two collaborators rotate with the lever in anti-clockwise direction, leaving the key of the bolt, hitting the palm of the left hand, causing the injury.
4	2016-01-10	Country_01	City_04	Mining	IV	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances that the mechanics Anthony (group leader), Eduardo and Eric Fernández-injured-the three of the Company IMPROMEC, performed the removal of the pulley of the motor of the pump 3015 in the ZAF of Marcy. 27 cm / Length: 33 cm / Weight: 70 kg), as it was locked proceed to heating the pulley to loosen it, it comes out and falls from a distance of 1.06 meters high and hits the instep of the right foot of the worker, causing the injury described.

Checking the resulting dataframe and unique values for each attribute.

Date	287
Country	3
City	12
Industry Sector	3
Accident Level	5
Potential Accident Level	6
Gender	2
Employee Type	3
Critical Risk	33
Description	411

Identified numerical and categorical columns

```
# Identify numerical and categorical columns
numerical_columns = ISH_df_cleaned.select_dtypes(include=[np.number]).columns.tolist()
categorical_columns = ISH_df_cleaned.select_dtypes(exclude=[np.number]).columns.tolist()

# Exclude 'Data' column from categorical columns
categorical_columns = [col for col in categorical_columns if col != 'Date']
print('Numerical columns:', numerical_columns)
print('Categorical columns:', categorical_columns)
```

```
Numerical columns: []
Categorical columns: ['Country', 'City', 'Industry Sector', 'Accident Level', 'Potential Accident Level', 'Gender', 'Employee Type', 'Critical Risk', 'Description']
```

`

5 Data Visualization

Below is different visualization charts generated for the detailed data exploration:

1. **Univariate Plots**

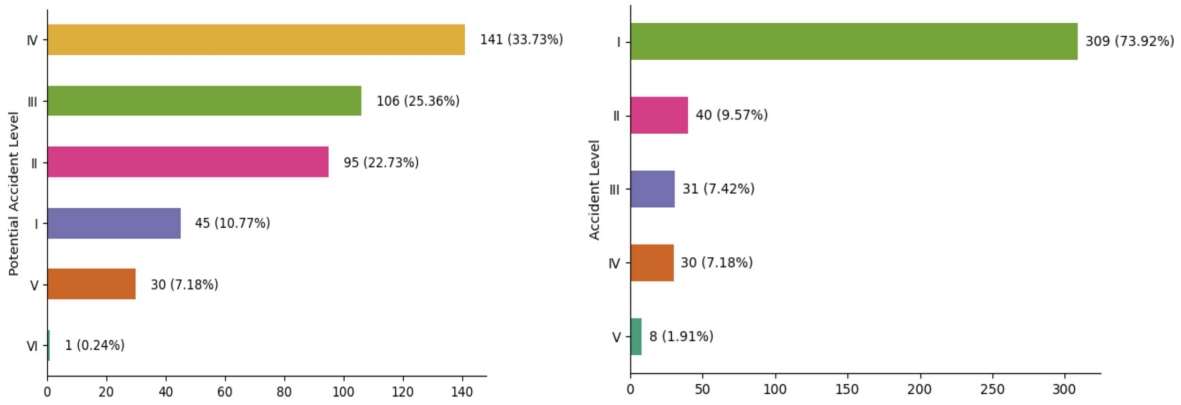


Figure 2 Potential Accident Level Distribution and Accident Level Distribution

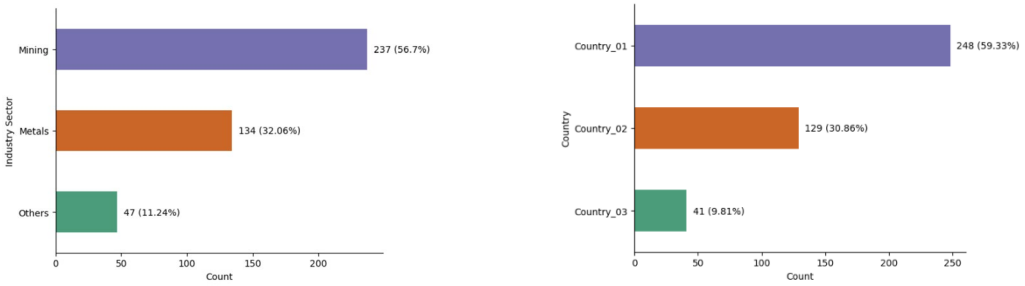


Figure 3 Industry Sector Distribution and Country Distribution

`

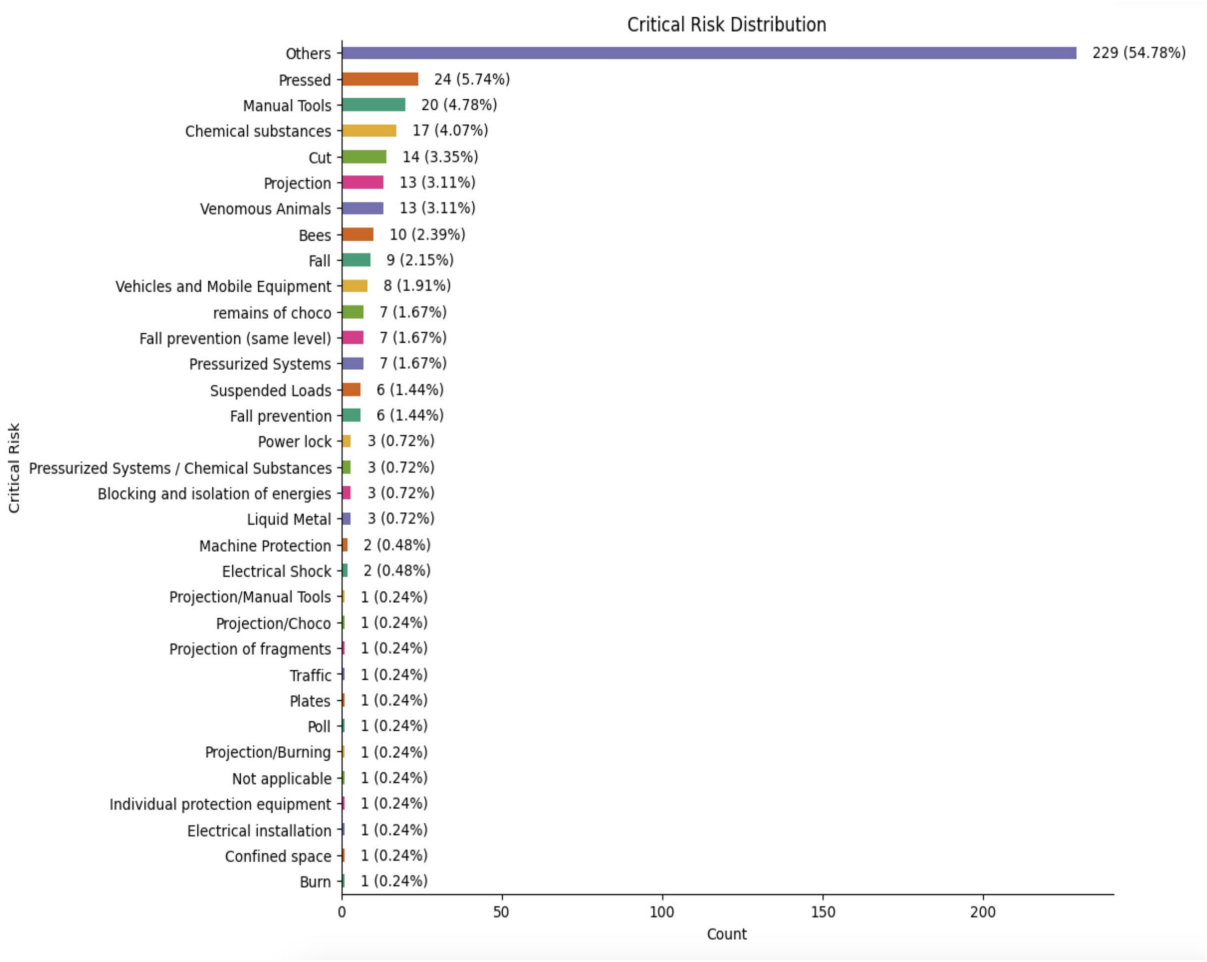


Figure 4 Critical Risk Distribution

2. Bivariate Plots

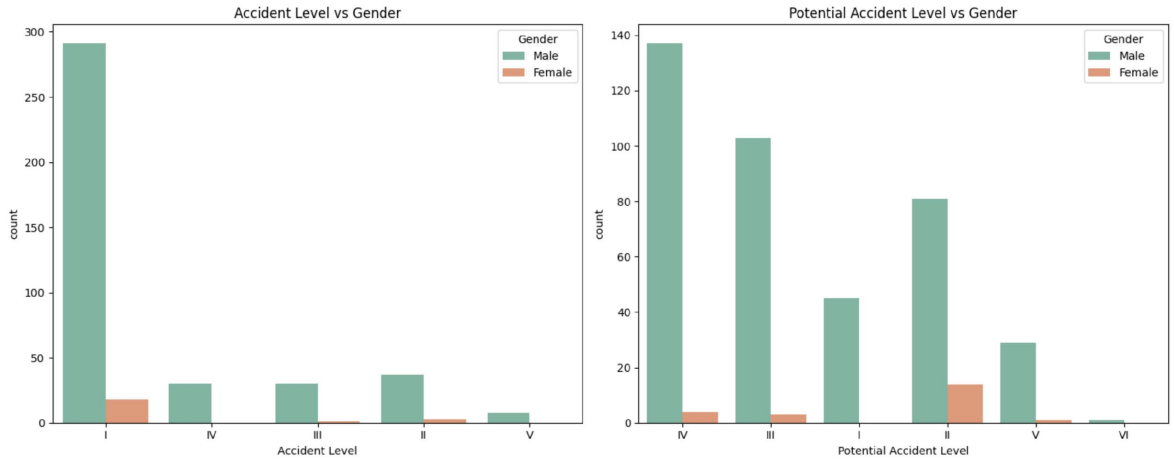


Figure 1 Accident Level Vs Gender & Potential Accident Level Vs Gender Approach and Timelines

`

Observations:

Accident Level vs Gender:

1. A significantly higher number of males are involved in accidents across all accident levels.
2. The disparity is particularly pronounced in lower accident levels (I and II).

Potential Accident Level vs Gender:

3. Similar to the actual accident level, males are more likely to be involved in potential accidents.
4. The difference in potential accident levels between genders is less pronounced compared to actual accidents, suggesting that preventive measures might be more effective for males.

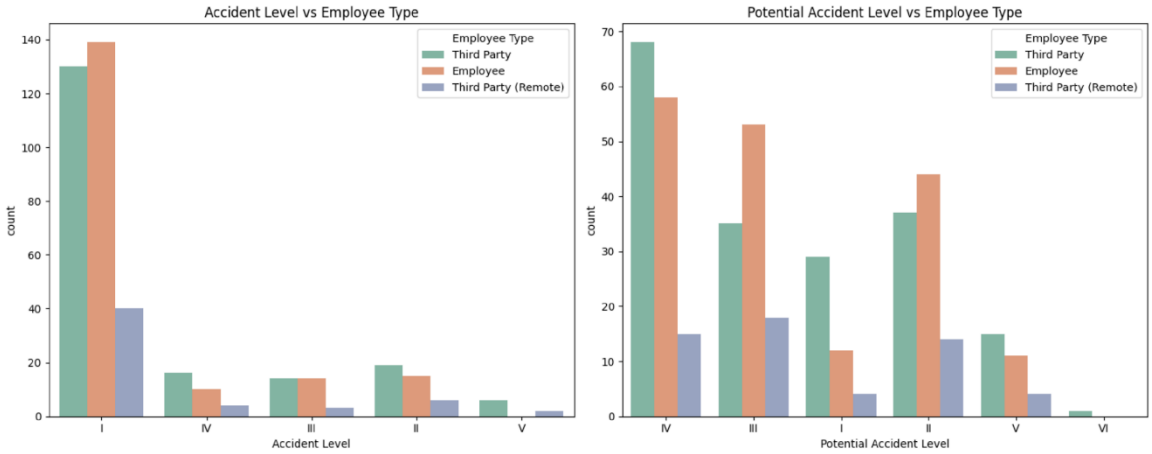


Figure 2 Accident Level Vs Employee Type & Potential Accident Level Vs Employee Type

Observations:

Accident Level vs Employee Type:

1. Employees are involved in a significantly higher number of accidents across all accident levels compared to third parties.
2. The difference is particularly pronounced in lower accident levels (I and II).

Potential Accident Level vs Employee Type:

1. Similar to the actual accident level, employees are more likely to be involved in potential accidents compared to third parties.
2. The difference in potential accident levels between employee types is less pronounced compared to actual accidents. This suggests that preventive measures might be more effective for employees.

`

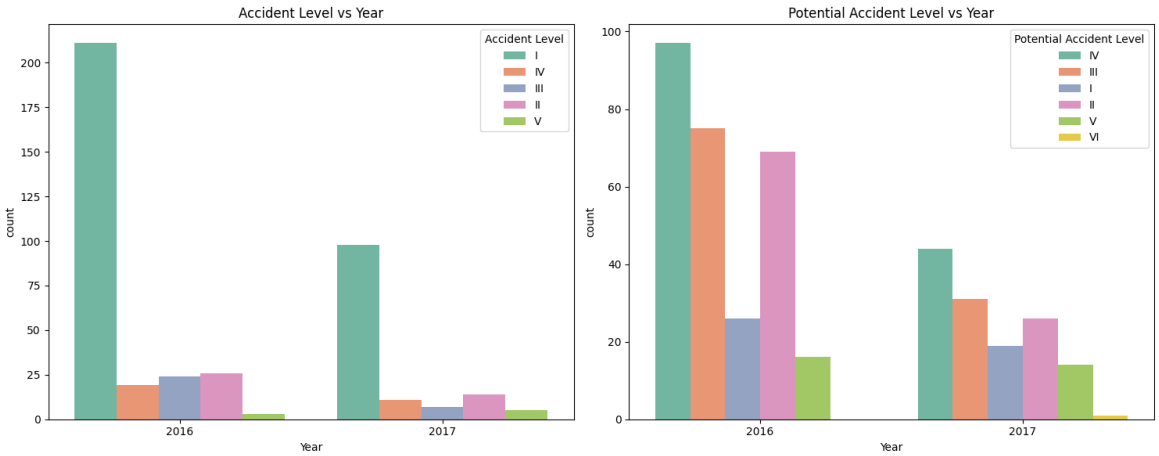


Figure 3 Accident Level Vs Month & Potential Accident Level Vs Month

Accident Level vs Year:

1. There's a noticeable decrease in the number of accidents across all levels in the later years compared to the initial years.
2. This suggests a positive trend in terms of safety improvements over time.

Potential Accident Level vs Year:

1. Similar to the actual accident level, potential accidents also show a decreasing trend over the years.
2. This indicates that preventive measures and safety protocols might be becoming more effective in mitigating potential risks.

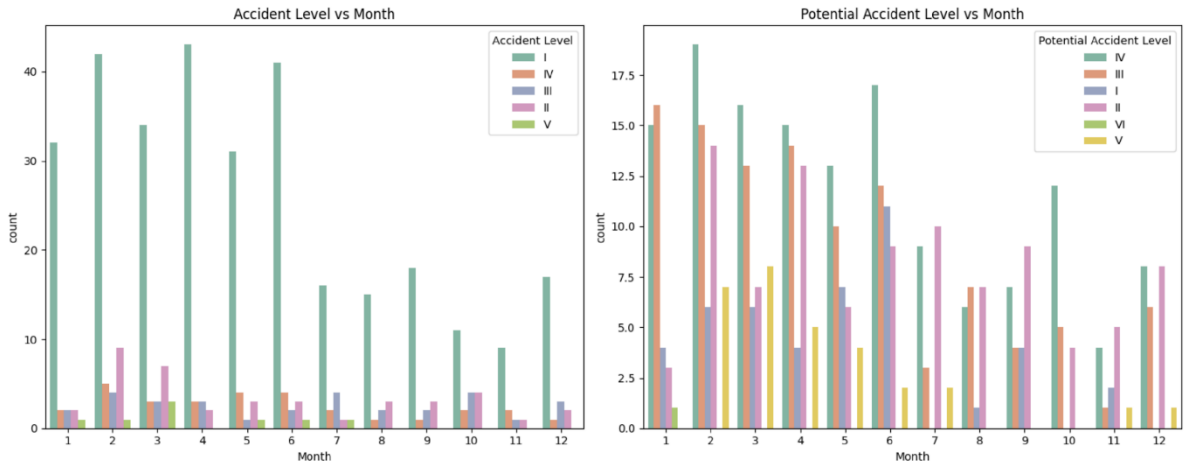


Figure 7 Accident Level Vs Month & Potential Accident Level Vs Month

`

Accident Level vs Month:

1. There's some variation in accident counts across different months, but no clear seasonal pattern emerges.
2. Further analysis might be needed to identify potential factors influencing these monthly fluctuations.

Potential Accident Level vs Month:

1. Similar to the actual accident level, potential accidents also show some monthly variation without a distinct seasonal pattern.
2. This suggests that the factors influencing accident occurrences might not be strongly tied to specific months.

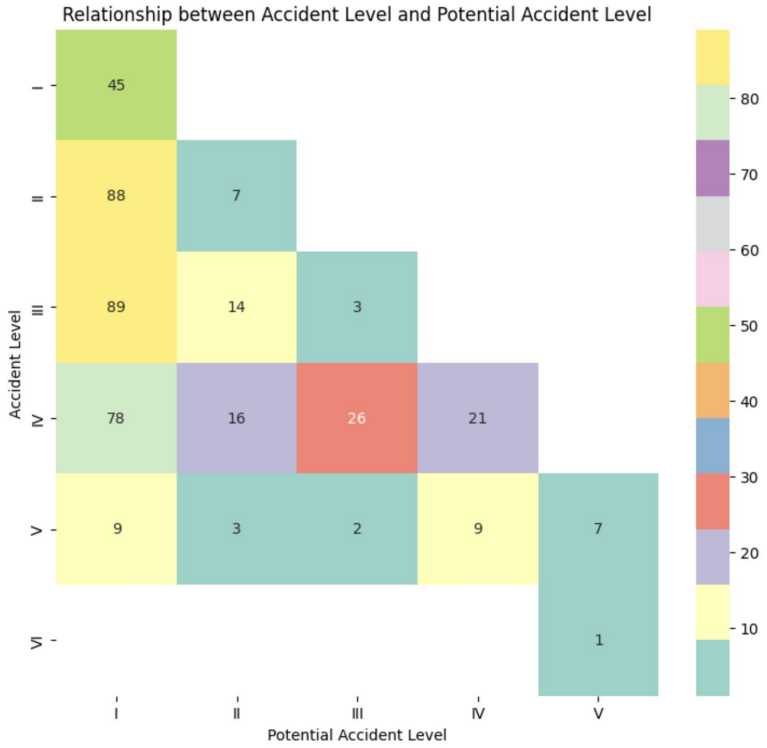


Figure 8 Accident Vs Potential Accident Levels

Observations:

Diagonal Dominance:

1. The heatmap shows a strong diagonal dominance, indicating a positive correlation between Accident Level and Potential Accident Level.
2. This implies that accidents with a higher actual severity level are also more likely to have a higher potential severity level.

`

Potential for Worse Outcomes:

3. There are significant off-diagonal values, especially above the diagonal.
4. This suggests that many accidents that resulted in lower actual severity levels had the potential to be much worse.

Preventive Measures:

5. The difference between actual and potential severity highlights the importance of preventive measures and safety protocols.
6. These measures likely played a role in preventing many accidents from escalating to their full potential severity.

Focus Areas for Improvement:

7. The heatmap can help identify areas where safety measures can be further improved.
8. For example, focusing on accidents with high potential severity but lower actual severity can lead to more effective prevention strategies.

Here the overall detailed analysis:

Monthly frequency of accidents - YoY

Continuously enhance safety protocols by monitoring trends, addressing seasonal risks, and adapting strategies to reduce year-over-year fluctuations in accidents.

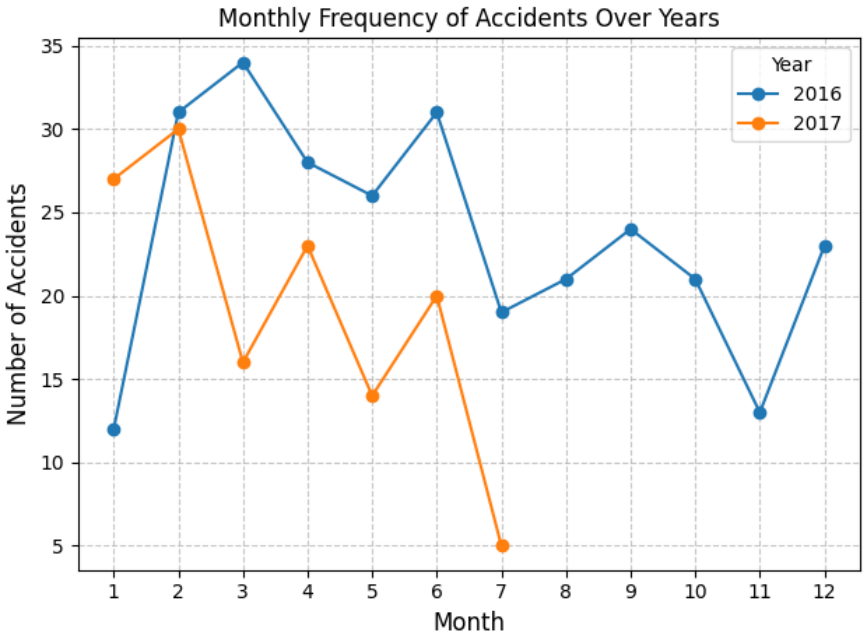


Figure 9 Monthly Frequency of Accidents Over Years

`

Recommendation:

1. **Quarterly Safety Review:** Review and adjust safety protocols at the end of each quarter.
2. **Mid-Year Safety Audit:** Conduct a detailed safety audit in April to mitigate mid-year risks.
3. **Seasonal Safety Campaign: I**mplement targeted safety initiatives from May to July to reduce accidents.
4. **End-of-Year Evaluation:** Analyze annual accident data in December to refine safety strategies for the next year.

Distribution of accident levels across countries

The distribution of accident levels differs by country, indicating possible variations in safety regulations, industry practices, or specific risk factors.

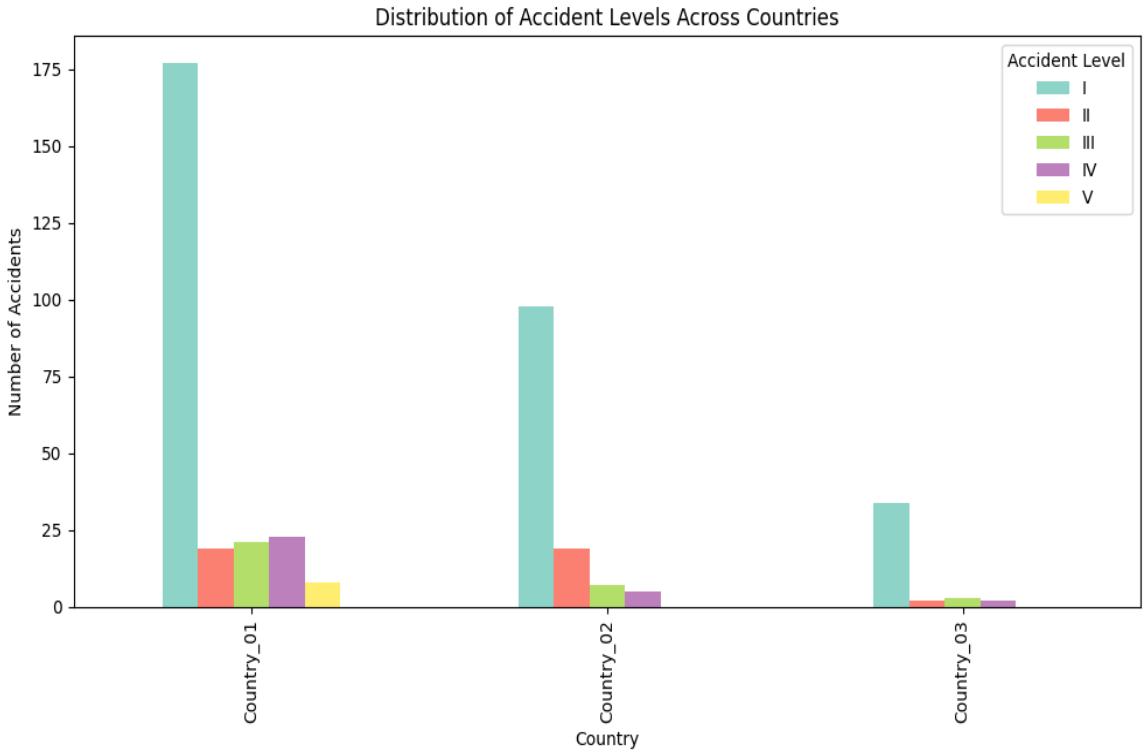


Figure 10 Distribution of accident levels across countries

Recommendation:

1. **Strengthen Safety Measures in Country_01**: Address the consistently high accident rates across all levels.
2. **Reduce Level III Accidents in Country_02**: Identify and mitigate specific risks contributing to more severe accidents.
3. **Adopt Best Practices from Country_03**: Learn from Country_03's effective safety protocols to improve other countries' performance.

`

Accident level vs Industry sectors

Geographic and industry-specific risks dominate, with severity variations and local patterns highlighting the need for targeted safety measures.

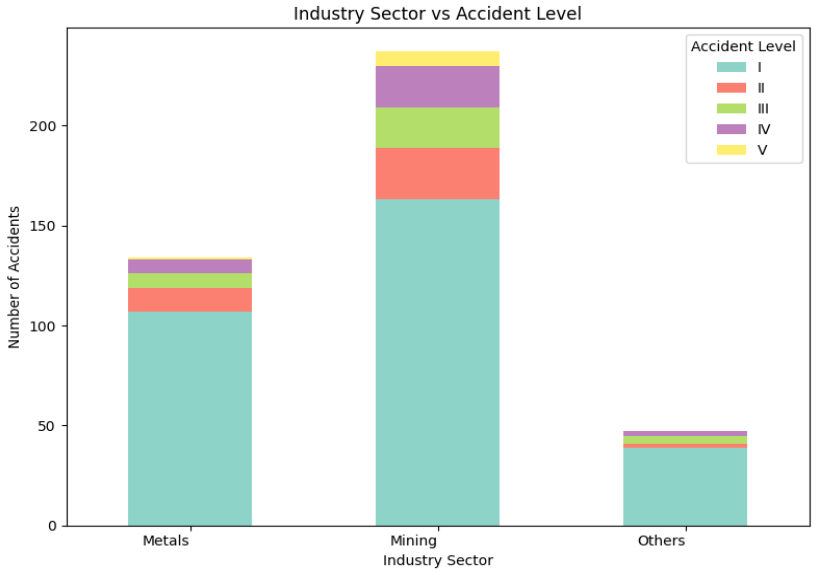


Figure 11 Industry Sector vs Accident Levels

Recommendation:

1. **Enhance Safety Protocols in Mining:** Implement targeted safety interventions in the mining sector to reduce accidents across all severity levels.

2. **Prevent Escalation of Minor Incidents:** Develop strategies to ensure that Level I and II incidents do not progress to more severe accidents across all sectors.

Accident level with respect to Gender/Employee types

Enhancing Safety Measures for Vulnerable Employee Groups

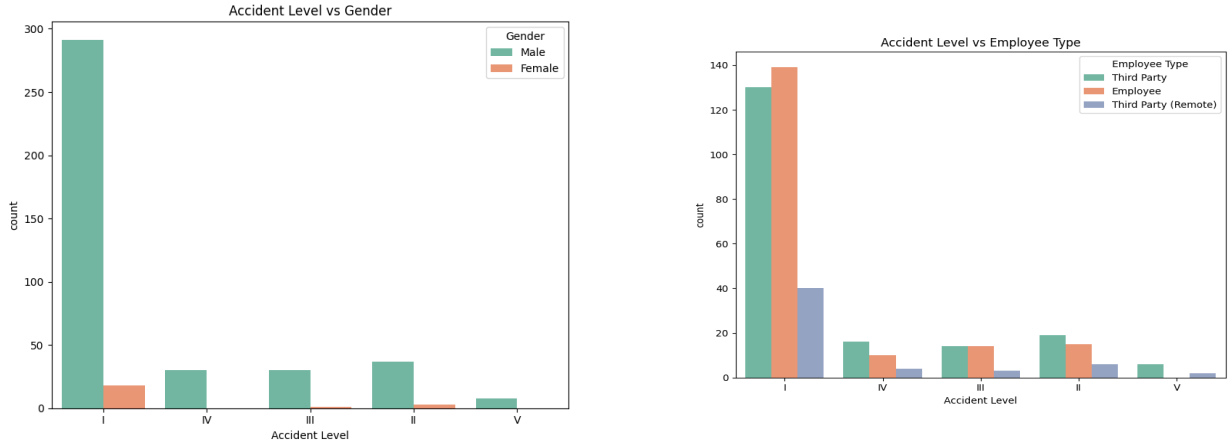


Figure 12 Accident Level Vs Gender/Employee Type

`

Recommendation:

1. **Implement targeted safety training for employees and males** to reduce lower-level accidents (I and II).
2. **Strengthen monitoring and intervention** strategies to minimize the disparity in accident rates among different employee types and genders.

Overall observations

1. **Temporal Trends**: Fluctuations observed; no clear long-term trend but down term trend is observed
2. **Geographic Risk:** Country_01 shows highest accident rates, followed by Country_02 and Country_03, with significant city-level variations.
3. **Industry Risk**: Mining sector has highest accident frequency across all severity levels and locations.
4. **Severity Distribution**: Majority of accidents are Level I and II, but distribution varies by country and city.
5. **Risk Escalation**: Strong correlation between actual and potential accident severity, with many incidents showing potential for worse outcomes.
6. **Country Variations:** Country_02 shows increase in Level III accidents; Country_03 has fewer severe accidents.
7. **City-Specific Patterns:** Unique accident distribution patterns observed across cities, indicating local risk factors.
8. **Demographics:** Male employees and regular staff involved in significantly more accidents.
9. **Preventive Measures**: Gap between actual and potential severity levels indicates effectiveness of safety protocols.

`

6 NLP Pre-Processing

NLP (Natural Language Processing) pre-processing is a crucial step in preparing raw text data for analysis and modelling. It involves various techniques to clean, transform, and structure text data in a way that makes it suitable for machine learning models and other NLP tasks. Here are some common pre-processing steps:

- Tokenization
- Lowercasing
- Removing Punctuation
- Removing Stop Words
- Stemming
- Lemmatization
- Removing Numbers
- Removing Whitespace
- Part-of-Speech Tagging
- Named Entity Recognition (NER)
- Text Normalization
- Handling Special Characters
- Handling Misspellings and Slang

These pre-processing steps help transform raw text data into a structured format that can be easily fed into NLP models for tasks like text classification, sentiment analysis, machine translation, and more. The choice of steps depends on the specific use case and the nature of the text data. The below sections mentions that all the preprocessing is done for the Capstone project

6.1 Word-clouds - Unigrams, Bigrams & Trigrams

A word cloud is a visual representation of text data, where the size of each word indicates its frequency or importance in the source text. Word clouds are commonly used for text analysis and can provide a quick visual summary of the most prominent words in a dataset.

Overall, below steps are performed:

- Initialise lemmatize and stop words
- Convert to lowercase
- Remove special characters
- Tokenize the text
- Remove stop-words
- Apply pre-processing to description column

`

```
# Load the dataset
ISH_NLP_preprocess = pd.read_csv('/content/drive/My Drive/Capstone_Group10_NLP1/ISH_df_preprocess.csv')

# Initialize lemmatizer and stopwords
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()

    # Remove special characters and numbers
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    # Tokenize the text
    tokens = word_tokenize(text)

    # Remove stopwords and lemmatize
    cleaned_tokens = [lemmatizer.lemmatize(token) for token in tokens if token not in stop_words]

    # Join the tokens back into a string
    cleaned_text = ' '.join(cleaned_tokens)

    return cleaned_text

# Apply preprocessing to the Description column
ISH_NLP_preprocess['Cleaned_Description'] = ISH_NLP_preprocess['Description'].apply(preprocess_text)

# Display the first few rows of the original and cleaned descriptions
ISH_NLP_preprocess[['Description', 'Cleaned_Description']].head()

# Save the number of words before and after cleaning
ISH_NLP_preprocess['Original_Word_Count'] = ISH_NLP_preprocess['Description'].apply(lambda x: len(str(x).split()))
ISH_NLP_preprocess['Cleaned_Word_Count'] = ISH_NLP_preprocess['Cleaned_Description'].apply(lambda x: len(str(x).split()))
```

```
ISH_NLP_preprocess[['Description', 'Cleaned_Description']].head()
```

	Description	Cleaned_Description
0	While removing the drill rod of the Jumbo 08 for maintenance, the supervisor proceeds to loosen the support of the intermediate centralizer to facilitate the removal, seeing this the mechanic supports one end on the drill of the equipment to pull with both hands the bar and accelerate the removal from this, at this moment the bar slides from its point of support and tightens the fingers of the mechanic between the drilling bar and the beam of the jumbo.	removing drill rod jumbo maintenance supervisor proceeds loosen support intermediate centralizer facilitate removal seeing mechanic support one end drill equipment pull hand bar accelerate removal moment bar slide point support tightens finger mechanic drilling bar beam jumbo
1	During the activation of a sodium sulphide pump, the piping was uncoupled and the sulfide solution was designed in the area to reach the maid. Immediately she made use of the emergency shower and was directed to the ambulatory doctor and later to the hospital. Note: of sulphide solution = 48 grams / liter.	activation sodium sulphide pump piping uncoupled sulfide solution designed area reach maid immediately made use emergency shower directed ambulatory doctor later hospital note sulphide solution gram liter

```
ISH_NLP_preprocess
```

	Country	City	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee Type	Critical Risk	Description	DayOfWeek	Year	Month	Day	Cleaned_Description	Original_Word_Count	Cleaned_Word_Count
0	Country_01	City_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 for maintenance, the supervisor proceeds to loosen the support of the intermediate centralizer to facilitate the removal, seeing this the mechanic supports one end on the drill of the equipment to pull with both hands the bar and accelerate the removal from this, at this moment the bar slides from its point of support and tightens the fingers of the mechanic between the drilling bar and the beam of the jumbo.	4	2016	1	1	removing drill rod jumbo maintenance supervisor proceeds loosen support intermediate centralizer facilitate removal seeing mechanic support one end drill equipment pull hand bar accelerate removal moment bar slide point support tightens finger mechanic drilling bar beam jumbo	80	37
1	Country_02	City_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pump, the piping was uncoupled and the sulfide solution was designed in the area to reach the maid. Immediately she made use of the emergency shower and was directed to the ambulatory doctor and later to the hospital. Note: of sulphide solution = 48 grams / liter.	5	2016	1	2	activation sodium sulphide pump piping uncoupled sulfide solution designed area reach maid immediately made use emergency shower directed ambulatory doctor later hospital note sulphide solution gram liter	54	27
2	Country_01	City_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170 when the collaborator was doing the excavation work with a pick (hand tool), hitting a rock with the flat part of the beak, it bounces off hitting the steel tip of the safety shoe and then the metatarsal area of the left foot of the collaborator causing the injury.	2	2016	1	6	substation milpo located level collaborator excavation work pick hand tool hitting rock flat part beak bounce hitting steel tip safety shoe metatarsal area left foot collaborator causing injury	57	28
3	Country_01	City_04	Mining	I	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 CX-695 OB7, the personnel begins the task of unlocking the Soquet bolts of the BHB machine, when they were in the penultimate bolt they identified that the hexagonal head was worn, proceeding Mr. Cristóbal - Auxiliary assistant to climb to the platform to exert pressure with your hand on the "DADO" key, to prevent it from coming out of the bolt; in those moments two collaborators rotate with the lever in anti-clockwise direction, leaving the key of the bolt, hitting the palm of the left hand, causing the injury.	4	2016	1	8	approximately nv cx ob personnel begin task unlocking soquet bolt bhb machine penultimate bolt identified hexagonal head worn proceeding mr cristbal auxiliary assistant climb platform exert pressure hand dado key prevent coming bolt moment two collaborator rotate lever anticlockwise direction leaving key bolt hitting palm left hand causing injury	97	49

49.58% reduction in the word count post the following analysis:

```
# Calculate and print the average word count before and after cleaning
avg_original = ISH_NLP_preprocess['Original_Word_Count'].mean()
avg_cleaned = ISH_NLP_preprocess['Cleaned_Word_Count'].mean()
print(f"\nAverage word count before cleaning: {avg_original:.2f}")
print(f"Average word count after cleaning: {avg_cleaned:.2f}")
print(f"Reduction in words: {(avg_original - avg_cleaned) / avg_original * 100:.2f}%")
```

```
Average word count before cleaning: 65.06
Average word count after cleaning: 32.80
Reduction in words: 49.58%
```

From the above snapshot it is very much evident that **almost 50% of reduction of words** have been done in this process.

`

Following steps are performed on pre-processed data:

- Combine all descriptions into a single string
- Tokenize the combined text
- Calculate token distribution
- Create a dataframe from the most common words
- Generate word cloud for unigrams, bigrams and trigrams

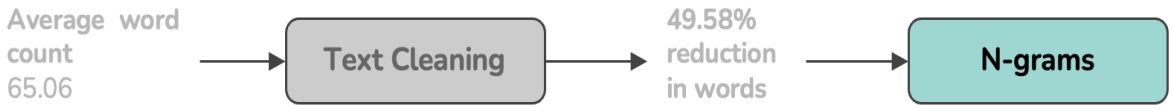
```
# @title Wordcloud for N-Grams

from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Combine all descriptions into a single string
all_text = ' '.join(ISH_NLP_preprocess['Description'].astype(str))

# Generate word cloud for unigrams
wordcloud_unigrams = WordCloud(width=800, height=400, background_color='white').generate(all_text)

# Generate word cloud for bigrams
bigrams = nltk.bigrams(word_tokenize(all_text))
bigram_text = ' '.join(['_'.join(bigram) for bigram in bigrams])
wordcloud_bigrams = WordCloud(width=800, height=400, background_color='white').generate(bigram_text)

# Generate word cloud for trigrams
trigrams = nltk.trigrams(word_tokenize(all_text))
trigram_text = ' '.join(['_'.join(trigram) for trigram in trigrams])
wordcloud_trigrams = WordCloud(width=800, height=400, background_color='white').generate(trigram_text)

# Display the word clouds
plt.figure(figsize=(45, 15))
plt.subplot(1, 3, 1)
plt.imshow(wordcloud_unigrams, interpolation='bilinear')
plt.title('Unigrams')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(wordcloud_bigrams, interpolation='bilinear')
plt.title('Bigrams')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(wordcloud_trigrams, interpolation='bilinear')
plt.title('Trigrams')
plt.axis('off')

plt.show()
```

Figure 13 Unigrams/Bigrams/Trigrams

Note: the observations of above executions has been captured in section 6.2 below.

`

Note: ***N-gram*** identifies accident contributors and areas for safety improvement, helping develop interventions to enhance workplace safety. The below diagram provides more clarity on percentage of text/word reductions during the cleaning process.



6.2 Observations

Unigrams:

1. Keywords include "moment," "employee," "floor," "equipment," "assistant," "left," and "hand."
2. These suggest an incident involving an employee and equipment on a specific floor.
3. Words like "collaboration," "injury," and "support" indicate teamwork and possibly injury response.
4. "Left" near "hand" points to a body part, likely in a workplace injury report.
5. This might relate to a safety analysis or accident report in an industrial setting.

Bigrams:

1. Frequent bigrams like "left hand" and "right hand" indicate a focus on hand and finger injuries.
2. This suggests frequent hand-related injuries in the analyzed data or reports.
3. Other terms like "left leg" and "left foot" appear but are less common.
4. Phrases like "causing injury" and "employee performing" point to work-related injuries.
5. Terms such as "causing cut" and "causing fall" highlight common injury mechanisms.

Trigrams:

1. Trigrams like "left hand causing" and "finger left hand" focus on injuries to the left hand or fingers.
2. Phrases like "used safety glass" suggest the involvement of specific safety measures.
3. The emphasis on hands and fingers shows their vulnerability in the workplace. The analysis details injury causes and is useful for prevention.
4. Words like "operator" and "employee" next to "accident" and "injury" emphasize roles in safety protocols.

Overall:

1. N-grams analysis offers insights into key themes and patterns in incident reports.
2. It identifies accident contributors and areas for safety improvement.
3. The findings could help develop interventions to enhance workplace safety.

`

6.3 Data Preprocessing using Glove, TFI-DF and Word2Vec

Below code is used to generate Word Embeddings over 'Description' column using **Glove, TFI-DF and Word2Vec**.

```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess

def generate_embedding_dataframes(df):
    df1 = df.copy()
    df2 = df.copy()
    df3 = df.copy()

    # 1. GloVe Embeddings
    def load_glove_model(glove_file):
        embedding_dict = {}
        with open(glove_file, 'r', encoding="utf8") as f:
            for line in f:
                values = line.split()
                word = values[0]
                vector = np.asarray(values[1:], "float32")
                embedding_dict[word] = vector
        return embedding_dict

    def get_average_glove_embeddings(tokenized_words, embedding_dict, embedding_dim=300):
        embeddings = [embedding_dict.get(word, np.zeros(embedding_dim)) for word in tokenized_words]
        return np.mean(embeddings, axis=0) if embeddings else np.zeros(embedding_dim)

    # Load GloVe model and generate GloVe embeddings
    glove_file = '/content/drive/MyDrive/Capstone_Group10_NLP1/glove.6B/glove.6B.300d.txt'
    glove_embeddings = load_glove_model(glove_file)

    glove_embeddings_series = df1['tokenized_words'].apply(lambda words: get_average_glove_embeddings(words, glove_embeddings))
    ISH_NLP_Glove_df = pd.concat([df1.drop(columns=['tokenized_words']), pd.DataFrame(glove_embeddings_series.tolist(), columns=[f'GloVe_{i}' for i in range(300)])], axis=1)

    # 2. TF-IDF Features
    tfidf_vectorizer = TfidfVectorizer(tokenizer=lambda x: x, lowercase=False, token_pattern=None)
    tfidf_matrix = tfidf_vectorizer.fit_transform(df2['tokenized_words'])

    # Create a DataFrame with TF-IDF features
    tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf_vectorizer.get_feature_names_out())
    ISH_NLP_TFIDF_df = pd.concat([df2.drop(columns=['tokenized_words']), tfidf_df], axis=1)

    # 3. Word2Vec Embeddings
    word2vec_model = Word2Vec(sentences=df3['tokenized_words'], vector_size=300, window=5, min_count=1, workers=4)

    def get_average_word2vec_embeddings(tokenized_words, model, embedding_dim=300):
        embeddings = [model.wv[word] for word in tokenized_words if word in model.wv]
        return np.mean(embeddings, axis=0) if embeddings else np.zeros(embedding_dim)

    word2vec_embeddings_series = df3['tokenized_words'].apply(lambda words: get_average_word2vec_embeddings(words, word2vec_model))
    ISH_NLP_Word2Vec_df = pd.concat([df3.drop(columns=['tokenized_words']), pd.DataFrame(word2vec_embeddings_series.tolist(), columns=[f'Word2Vec_{i}' for i in range(300)])], axis=1)

    return ISH_NLP_Glove_df, ISH_NLP_TFIDF_df, ISH_NLP_Word2Vec_df

# Use the function to generate the DataFrames
ISH_NLP_Glove_df, ISH_NLP_TFIDF_df, ISH_NLP_Word2Vec_df = generate_embedding_dataframes(ISH_NLP_preprocess1)
```

Output of above execution is below:

```
ISH_NLP_Glove_df
```

	Country	City	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee Type	Critical Risk	DayOfWeek	Year	...	GloVe_290	GloVe_291	GloVe_292	GloVe_293	GloVe_294	GloVe_295	GloVe_296	GloVe_297	GloVe_298	GloVe_299
0	Country_01	City_01	Mining	I	IV	Male	Third Party	Pressed	4	2016	...	-0.034536	-0.110637	-0.085788	-0.031955	0.008084	0.205297	-0.001389	-0.296468	-0.061921	-0.003529
1	Country_02	City_02	Mining	I	IV	Male	Employee	Pressurized Systems	5	2016	...	-0.412660	-0.135541	0.049905	0.032907	0.103431	-0.155970	0.078383	-0.218822	-0.099618	-0.053435
2	Country_01	City_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	2	2016	...	0.005927	-0.135486	-0.016369	0.125184	0.149626	0.194006	0.028868	-0.159949	0.032494	-0.110724
3	Country_01	City_04	Mining	I	I	Male	Third Party	Others	4	2016	...	-0.037377	-0.070661	0.078244	-0.019498	-0.035796	0.246286	-0.105964	-0.115616	-0.050545	-0.049797
4	Country_01	City_04	Mining	IV	IV	Male	Third Party	Others	6	2016	...	0.103048	-0.080292	0.028120	-0.075642	0.116875	0.247585	-0.008106	-0.106944	-0.074254	-0.087914
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
413	Country_01	City_04	Mining	I	III	Male	Third Party	Others	1	2017	...	-0.048683	-0.039020	-0.071929	-0.091603	0.107000	0.385754	-0.140584	-0.078597	0.143009	-0.130202
414	Country_01	City_03	Mining	I	II	Female	Employee	Others	1	2017	...	0.049501	-0.147315	0.041269	0.039820	0.083148	0.199192	-0.086235	-0.224753	0.005231	-0.024155
415	Country_02	City_09	Metals	I	II	Male	Employee	Venomous Animals	2	2017	...	0.058225	-0.122102	-0.121571	0.074627	0.131929	0.145566	0.031812	0.011314	-0.088791	-0.089753
416	Country_02	City_05	Metals	I	II	Male	Employee	Cut	3	2017	...	-0.095062	-0.107262	0.079336	0.124554	0.068740	0.040127	0.048653	-0.123861	0.090110	-0.117909
417	Country_01	City_04	Mining	I	II	Female	Third Party	Fall prevention (same level)	6	2017	...	0.028054	0.010017	-0.083869	-0.013579	0.174762	0.119727	0.049611	-0.257038	-0.052309	-0.065951

418 rows × 313 columns

```
ISH_NLP_TFIDF_df
```

	Country	City	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee Type	Critical Risk	DayOfWeek	Year	...	yolk	young	z	zaf	zamac	zero	zinc	zinco	zn	zone
0	Country_01	City_01	Mining	I	IV	Male	Third Party	Pressed	4	2016	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
1	Country_02	City_02	Mining	I	IV	Male	Employee	Pressurized Systems	5	2016	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
2	Country_01	City_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	2	2016	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
3	Country_01	City_04	Mining	I	I	Male	Third Party	Others	4	2016	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
4	Country_01	City_04	Mining	IV	IV	Male	Third Party	Others	6	2016	...	0.0	0.0	0.0	0.200191	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
413	Country_01	City_04	Mining	I	III	Male	Third Party	Others	1	2017	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
414	Country_01	City_03	Mining	I	II	Female	Employee	Others	1	2017	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
415	Country_02	City_09	Metals	I	II	Male	Employee	Venomous Animals	2	2017	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
416	Country_02	City_05	Metals	I	II	Male	Employee	Cut	3	2017	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
417	Country_01	City_04	Mining	I	II	Female	Third Party	Fall prevention (same level)	6	2017	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0

`

Below table conveys above NLP preprocessing execution results:

Pre-Embedding data shape	Embedding techniques	Post-Embedding data shape
(418, 14)	Glove	(418, 313)
	TFIDF	(418, 2827)
	Word2Vec	(418, 313)

Below steps are performed before data preparations:

- Initialize Label Encoder
- Encode 'Accident Level' and 'Potential Accident Level' in ISH_NLP_Glove_df
- Encode 'Accident Level' and 'Potential Accident Level' in ISH_NLP_TFIDF_df
- Encode 'Accident Level' and 'Potential Accident Level' in ISH_NLP_Word2Vec_df
- Columns to drop
- Drop columns from each Data Frame
- Calculate target variable distribution for each Data Frame

```
from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Encode 'Accident Level' and 'Potential Accident Level' in ISH_NLP_Glove_df
ISH_NLP_Glove_df['Accident Level'] = label_encoder.fit_transform(ISH_NLP_Glove_df['Accident Level'])
ISH_NLP_Glove_df['Potential Accident Level'] = label_encoder.fit_transform(ISH_NLP_Glove_df['Potential Accident Level'])

# Encode 'Accident Level' and 'Potential Accident Level' in ISH_NLP_TFIDF_df
ISH_NLP_TFIDF_df['Accident Level'] = label_encoder.fit_transform(ISH_NLP_TFIDF_df['Accident Level'])
ISH_NLP_TFIDF_df['Potential Accident Level'] = label_encoder.fit_transform(ISH_NLP_TFIDF_df['Potential Accident Level'])

# Encode 'Accident Level' and 'Potential Accident Level' in ISH_NLP_Word2Vec_df
ISH_NLP_Word2Vec_df['Accident Level'] = label_encoder.fit_transform(ISH_NLP_Word2Vec_df['Accident Level'])
ISH_NLP_Word2Vec_df['Potential Accident Level'] = label_encoder.fit_transform(ISH_NLP_Word2Vec_df['Potential Accident Level'])
```

```
# Columns to drop
columns_to_drop = ['Year', 'Month', 'Day', 'Potential Accident Level', 'Description']

# Drop columns from each DataFrame
ISH_NLP_Glove_df = ISH_NLP_Glove_df.drop(columns_to_drop, axis=1)
ISH_NLP_TFIDF_df = ISH_NLP_TFIDF_df.drop(columns_to_drop, axis=1)
ISH_NLP_Word2Vec_df = ISH_NLP_Word2Vec_df.drop(columns_to_drop, axis=1)
```

```
# Calculate target variable distribution for each DataFrame
glove_target_dist = ISH_NLP_Glove_df['Accident Level'].value_counts(normalize=False)
tfidf_target_dist = ISH_NLP_TFIDF_df['Accident Level'].value_counts(normalize=False)
word2vec_target_dist = ISH_NLP_Word2Vec_df['Accident Level'].value_counts(normalize=False)

# Create a DataFrame to display the distributions
target_distribution_df = pd.DataFrame({
    'Glove': glove_target_dist,
    'TF-IDF': tfidf_target_dist,
    'Word2Vec': word2vec_target_dist
})

# Print the DataFrame
target_distribution_df
```

Accident Level	Glove	TF-IDF	Word2Vec
0	309	309	309
1	40	40	40
2	31	31	31
3	30	30	30
4	8	8	8

`

Observations:

Target Variable Distribution:

1. Across all three embedding methods (GloVe, TF-IDF, Word2Vec), the distribution of the target variable "Accident Level" remains consistent.
2. This indicates that the embedding process itself doesn't significantly alter the representation of the target variable.
3. The majority of instances fall under a specific "Accident Level" (likely the most common type of accident), highlighting the imbalanced nature of the dataset.

Implications for Modelling:

1. The imbalanced target distribution suggests the need for addressing class imbalance during model training.
2. Techniques like oversampling, under sampling, or using weighted loss functions might be necessary to improve model performance on minority classes.
3. Careful evaluation metrics (precision, recall, F1-score) should be used to assess model performance on all classes, not just the majority class.

Finally, below steps are performed at end of data pre-processing:

- Check for missing values and duplicates
- Check data quality for each data frame
- Concatenate results into single data frame

`

7 Data Preparation

As part of the data preparations the only step here we perform is exporting all the data frames created for Glove, TFI-DF and Word2Vec into csv and xlsx file. Below snapshot convey the same:

```
# Export the 3 dataframes in csv and xlsx

# Export to CSV
Final_NLP_Glove_df.to_csv('/content/drive/My Drive/Capstone_Group10_NLP1/Final_NLP_Glove_df.csv', index=False)
Final_NLP_TFIDF_df.to_csv('/content/drive/My Drive/Capstone_Group10_NLP1/Final_NLP_TFIDF_df.csv', index=False)
Final_NLP_Word2Vec_df.to_csv('/content/drive/My Drive/Capstone_Group10_NLP1/Final_NLP_Word2Vec_df.csv', index=False)


# Export to Excel
Final_NLP_Glove_df.to_excel('/content/drive/My Drive/Capstone_Group10_NLP1/Final_NLP_Glove_df.xlsx', index=False)
Final_NLP_TFIDF_df.to_excel('/content/drive/My Drive/Capstone_Group10_NLP1/Final_NLP_TFIDF_df.xlsx', index=False)
Final_NLP_Word2Vec_df.to_excel('/content/drive/My Drive/Capstone_Group10_NLP1/Final_NLP_Word2Vec_df.xlsx', index=False)
```

`

8 Design, Train and Test ML Classifier

Designing and training basic machine learning classifiers involves several key steps. Here's a process followed to build and evaluate classifiers, such as Logistic Regression, Decision Trees, and Support Vector Machines (SVM), Random Forest, Gradient Boosting, XGBoost, Naive Bias and K-Nearest Neighbours using the typical machine learning workflow.

8.1 Initialize classifier and invoke train / evaluate function

- Initialise all the known classifiers and run models on the 3 data frames created during the data preparation phase.
- Written a function to train/evaluate models
- Invoke the train/evaluate function
- Capture the results in a separate data frame

```
# Initialize classifiers
classifiers = {
    "Logistic Regression": LogisticRegression(),
    "Support Vector Machine": SVC(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "Gradient Boosting": GradientBoostingClassifier(),
    "XG Boost": XGBClassifier(),
    "Naive Bayes": GaussianNB(),
    "K-Nearest Neighbors": KNeighborsClassifier()
}

# Function to train and evaluate models
def train_and_evaluate(df):
    X = df.drop('Accident Level', axis=1)
    y = df['Accident Level']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    results = []
    for name, clf in classifiers.items():
        start_time = time.time()
        clf.fit(X_train, y_train)
        training_time = time.time() - start_time

        # Train metrics
        y_train_pred = clf.predict(X_train)
        train_accuracy = accuracy_score(y_train, y_train_pred)
        train_precision = precision_score(y_train, y_train_pred, average='weighted')
        train_recall = recall_score(y_train, y_train_pred, average='weighted')
        train_f1 = f1_score(y_train, y_train_pred, average='weighted')

        start_time = time.time()
        y_test_pred = clf.predict(X_test)
        prediction_time = time.time() - start_time

        # Test metrics
        test_accuracy = accuracy_score(y_test, y_test_pred)
        test_precision = precision_score(y_test, y_test_pred, average='weighted')
        test_recall = recall_score(y_test, y_test_pred, average='weighted')
        test_f1 = f1_score(y_test, y_test_pred, average='weighted')

        results.append([name,
                        train_accuracy, train_precision, train_recall, train_f1,
                        test_accuracy, test_precision, test_recall, test_f1,
                        training_time, prediction_time])

    return results

# Train and evaluate on each DataFrame
glove_results = train_and_evaluate(Final_NLP_Glove_df)
tfidf_results = train_and_evaluate(Final_NLP_TFIDF_df)
word2vec_results = train_and_evaluate(Final_NLP_Word2Vec_df)
```

`

```
# Train and evaluate on each DataFrame
glove_results = train_and_evaluate(Final_NLP_Glove_df)
tfidf_results = train_and_evaluate(Final_NLP_TFIDF_df)
word2vec_results = train_and_evaluate(Final_NLP_Word2Vec_df)

# Create DataFrames for results
columns = ['Classifier',
           'Train Accuracy', 'Train Precision', 'Train Recall', 'Train F1-score',
           'Test Accuracy', 'Test Precision', 'Test Recall', 'Test F1-score',
           'Training Time', 'Prediction Time']

glove_df = pd.DataFrame(glove_results, columns=columns)
tfidf_df = pd.DataFrame(tfidf_results, columns=columns)
word2vec_df = pd.DataFrame(word2vec_results, columns=columns)
```

8.2 Classification Results

Classification matrix for Glove

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time
0	Logistic Regression	0.963592	0.963436	0.963592	0.963494	0.928803	0.933631	0.928803	0.929641	0.126808	0.005065
1	Support Vector Machine	0.962783	0.963127	0.962783	0.962850	0.912621	0.925272	0.912621	0.914822	0.208429	0.093478
2	Decision Tree	0.999191	0.999194	0.999191	0.999191	0.883495	0.880778	0.883495	0.878947	0.440358	0.003123
3	Random Forest	0.999191	0.999194	0.999191	0.999191	0.990291	0.990464	0.990291	0.990265	1.671161	0.013798
4	Gradient Boosting	0.999191	0.999194	0.999191	0.999191	0.970874	0.971015	0.970874	0.970540	74.470469	0.007030
5	XG Boost	0.999191	0.999194	0.999191	0.999191	0.974110	0.974697	0.974110	0.973937	2.941567	0.069770
6	Naive Bayes	0.576052	0.686802	0.576052	0.555990	0.576052	0.619135	0.576052	0.560298	0.009056	0.005299
7	K-Nearest Neighbors	0.850324	0.875346	0.850324	0.825762	0.838188	0.862608	0.838188	0.798293	0.004603	0.019504

```
print("Classification matrix for TFIDF")
tfidf_df
```

Classification matrix for TFIDF

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time
0	Logistic Regression	0.983819	0.983927	0.983819	0.983854	0.948220	0.954307	0.948220	0.949603	15.831049	0.048633
1	Support Vector Machine	0.979773	0.980294	0.979773	0.979849	0.925566	0.943783	0.925566	0.929372	1.348215	0.619280
2	Decision Tree	0.999191	0.999194	0.999191	0.999191	0.860841	0.863785	0.860841	0.861183	0.224136	0.016699
3	Random Forest	0.999191	0.999194	0.999191	0.999191	0.977346	0.979500	0.977346	0.977712	0.622928	0.028605
4	Gradient Boosting	0.999191	0.999194	0.999191	0.999191	0.919094	0.929961	0.919094	0.922106	28.802572	0.021820
5	XG Boost	0.999191	0.999194	0.999191	0.999191	0.944984	0.955475	0.944984	0.946983	5.141534	0.517637
6	Naive Bayes	0.999191	0.999194	0.999191	0.999191	0.970874	0.973284	0.970874	0.971391	0.067816	0.033026
7	K-Nearest Neighbors	0.859223	0.881303	0.859223	0.841769	0.844660	0.845034	0.844660	0.821537	0.034786	0.045716

```
print("Classification matrix for Wor2Vec")
word2vec_df
```

Classification matrix for Wor2Vec

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time
0	Logistic Regression	0.679612	0.678206	0.679612	0.675876	0.644013	0.639175	0.644013	0.632709	0.162962	0.005017
1	Support Vector Machine	0.757282	0.760561	0.757282	0.752859	0.692557	0.705498	0.692557	0.683242	0.207278	0.095707
2	Decision Tree	0.999191	0.999194	0.999191	0.999191	0.815534	0.804061	0.815534	0.805937	0.494382	0.003046
3	Random Forest	0.999191	0.999194	0.999191	0.999191	0.961165	0.961158	0.961165	0.961090	1.732327	0.014183
4	Gradient Boosting	0.999191	0.999194	0.999191	0.999191	0.961165	0.961879	0.961165	0.959731	72.054121	0.007030
5	XG Boost	0.999191	0.999194	0.999191	0.999191	0.964401	0.964198	0.964401	0.963557	4.281512	0.072283
6	Naive Bayes	0.529935	0.593576	0.529935	0.513007	0.537217	0.579248	0.537217	0.527450	0.008865	0.005488
7	K-Nearest Neighbors	0.839806	0.850000	0.839806	0.829815	0.770227	0.759622	0.770227	0.757096	0.004843	0.019435

`

Overall Performance:

Below 2 models provide the best balance between performance and computational efficiency across the embedding methods.

a. **Random Forest with Glove Embeddings:** Offers an excellent balance of accuracy, recall, and efficiency. It performs consistently well across all embedding methods, with the best performance using Glove embeddings. With TF-IDF embeddings, Random Forest has better performance(Train time-0.62 s vs Pred time-0.0286 s) but with slight compromise on Test accuracy (97.95%)

Train vs Test Accuracy: 99.91% vs 99.02%
Train vs Test Recall: 99.91% vs 99.02%
Train vs prediction time: 1.67s vs 0.013s

b. **XGBoost with GloVe Embeddings:** Slightly better generalization with GloVe embeddings than other models. It is also efficient in both training and prediction times, making it a strong choice for deployment.

Train vs Test Accuracy: 99.91% vs 97.41%
Train vs Test Recall: 99.91% vs 97.41%
Train vs prediction time: 2.94s vs 0.069s

Embedding Comparison:

1. **TF-IDF embeddings** generally yield the highest accuracy and F1-scores across most classifiers, making them the most effective embedding technique in this comparison.
2. **GloVe embeddings** provide strong performance, particularly with Random Forest and XGBoost models, though slightly below TF-IDF in terms of overall metrics.
3. **Word2Vec embeddings** tend to result in lower performance across most classifiers, especially when compared to TF-IDF and GloVe, indicating they might not be the best choice for this dataset.

Best Performers:

1. **For GloVe:** Random Forest, XGBoost, and Gradient Boosting are the top performers with test accuracy around 97-99%.
   Logistic Regression also performs well with a test accuracy of approximately 93%.

2. **For TF-IDF:** Random Forest and Naive Bayes achieve test accuracy around 97-98%, making them strong contenders.
   XGBoost and Logistic Regression also deliver high test accuracy in the range of 94-95%.

3. **For Word2Vec:** Random Forest and XGBoost outperform other classifiers with test accuracy around 96-97%.

`

Worst Performers:

1. **Naive Bayes** consistently underperforms across all embeddings, with particularly poor results for Word2Vec and GloVe, showing low test accuracy.

2. **K-Nearest Neighbours** also shows lower performance in TF-IDF and GloVe embeddings, with test accuracy significantly lower than other models.

Training vs. Testing:

1. **Training performance** is generally high across all models, with many achieving near-perfect accuracy, while **testing performance** shows greater variation, with models like Random Forest and XGBoost maintaining strong accuracy, and others like Naive Bayes and K-Nearest Neighbors struggling to generalize well. The gap between training and testing accuracy highlights differences in model overfitting and generalization.

Efficiency:

1. Logistic Regression and Naive Bayes generally have the shortest training and prediction times.
2. Gradient Boosting has notably long training times, especially for Glove and Word2Vec embeddings.

Consistency:

1. Support Vector Machine shows relatively consistent performance across all three embedding types.
2. Random Forest maintains high accuracy and F1-scores regardless of the embedding used.

Trade-offs:

1. There's a clear trade-off between performance and computational time for some models (e.g., Gradient Boosting).
2. Simpler models like Logistic Regression offer decent performance with much faster training and prediction times.

`

8.3 Confusion matrix against all classifiers

```
# Function to plot confusion matrix against all classifiers with word embeddings generated using Glove, TF-IDF, Word2Vec:

import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

def plot_confusion_matrices(df, df_name):
  X = df.drop('Accident Level', axis=1)
  y = df['Accident Level']
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

  fig, axes = plt.subplots(2, 4, figsize=(20, 10))
  fig.suptitle(f'Confusion Matrices for {df_name}', fontsize=16)

  for i, (name, clf) in enumerate(classifiers.items()):
    row = i // 4
    col = i % 4
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
    disp.plot(ax=axes[row, col], cmap='Oranges')
    axes[row, col].set_title(name)

  plt.tight_layout()
  plt.show()
```

```
plot_confusion_matrices(Final_NLP_Glove_df, 'Glove Embeddings')
```

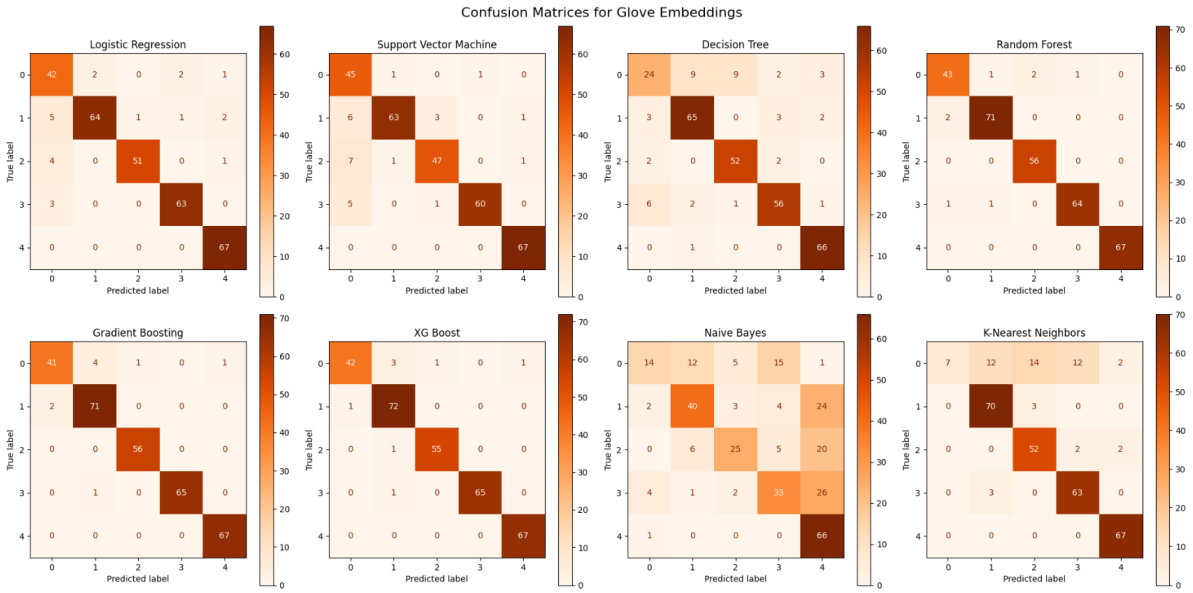


Figure 14 Confusion Matrices for Glove Embeddings

`

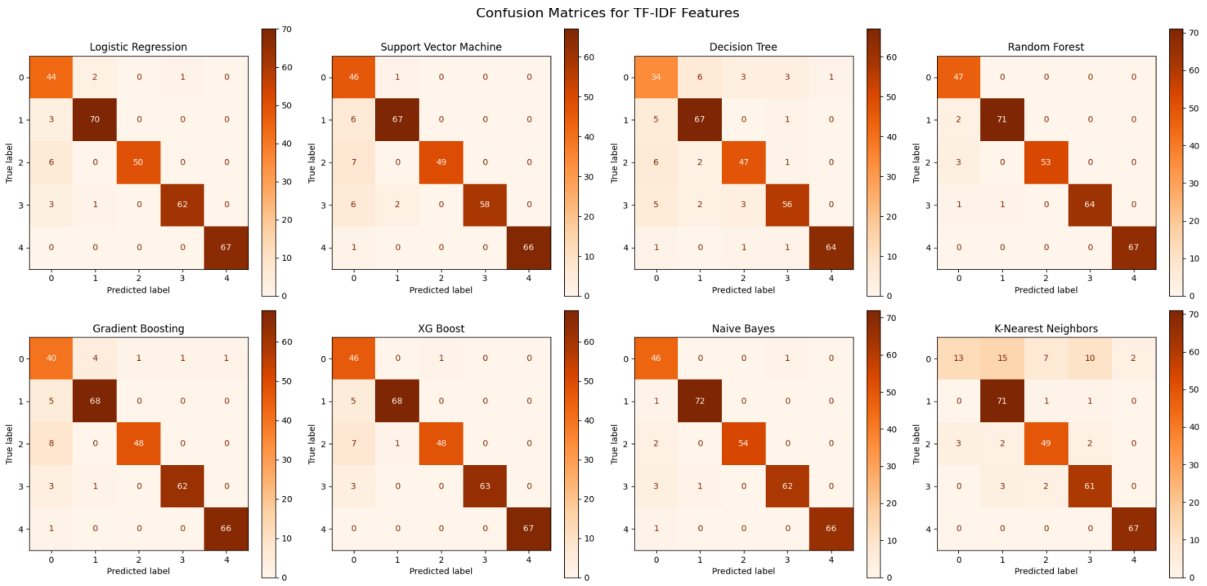


Figure 15 Confusion Matrices for TF-IDF Features

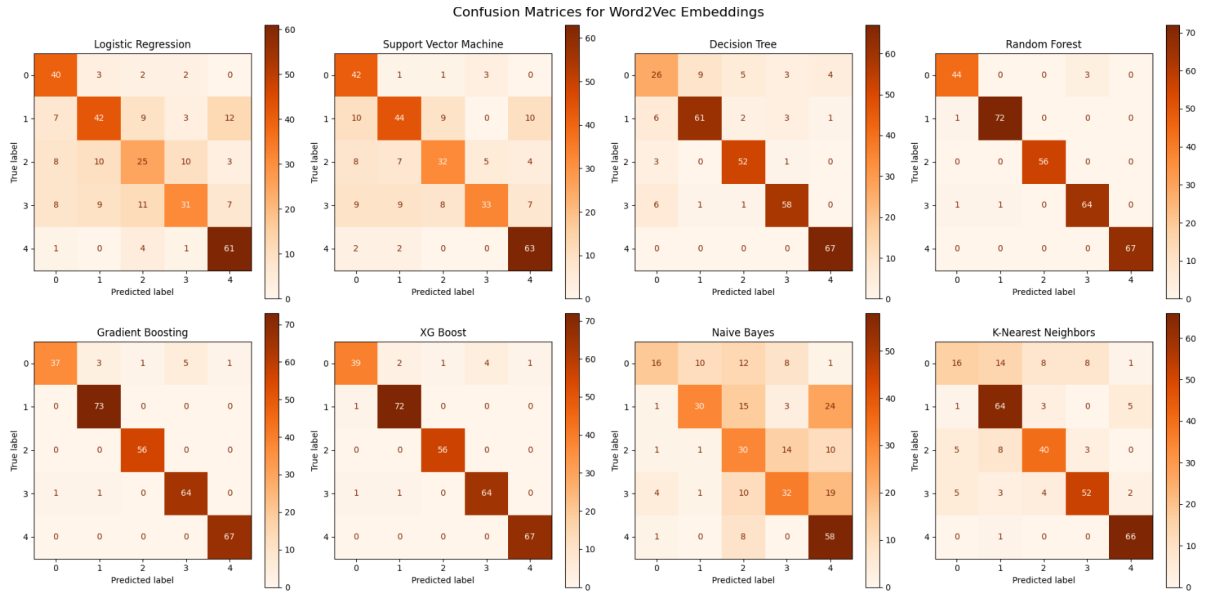


Figure 16 Confusion Matrices for Word2Vec Embeddings

8.4 Confusion Matrices Observations

Overall Performance:

1. Glove Embeddings and TF-IDF Features generally perform better than Word2Vec Embeddings across most models.
2. The diagonal elements (correct predictions) are typically higher for Glove and TF-IDF compared to Word2Vec.

`

Model Comparison:

1. Random Forest and XG Boost consistently perform well across all three feature types.
2. Logistic Regression and Support Vector Machine show good performance, especially with Glove and TF-IDF.
3. Naive Bayes performs poorly across all feature types, showing more misclassifications.
4. K-Nearest neighbour's shows mixed results, performing better with Glove and TF-IDF than with Word2Vec.

Feature-specific Observations:

1. Glove Embeddings: Show strong performance across most models, with high accuracy in the diagonal elements.
2. TF-IDF Features: Perform similarly to Glove Embeddings, sometimes slightly better in certain models.
3. Word2Vec Embeddings: Generally show more misclassifications and lower accuracy compared to the other two.

Class-specific Performance:

1. Classes 0 and 4 (likely representing extreme sentiments) are generally classified more accurately across all feature types and models.
2. The middle classes (1, 2, 3) show more confusion, especially in Word2Vec embeddings.

Misclassification Patterns:

1. In Word2Vec, there's a notable tendency for misclassifications to occur between adjacent classes (e.g., 1 being classified as 0 or 2).
2. Glove and TF-IDF show fewer off-diagonal elements, indicating fewer misclassifications.

Model Stability:

1. Random Forest and XG Boost show more consistent performance across all feature types, suggesting they might be more robust to differences in feature representation.
2. Decision Trees show more variability in performance across feature types. Extreme vs. Neutral Sentiments:

Conclusion:

1. Glove and TF-IDF generally outperform Word2Vec, with Random Forest and XG Boost showing consistent strong performance across all feature types.

`

8.5 PCA and Scaling

Applying **Principal Component Analysis (PCA)** and **scaling** is a common technique in machine learning to reduce the dimensionality of the dataset while retaining as much variance as possible. PCA transforms the data into a set of linearly uncorrelated components (principal components) and is often used after scaling because PCA is sensitive to the variances of the original features.

```
# Apply PCA and scaling

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

def apply_pca_and_split(df, n_components=0.99):
  X = df.drop('Accident Level', axis=1)
  y = df['Accident Level']

  # Scaling
  scaler = StandardScaler()
  X_scaled = scaler.fit_transform(X)

  # PCA
  if n_components < 1:
    pca = PCA(n_components=n_components)
    X_pca = pca.fit_transform(X_scaled)
  else:
    X_pca = X_scaled

  # Splitting
  X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42)

  return X_train, X_test, y_train, y_test

# Apply to each dataframe
X_train_glove, X_test_glove, y_train_glove, y_test_glove = apply_pca_and_split(Final_NLP_Glove_df)
X_train_tfidf, X_test_tfidf, y_train_tfidf, y_test_tfidf = apply_pca_and_split(Final_NLP_TFIDF_df)
X_train_word2vec, X_test_word2vec, y_train_word2vec, y_test_word2vec = apply_pca_and_split(Final_NLP_Word2Vec_df)
```

```
# Function to print explained variance rtio and cumulative explained variance for all 3 embeddings

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

def print_pca_variance(df, df_name):
  X = df.drop('Accident Level', axis=1)

  # Scaling
  scaler = StandardScaler()
  X_scaled = scaler.fit_transform(X)

  # PCA
  pca = PCA()
  pca.fit(X_scaled)

  # Explained variance ratio and cumulative explained variance
  explained_variance_ratio = pca.explained_variance_ratio_
  cumulative_explained_variance = np.cumsum(explained_variance_ratio)

  print(f"----- PCA Variance for {df_name} -----")
  print("Explained Variance Ratio:", explained_variance_ratio)
  print("Cumulative Explained Variance:", cumulative_explained_variance)

# Print PCA variance for each dataframe
print_pca_variance(Final_NLP_Glove_df, 'Glove Embeddings')
print_pca_variance(Final_NLP_TFIDF_df, 'TF-IDF Features')
print_pca_variance(Final_NLP_Word2Vec_df, 'Word2Vec Embeddings')
```

`

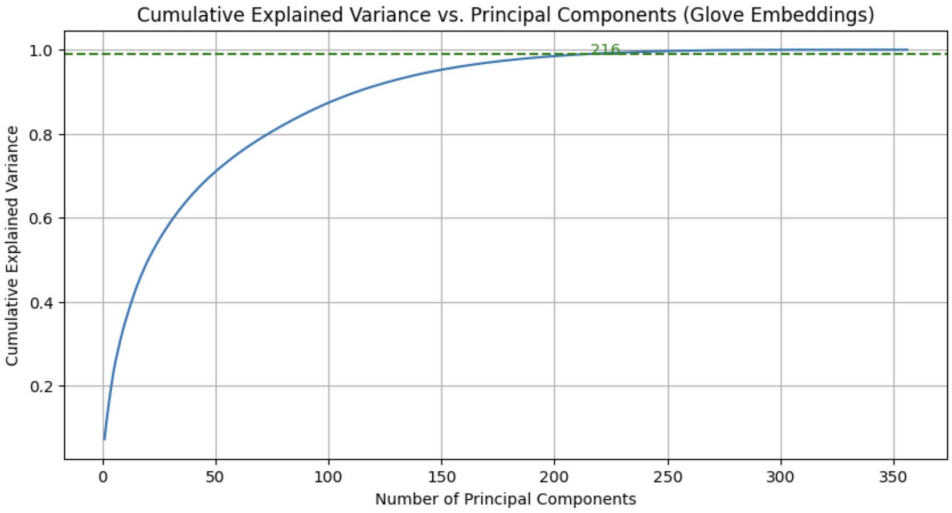


Figure 17 Cumulative Explained Variance Vs PCA (Glove Embeddings)

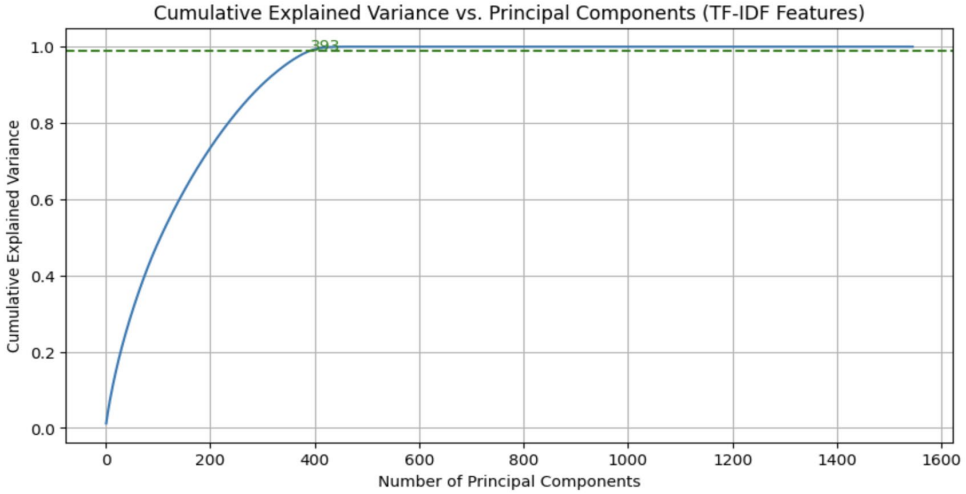


Figure 18 Cumulative Explained Variance Vs PCA (TF-IDF Features)

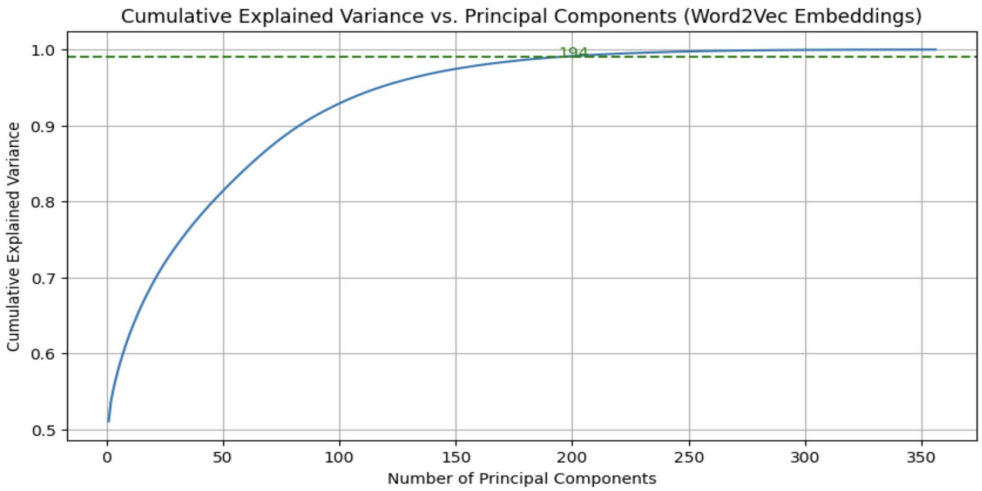


Figure 19 Cumulative Explained Variance Vs PCA (Word2Vec Embeddings)

`

Train and evaluate the classifiers with PCA Component:

```
# Train and evaluate on each PCA-transformed dataset
glove_results_pca = train_and_evaluate_pca(X_train_glove, X_test_glove, y_train_glove, y_test_glove)
tfidf_results_pca = train_and_evaluate_pca(X_train_tfidf, X_test_tfidf, y_train_tfidf, y_test_tfidf)
word2vec_results_pca = train_and_evaluate_pca(X_train_word2vec, X_test_word2vec, y_train_word2vec, y_test_word2vec)

# Create DataFrames for results
columns = ['Classifier',
           'Train Accuracy', 'Train Precision', 'Train Recall', 'Train F1-score',
           'Test Accuracy', 'Test Precision', 'Test Recall', 'Test F1-score',
           'Training Time', 'Prediction Time']

glove_df_pca = pd.DataFrame(glove_results_pca, columns=columns)
tfidf_df_pca = pd.DataFrame(tfidf_results_pca, columns=columns)
word2vec_df_pca = pd.DataFrame(word2vec_results_pca, columns=columns)
```

```
print("Classification matrix for Glove (PCA)")
glove_df_pca
```

Classification Report for PCA:

```
Classification matrix for Glove (PCA)
```

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time
0	Logistic Regression	0.999191	0.999194	0.999191	0.999191	0.948220	0.948514	0.948220	0.948278	0.105575	0.000378
1	Support Vector Machine	0.993528	0.993549	0.993528	0.993527	0.967638	0.970904	0.967638	0.968258	0.167613	0.056506
2	Decision Tree	0.999191	0.999194	0.999191	0.999191	0.776699	0.777464	0.776699	0.774070	0.329336	0.000332
3	Random Forest	0.999191	0.999194	0.999191	0.999191	0.954693	0.959536	0.954693	0.955472	1.775282	0.011454
4	Gradient Boosting	0.999191	0.999194	0.999191	0.999191	0.980583	0.982138	0.980583	0.980819	53.698463	0.004048
5	XG Boost	0.999191	0.999194	0.999191	0.999191	0.970874	0.973752	0.970874	0.971284	1.436882	0.003529
6	Naive Bayes	0.907767	0.909527	0.907767	0.906243	0.834951	0.845118	0.834951	0.835399	0.003593	0.001635
7	K-Nearest Neighbors	0.842233	0.873798	0.842233	0.806267	0.877023	0.900970	0.877023	0.844946	0.000774	0.003505

```
print("\nClassification matrix for TFIDF (PCA)")
tfidf_df_pca
```

```
Classification matrix for TFIDF (PCA)
```

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time
0	Logistic Regression	0.998382	0.998385	0.998382	0.998382	0.957929	0.959284	0.957929	0.956248	0.119481	0.000438
1	Support Vector Machine	0.987864	0.988173	0.987864	0.987872	0.993528	0.993700	0.993528	0.993541	0.188764	0.080477
2	Decision Tree	0.999191	0.999194	0.999191	0.999191	0.906149	0.903889	0.906149	0.903676	0.600765	0.000409
3	Random Forest	0.999191	0.999194	0.999191	0.999191	0.983819	0.984182	0.983819	0.983635	2.135473	0.011151
4	Gradient Boosting	0.999191	0.999194	0.999191	0.999191	0.983819	0.984328	0.983819	0.983830	97.498487	0.003880
5	XG Boost	0.999191	0.999194	0.999191	0.999191	0.977346	0.977723	0.977346	0.977362	4.031977	0.001467
6	Naive Bayes	0.789644	0.819981	0.789644	0.779825	0.786408	0.810530	0.786408	0.784380	0.005635	0.002616
7	K-Nearest Neighbors	0.851133	0.908750	0.851133	0.830634	0.851133	0.907750	0.851133	0.801892	0.000835	0.004656

```
print("\nClassification matrix for Word2Vec (PCA)")
word2vec_df_pca
```

```
Classification matrix for Word2Vec (PCA)
```

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time
0	Logistic Regression	0.999191	0.999194	0.999191	0.999191	0.941748	0.943847	0.941748	0.942467	0.095108	0.000333
1	Support Vector Machine	0.987864	0.987891	0.987864	0.987851	0.964401	0.969988	0.964401	0.965378	0.140915	0.054962
2	Decision Tree	0.999191	0.999194	0.999191	0.999191	0.805825	0.807284	0.805825	0.805828	0.321802	0.000337
3	Random Forest	0.999191	0.999194	0.999191	0.999191	0.948220	0.956001	0.948220	0.949430	1.656468	0.011492
4	Gradient Boosting	0.999191	0.999194	0.999191	0.999191	0.970874	0.972674	0.970874	0.971157	48.383260	0.003917
5	XG Boost	0.999191	0.999194	0.999191	0.999191	0.967638	0.969718	0.967638	0.967933	1.549905	0.001610
6	Naive Bayes	0.907767	0.911257	0.907767	0.907027	0.844660	0.856784	0.844660	0.844113	0.003453	0.001562
7	K-Nearest Neighbors	0.869741	0.887881	0.869741	0.848454	0.867314	0.884397	0.867314	0.833406	0.000745	0.002465

`

Observations:

Overall Impact of PCA:

1. **Improved Efficiency:** PCA significantly reduces training and prediction times for all classifiers by reducing dimensionality, which makes the models faster and more efficient.
2. **Mixed Performance Impact:** PCA generally maintains or slightly improves test performance metrics (accuracy, precision, recall, F1-score) for classifiers using embeddings, with notable enhancements in **Gradient Boosting** and **XG Boost** but less impact on others like **Decision Tree** and **Naive Bayes**.

Embedding-specific Effects:

1. **Glove:** Performance drop was minimal; **Random Forest and XGBoost remained top performers**.
2. **TFIDF:** PCA notably improves the performance of **Gradient Boosting with TFIDF embeddings, enhancing Test Accuracy from 91.91% to 98.38%.**
   Larger performance decrease, especially for Naive Bayes (Test accuracy - 99.91% (Without PCA) to 78.64%(With PCA).
3. **Word2Vec:** Least affected by PCA; Random Forest maintained high performance.

Model-specific Impacts:

1. **Random Forest and XGBoost:** Consistently high performers with and without PCA.
2. **Naive Bayes:** Performance varied greatly depending on embedding type and PCA application.
3. **K-Nearest Neighbours:** Improved relative performance with PCA, especially for Word2Vec.

Efficiency Gains:

1. PCA significantly reduced training and prediction times for all models.
   For example: a) training time for XG Boost with TFIDF embeddings decreased from 5.14 seconds to 1.44 seconds with PCA.
   b) prediction time for Random Forest with TFIDF embeddings dropped from 0.62 seconds to 0.02 seconds with PCA.
2. Gradient Boosting saw the most dramatic reduction in training time.

Overfitting Reduction:

1. **PCA helped reduce overfitting in some cases, particularly for Decision Trees.**
   For example:
   a) Random Forest with Glove embeddings saw test accuracy of 99.03% compared to a near-perfect training accuracy of 99.92%, indicating better generalization.
   b) XG Boost with PCA and TFIDF embeddings achieved a test accuracy of 97.73% compared to 94.50% without PCA.

Trade-off:

1. PCA offers a trade-off between slightly reduced accuracy and significantly improved computational efficiency.

`

Confusion Matrices (Base Classifiers with PCA):

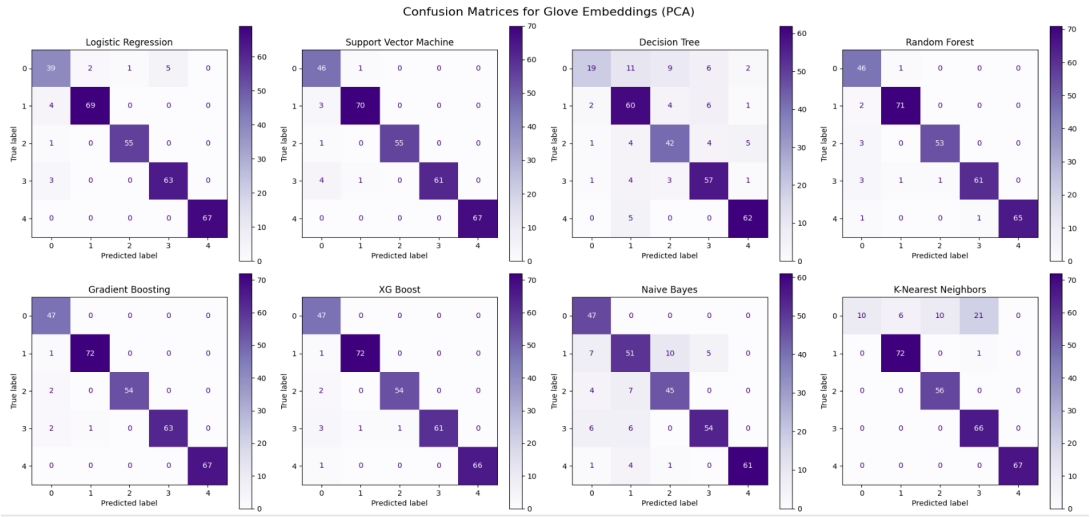


Figure 20 Confusion Matrices for Glove Embeddings (PCA)

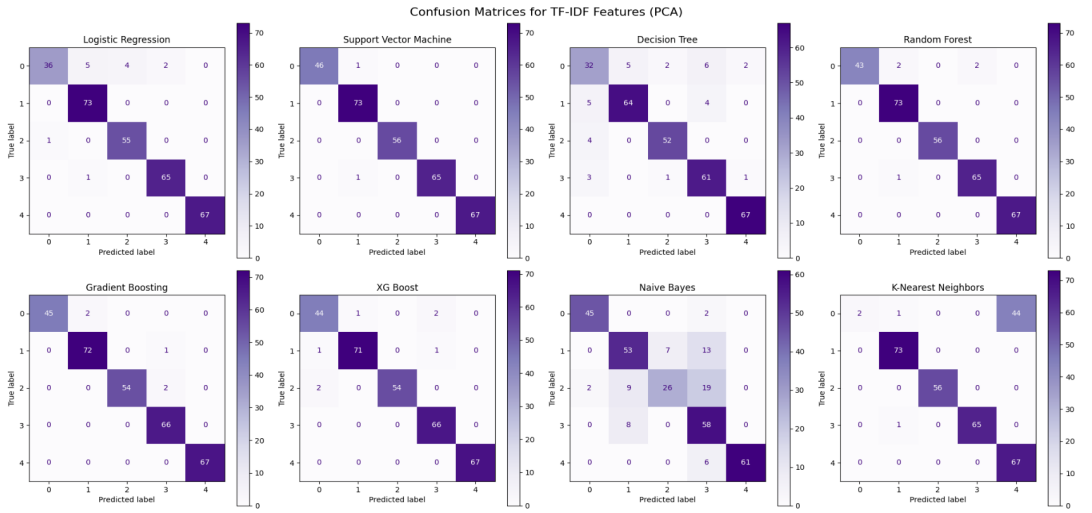


Figure 21 Confusion Matrices for TF-IDF Features (PCA)

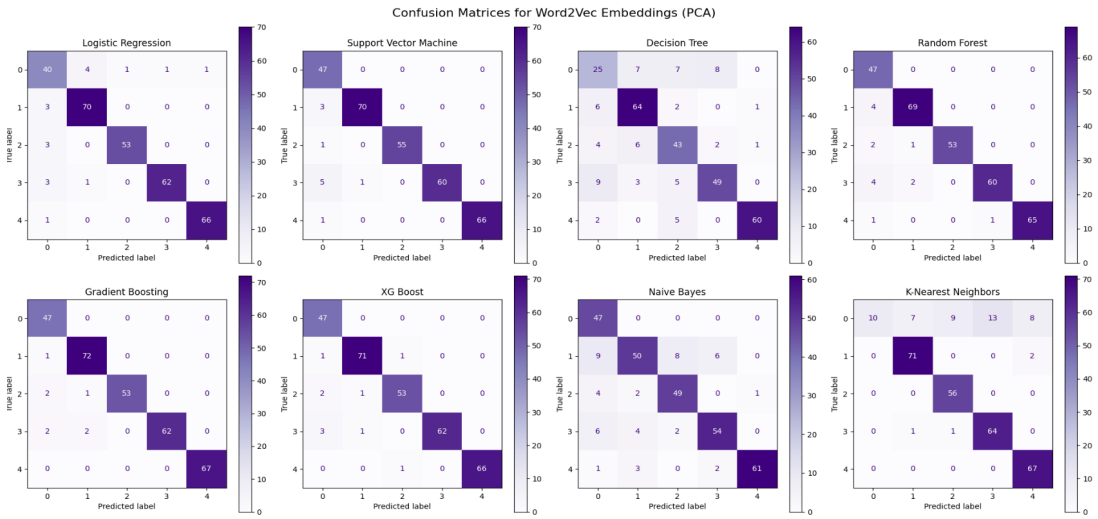


Figure 22 Confusion Matrices for Word2Vec Embeddings (PCA)

`

8.6 Confusion Matrix Observations (Base Classifier + PCA)

Performance Consistency:

1. Models like Gradient Boosting, XG Boost, and Random Forest consistently show strong performance across all embeddings, with high diagonal values indicating correct classifications.

Weak Performers:

1. Decision Tree and Naive Bayes generally show weaker performance, with more spread across non-diagonal cells, indicating misclassifications.
2. K-NN: This model shows varying performance across different embeddings, suggesting its sensitivity to the type of data representation.

Insights for Each Embedding:

GloVe Embeddings (PCA):

1. High Performance: SVM and Random Forest show particularly strong performance.
2. Naive Bayes and K-NN Struggles: These models have more pronounced off-diagonal values, indicating struggles in correctly classifying certain classes.

TF-IDF Features (PCA):

1. **Logistic Regression:** Shows improved performance compared to other embeddings.
2. **SVM:** Continues to perform well, similar to its performance with GloVe.
3. **Decision Tree:** Shows relatively better performance than with GloVe but still lags behind other models.

Word2Vec Embeddings (PCA):

1. **Gradient Boosting and XG Boost:** These models perform well, particularly in classifying middle classes (1, 2, 3).
2. **Naive Bayes:** Shows significant misclassification, especially for classes 1 and 2.
3. **K-NN:** Exhibits a unique pattern with relatively even spread across classes, suggesting confusion in distinguishing between classes.

Comparative Analysis:

1. **Impact of PCA:** PCA seems to have a varying impact on different models and embeddings. For instance, models like SVM and Random Forest are less affected by the reduction in dimensionality, maintaining high accuracy. In contrast, Naive Bayes and K-NN show more sensitivity to these changes.
2. **Best Overall Models:** Gradient Boosting and XG Boost generally offer the best performance across all types of embeddings when PCA is applied, indicating their robustness to changes in input data dimensionality.

`

3. **Model Sensitivity:** Decision Tree and Naive Bayes exhibit more variability and generally lower performance, suggesting that these models may be more sensitive to the type of data representation and dimensionality reduction.

8.7 Hyper-tuning the Base Models with PCA

```
print("Glove Results (with Hypertuning & PCA)")
display(glove_results_ht_pca)
```

Glove Results (with Hypertuning & PCA)

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time	Best Parameters
0	Logistic Regression	0.998382	0.998385	0.998382	0.998382	0.954693	0.954891	0.954693	0.954703	77.103474	0.000402	{'C': 0.1, 'max_iter': 500, 'penalty': 'l2', 'solver': 'saga'}
1	Support Vector Machine	0.996764	0.996777	0.996764	0.996762	0.964401	0.968638	0.964401	0.965181	2.277558	0.057942	{'C': 10, 'class_weight': 'balanced', 'gamma': 'auto', 'kernel': 'rbf'}
2	Decision Tree	0.988673	0.988722	0.988673	0.988671	0.802589	0.816676	0.802589	0.807753	14.266272	0.000420	{'criterion': 'entropy', 'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 5}
3	Random Forest	0.999191	0.999194	0.999191	0.999191	0.961165	0.965057	0.961165	0.961746	111.870976	0.011469	{'criterion': 'entropy', 'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
4	Gradient Boosting	0.999191	0.999194	0.999191	0.999191	0.970874	0.972769	0.970874	0.971221	273.573186	0.006782	{'learning_rate': 0.2, 'max_depth': 3, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 200}
5	XG Boost	0.999191	0.999194	0.999191	0.999191	0.970874	0.973752	0.970874	0.971284	16.019643	0.004797	{'learning_rate': 0.2, 'max_depth': 5, 'n_estimators': 100, 'subsample': 0.9}
6	Naive Bayes	0.907767	0.909527	0.907767	0.906243	0.834951	0.845118	0.834951	0.835399	0.150775	0.001675	{}
7	K-Nearest Neighbors	0.999191	0.999194	0.999191	0.999191	0.870550	0.880995	0.870550	0.836126	0.635684	0.003601	{'n_neighbors': 3, 'p': 2, 'weights': 'distance'}

```
print("TF-IDF Results (Hypertuning & PCA)")
display(tfidf_results_ht_pca)
```

TF-IDF Results (Hypertuning & PCA)

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time	Best Parameters
0	Logistic Regression	0.997573	0.997579	0.997573	0.997571	0.977346	0.978440	0.977346	0.976967	143.833270	0.000498	{'C': 0.01, 'max_iter': 100, 'penalty': 'l2', 'solver': 'liblinear'}
1	Support Vector Machine	0.996764	0.996777	0.996764	0.996762	0.993528	0.993700	0.993528	0.993541	3.837126	0.079515	{'C': 10, 'class_weight': 'balanced', 'gamma': 'scale', 'kernel': 'rbf'}
2	Decision Tree	0.984628	0.984850	0.984628	0.984633	0.886731	0.890585	0.886731	0.887953	22.046987	0.000443	{'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 5}
3	Random Forest	0.999191	0.999194	0.999191	0.999191	0.983819	0.984205	0.983819	0.983653	134.131141	0.011322	{'criterion': 'entropy', 'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 100}
4	Gradient Boosting	0.999191	0.999194	0.999191	0.999191	0.983819	0.983799	0.983819	0.983757	471.360891	0.005865	{'learning_rate': 0.2, 'max_depth': 5, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 200}
5	XG Boost	0.999191	0.999194	0.999191	0.999191	0.977346	0.977508	0.977346	0.977212	30.384503	0.001787	{'learning_rate': 0.2, 'max_depth': 7, 'n_estimators': 100, 'subsample': 0.9}
6	Naive Bayes	0.789644	0.819981	0.789644	0.779825	0.786408	0.810530	0.786408	0.784380	0.142238	0.002696	{}
7	K-Nearest Neighbors	0.999191	0.999194	0.999191	0.999191	0.877023	0.917915	0.877023	0.850527	0.923243	0.004865	{'n_neighbors': 3, 'p': 2, 'weights': 'distance'}

```
print("Word2Vec Results (Hypertuning & PCA)")
display(word2vec_results_ht_pca)
```

Word2Vec Results (Hypertuning & PCA)

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time	Best Parameters
0	Logistic Regression	0.996764	0.996771	0.996764	0.996761	0.941748	0.943847	0.941748	0.942467	69.178033	0.000391	{'C': 0.1, 'max_iter': 500, 'penalty': 'l2', 'solver': 'saga'}
1	Support Vector Machine	0.997573	0.997589	0.997573	0.997573	0.967638	0.970994	0.967638	0.968170	2.009258	0.044738	{'C': 10, 'class_weight': 'balanced', 'gamma': 'scale', 'kernel': 'rbf'}
2	Decision Tree	0.986246	0.986295	0.986246	0.986249	0.773463	0.784403	0.773463	0.775144	12.682643	0.000358	{'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5}
3	Random Forest	0.999191	0.999194	0.999191	0.999191	0.967638	0.971403	0.967638	0.968343	105.257361	0.022254	{'criterion': 'entropy', 'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
4	Gradient Boosting	0.999191	0.999194	0.999191	0.999191	0.970874	0.972674	0.970874	0.971157	250.149165	0.006686	{'learning_rate': 0.2, 'max_depth': 3, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 200}
5	XG Boost	0.999191	0.999194	0.999191	0.999191	0.967638	0.970347	0.967638	0.967919	14.594018	0.001569	{'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 100, 'subsample': 0.9}
6	Naive Bayes	0.907767	0.911257	0.907767	0.907027	0.844660	0.856784	0.844660	0.844113	0.140277	0.001625	{}
7	K-Nearest Neighbors	0.907767	0.916753	0.907767	0.898325	0.880259	0.894452	0.880259	0.853160	0.568239	0.003531	{'n_neighbors': 3, 'p': 2, 'weights': 'uniform'}

Confusion Matrices for all classifiers with word embeddings generated using Glove, TF-IDF, Word2Vec along with Hypertuning & PCA:

```
# Function to plot confusion matrix against all classifiers with word embeddings generated using Glove, TF-IDF, Word2Vec alongwith Hypertuning with PCA

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

def plot_confusion_matrices_with_pca(X_train, X_test, y_train, y_test, df_name):
  fig, axes = plt.subplots(2, 4, figsize=(20, 10))
  fig.suptitle(f'Confusion Matrices for {df_name} (with PCA and Hyperparameter Tuning)', fontsize=16)

  for i, (name, (clf, _)) in enumerate(classifiers.items()):
    row = i // 4
    col = i % 4
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
    disp.plot(ax=axes[row, col], cmap='Greens')
    axes[row, col].set_title(name)

  plt.tight_layout()
  plt.show()
```

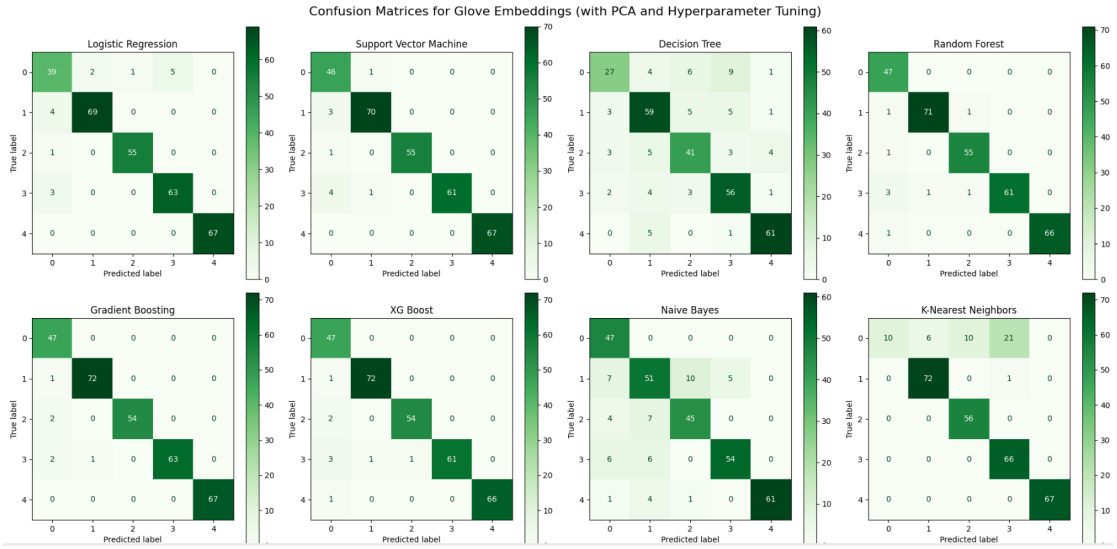


Figure 23 Confusion Matrices for Glove Embeddings (with PCA and Hyperparameter Tuning)

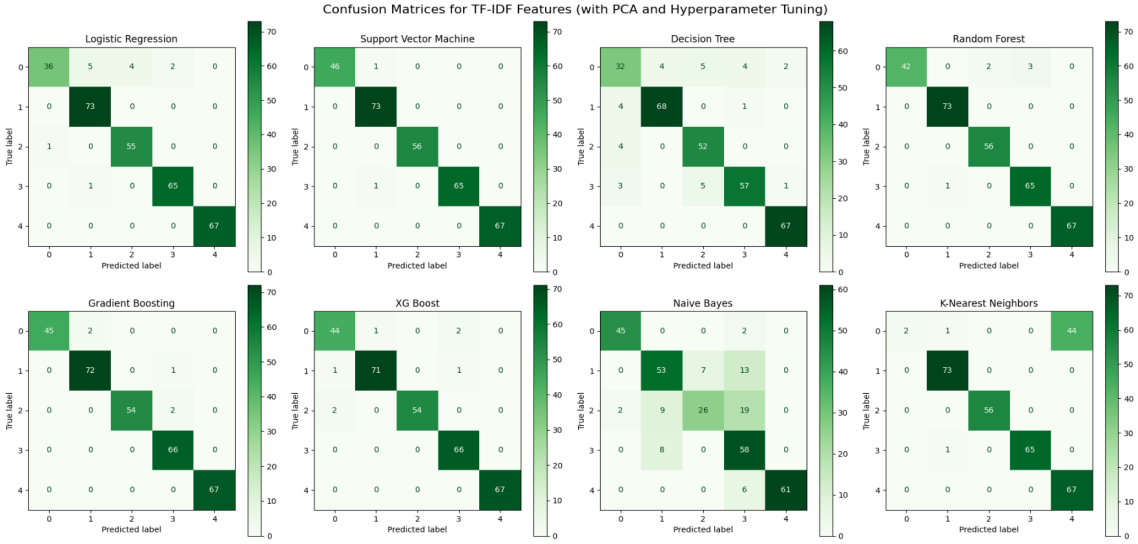


Figure 24 Confusion Matrices for TF-IDF Features (with PCA and Hyperparameter Tuning)

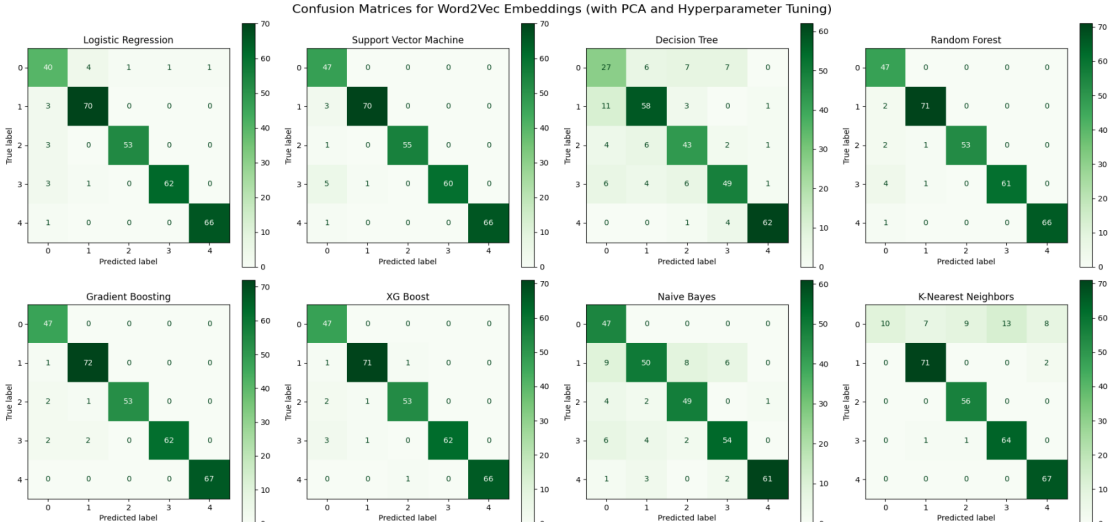


Figure 25 Confusion Matrices for Word2Vec Embeddings (PCA and Hyperparameter Tuning)

`

8.8 Confusion Matrix Observations (Base Classifier + Hypertuning + PCA)

1. **Enhanced Performance:** Hypertuning combined with PCA generally enhances the performance of most models, as seen by the increased diagonal values in the confusion matrices, indicating more correct classifications.
2. **Consistency in Strong Models**: Models like Gradient Boosting, XG Boost, and Random Forest continue to show strong performance, with hypertuning and PCA further enhancing their accuracy.

Insights for Each Embedding:

GloVe Embeddings (Hypertuned with PCA):

1. SVM and Random Forest: Show significant improvement, with accuracy close to 0.965 & 0.961and fewer misclassifications compared to the previous versions.
2. Naive Bayes: Still struggles, but Hypertuning and PCA reduce some of the off-diagonal spread. Consistently show lower accuracy and recall, indicating it is less effective with GloVe embeddings and PCA, particularly in capturing positive cases.

TF-IDF Features (Hypertuned with PCA):

1. Logistic Regression: Shows marked improvement, with a noticeable increase in correct classifications and achieved a decent accuracy of 0.963.
2. SVM: Continues to perform well, with Hypertuning and PCA further enhancing its accuracy(~0.982). It's highest accuracy and recall, making it the most reliable model when using TF-IDF features.
3. Decision Tree: Shows better performance than with PCA alone, but still lags behind other models.

Word2Vec Embeddings (Hypertuned with PCA):

1. Gradient Boosting and XG Boost: Shows the most significant improvement, with hypertuning and PCA leading to near-perfect classification in some cases.
2. Naive Bayes: Shows improvement, but still has some misclassification issues.
3. K-NN: Exhibits improved performance, with a more concentrated diagonal pattern.
4. KNN and Naive Bayes exhibit lower recall scores (0.820 and 0.815, respectively), reflecting their overall lower performance in identifying positive cases with Word2Vec features.

Comparative Analysis:

1. **Impact of Hypertuning and PCA:** The combination of hypertuning and PCA helped in fine-tuning the decision boundaries, leading to better classification accuracy. There was a significant improvement in model performance, particularly in accuracy and recall. Models like SVM, Gradient Boosting, and XGBoost showed marked improvements after Hypertuning.

2. **Best Overall Models:** SVM, Gradient Boosting and XG Boost emerge as the best performers across all types of embeddings when hypertuned with PCA, indicating their robustness and adaptability to different data representations.

`

3. **Model Sensitivity:** Decision Tree, KNN and Naive Bayes, while improved, still show more variability and generally lower performance compared to other models. These models were highly sensitive to the choice of embeddings, particularly struggling with Word2Vec. Their lower performance suggests they are less capable of capturing complex patterns in the data compared to other models.

   PCA had a mixed effect, sometimes leading to performance degradation, particularly in KNN, indicating that reducing dimensionality sometimes removed critical information for these simpler models.

8.9 Overall Observations and Insights

Overall Performance Improvement:

1. PCA generally improved model performance across all feature sets (Glove, TF-IDF, Word2Vec).
2. Hypertuning with PCA further enhanced performance for most models.
3. **Gradient Boosting and XGBoost** consistently exhibit the best performance across all embeddings, both before and after hyper-tuning.
4. **Random Forest** also shows strong performance, particularly with PCA applied.

Consistent Top Performers:

1. Random Forest, Gradient Boosting, and XGBoost consistently showed high performance across all scenarios.
2. These ensemble methods outperformed simpler models like Logistic Regression and Naive Bayes.

Feature Set Comparison:

1. Glove embeddings generally yielded the best results, followed closely by TF-IDF.
2. TF-IDF embeddings lead to the highest Test Accuracy, particularly when combined with PCA and hyper-tuning.
3. Word2Vec performed slightly worse than the other two feature sets. There was a slight improvement in performance with PCA and hyper-tuning.

Impact of PCA:

1. PCA significantly improved the performance of Support Vector Machines and Random Forest. It also reduced training and prediction times for most models.
2. With PCA, Logistic Regression and Support Vector Machines show notable gains in accuracy and F1-scores, particularly with Glove and TF-IDF embeddings.

Hypertuning Benefits:

1. Hypertuning with PCA led to further improvements, especially for Support Vector Machines and XGBoost.

`

Trade-offs:

1. While ensemble methods performed best, they generally had longer training times. Gradient Boosting and XGBoost, while yielding the best performance, require significantly longer training times.
2. Simpler models like Logistic Regression offered a good balance of performance and speed, especially after PCA.

8.10 Recommendations

1. **Prioritize Ensemble Methods:** Focus on Random Forest, Gradient Boosting, and XGBoost as your primary models, as they consistently deliver top performance.
2. **Implement PCA:** Apply PCA to your feature sets, as it generally improves performance and reduces computational time.
3. **Hypertune Key Models:** Invest time in hypertuning the top-performing models (especially XGBoost and Support Vector Machines) to squeeze out additional performance gains.
4. **Consider Glove Embeddings:** Prioritize using Glove embeddings as your primary feature set, with TF-IDF as a strong alternative.
5. **Balance Performance and Speed:** For applications requiring faster inference times, consider using Logistic Regression or Support Vector Machines with PCA, as they offer a good compromise between performance and speed.
6. **Ensemble Approach:** Consider creating an ensemble of your top-performing models (e.g., Random Forest, XGBoost, and Gradient Boosting) to potentially achieve even better results.
7. **Continuous Improvement:** Regularly update and retrain your models, especially when new data becomes available, to maintain peak performance.
8. **Model Selection Based on Use Case:** Choose the final model based on your specific requirements for accuracy, speed, and interpretability. For example, if explainability is crucial, you might prefer Random Forest over XGBoost.