PGP AIML Feb 24 A - Group 7

CAPSTONE PROJECT - Interim Report

NATURAL LANGUAGE PROCESSING

INDUSTRIAL SAFETY - NLP BASED CHATBOT

TEAM DETAILS

Mentor: ROHIT GUPTA	
NAVEEN JOTHI	naveenjothi040@gmail.com
MOHIT JAIN	mohit_jain@yahoo.com
RAHUL PATERIA	rahulpateriya121@gmail.com
AKASH KUMAR	akki.vik3@gmail.com
SOHIT SINGH	Singh.sohit46@gmail.com
PRATIK PATIL	pppratikpatil723@gmail.com

PROBLEM STATEMENT

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.

DATA DESCRIPTION

This database is basically records of accidents from 12 different plants in 03 different countries where every line in the data is an occurrence of an accident.

Columns description:

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

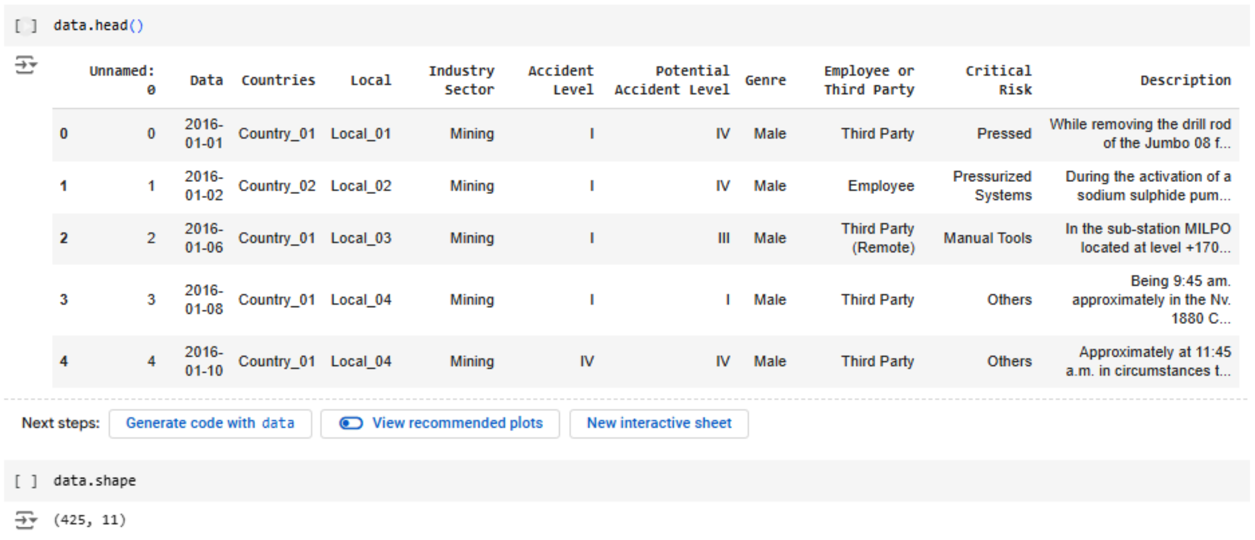
PROJECT OBJECTIVE

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

EXPLORATORY DATA ANALYSIS

INITIAL REVIEW

After importing the necessary libraries and loading the data, we analyzed the data's structure and significance, taking appropriate actions based on our observations.



```
[ ] data.head()
```

	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 f...
1	1	2016-01-02	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...
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...
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 C...
4	4	2016-01-10	Country_01	Local_04	Mining	IV	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances t...

```
[ ] data.shape
```

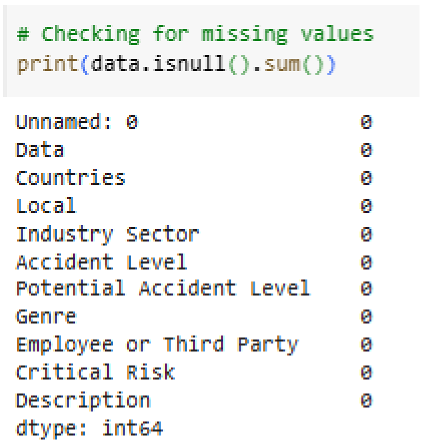
```
(425, 11)
```

Insights gained on dataset:

The dataset is relatively small (425 rows x 11 columns) but contains relevant information.

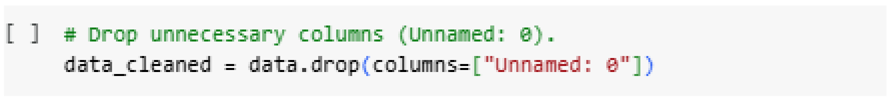
It is important to note that some components of the dataset were anonymized to conceal the names and locations of the facilities.

ANALYSING AND HANDLING MISSING/NULL VALUES



```
# Checking for missing values
print(data.isnull().sum())
```

```
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
```



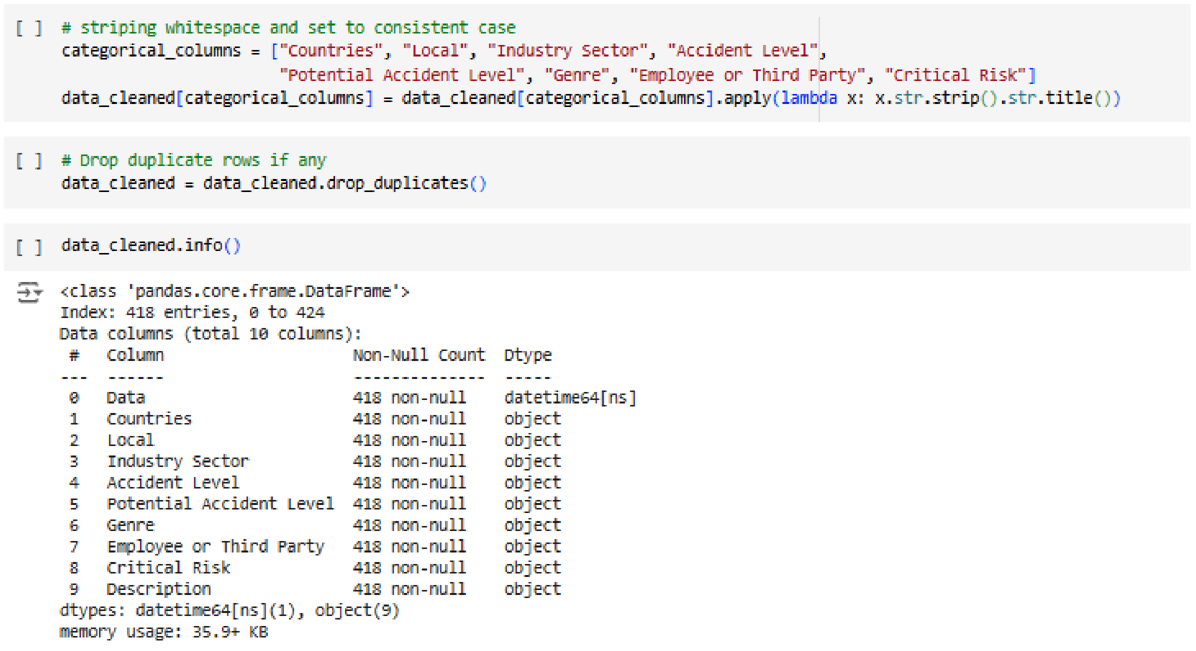
```
[ ]  # Drop unnecessary columns (Unnamed: 0).
     data_cleaned = data.drop(columns=["Unnamed: 0"])
```

There are no missing values in the dataset. An unnecessary column named "Unnamed" was removed due to the lack of related metadata, as it added no value to the analysis.

HANDLING DUPLICATES AND OTHER DATA CLEANING STEPS

We observe duplicate records in the given dataset, hence we drop the duplicates.

Initial dataset had 425 rows as observed in the previous step, after dropping 7 duplicate rows, we are left with 418 rows.



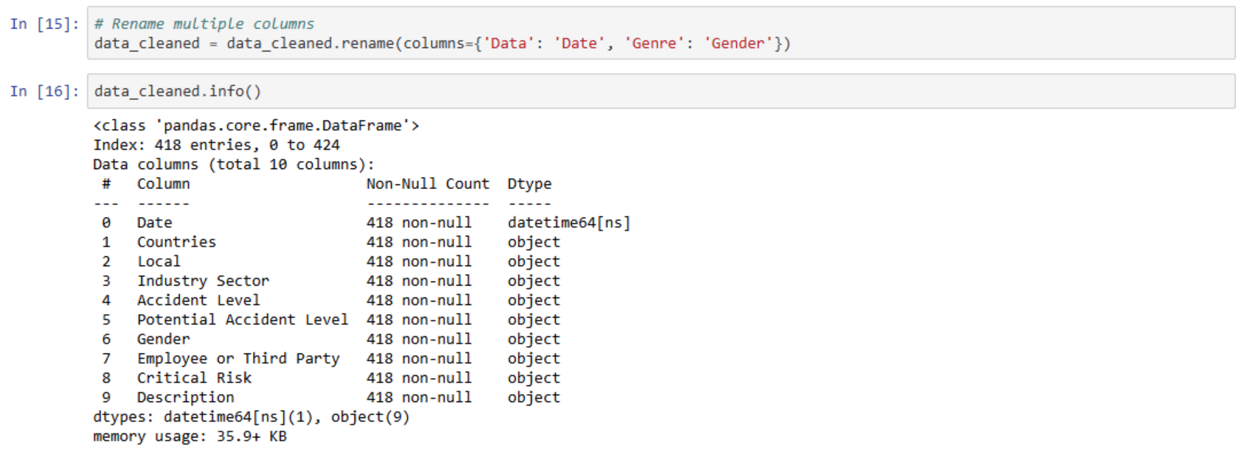
```
[ ] # striping whitespace and set to consistent case
    categorical_columns = ["Countries", "Local", "Industry Sector", "Accident Level",
                           "Potential Accident Level", "Genre", "Employee or Third Party", "Critical Risk"]
    data_cleaned[categorical_columns] = data_cleaned[categorical_columns].apply(lambda x: x.str.strip().str.title())
```

```
[ ] # Drop duplicate rows if any
    data_cleaned = data_cleaned.drop_duplicates()
```

```
[ ] data_cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 418 entries, 0 to 424
Data columns (total 10 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Data                      418 non-null    datetime64[ns]
 1   Countries                 418 non-null    object
 2   Local                     418 non-null    object
 3   Industry Sector           418 non-null    object
 4   Accident Level            418 non-null    object
 5   Potential Accident Level  418 non-null    object
 6   Genre                     418 non-null    object
 7   Employee or Third Party   418 non-null    object
 8   Critical Risk             418 non-null    object
 9   Description               418 non-null    object
dtypes: datetime64[ns](1), object(9)
memory usage: 35.9+ KB
```

We noticed that the dataset contains two column names, "Data" and "Genre," in Portuguese. To ensure consistency in the language across the dataset, we are converting these columns to English.



```
In [15]: # Rename multiple columns
         data_cleaned = data_cleaned.rename(columns={'Data': 'Date', 'Genre': 'Gender'})

In [16]: data_cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 418 entries, 0 to 424
Data columns (total 10 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Date                      418 non-null    datetime64[ns]
 1   Countries                 418 non-null    object
 2   Local                     418 non-null    object
 3   Industry Sector           418 non-null    object
 4   Accident Level            418 non-null    object
 5   Potential Accident Level  418 non-null    object
 6   Gender                    418 non-null    object
 7   Employee or Third Party   418 non-null    object
 8   Critical Risk             418 non-null    object
 9   Description               418 non-null    object
dtypes: datetime64[ns](1), object(9)
memory usage: 35.9+ KB
```

We are extracting the day, month, and year from the date column in order to conduct exploratory data analysis (EDA) in the subsequent steps. This will help us understand the distribution and identify patterns related to the days and months when incidents occurred.

```
# Ensure the 'Date' column is in datetime format
# data_cleaned['Date'] = pd.to_datetime(data_cleaned['Date'])

# Extract year, month, and day into separate columns
data_cleaned['Year'] = data_cleaned['Date'].dt.year
data_cleaned['Month'] = data_cleaned['Date'].dt.month
data_cleaned['Day'] = data_cleaned['Date'].dt.day

# Display the updated DataFrame
print(data_cleaned.head())
```

Since we have already extracted the date features into individual columns, the original "Date" feature is no longer necessary and will be dropped.

```
In [19]: data_cleaned.drop(columns=['Date'], inplace=True)

In [20]: data_cleaned.head()

Out[20]:
```

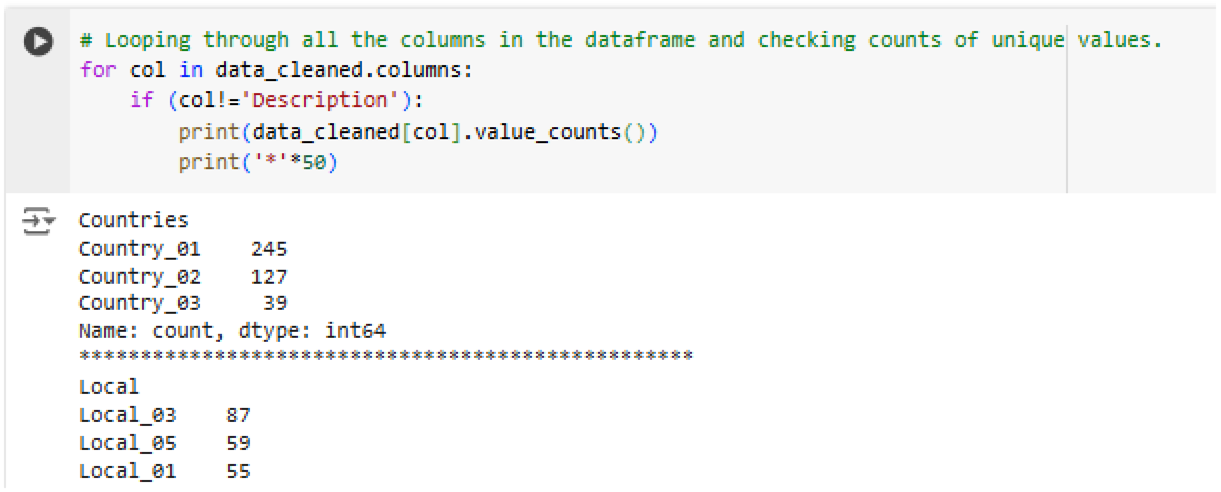
	Countries	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee or Third Party	Critical Risk	Description	Year	Month	Day
0	Country_01	Local_01	Mining	I	Iv	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...	2016	1	1
1	Country_02	Local_02	Mining	I	Iv	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...	2016	1	2
2	Country_01	Local_03	Mining	I	Iii	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...	2016	1	6
3	Country_01	Local_04	Mining	I	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 C...	2016	1	8
4	Country_01	Local_04	Mining	Iv	Iv	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances t...	2016	1	10

Additionally, we observe that the data frame contains 7 rows where only one or two column values differ, while the "Description" column remains the same across these rows. This inconsistency is logically incorrect, so we have decided to drop these rows as well.

```
In [21]: # Dropping the duplicates we detected above.
         data_cleaned.drop_duplicates(subset=['Description'], keep='first', inplace=True)
         print('After removing duplicates the shape of the dataset is:', data_cleaned.shape)

         After removing duplicates the shape of the dataset is: (411, 12)
```

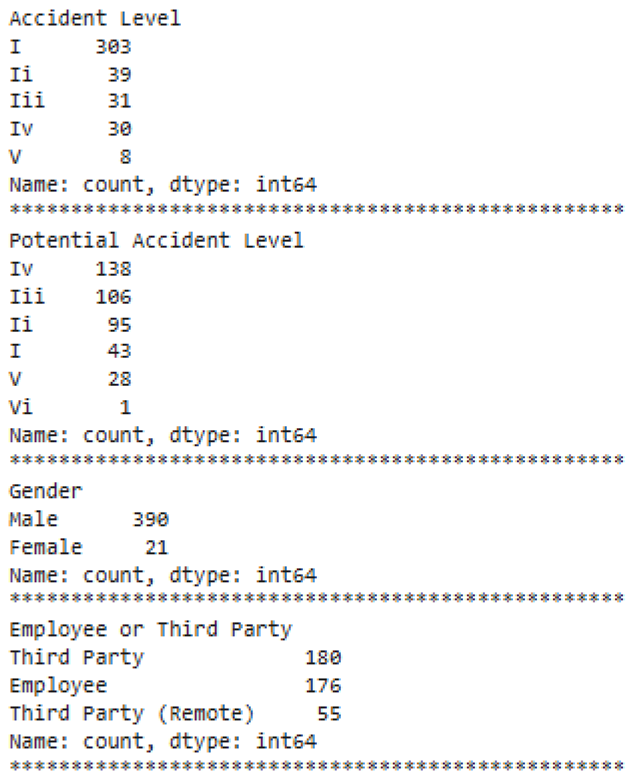
ANALYSING PATTERNS

Analyzing the occurrence patterns across categorical data within the dataset to identify key trends and insights.



```
# Looping through all the columns in the dataframe and checking counts of unique values.
for col in data_cleaned.columns:
    if (col!='Description'):
        print(data_cleaned[col].value_counts())
        print('*'*50)
```

```
Countries
Country_01    245
Country_02    127
Country_03     39
Name: count, dtype: int64
**************************************************
Local
Local_03    87
Local_05    59
Local_01    55
```



```
Accident Level
I      303
Ii      39
Iii     31
Iv      30
V        8
Name: count, dtype: int64
**************************************************
Potential Accident Level
Iv     138
Iii    106
Ii      95
I       43
V       28
Vi       1
Name: count, dtype: int64
**************************************************
Gender
Male      390
Female     21
Name: count, dtype: int64
**************************************************
Employee or Third Party
Third Party             180
Employee                176
Third Party (Remote)     55
Name: count, dtype: int64
**************************************************
```

```
Critical Risk
Others                                      223
Pressed                                      24
Manual Tools                                 20
Chemical Substances                          17
Cut                                          14
Venomous Animals                             13
Projection                                   12
Bees                                         10
```

```
 Month
 2     60
 4     51
 3     50
 6     49
 1     39
 5     38
 9     24
 7     23
 12    22
 8     21
 10    21
 11    13
```

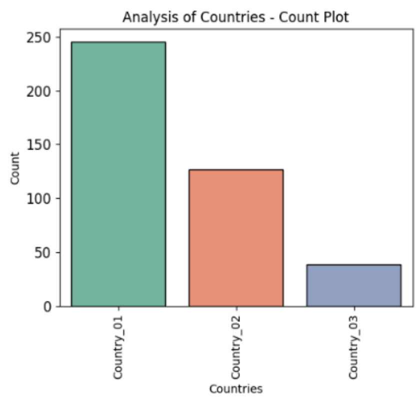
OBSERVATIONS

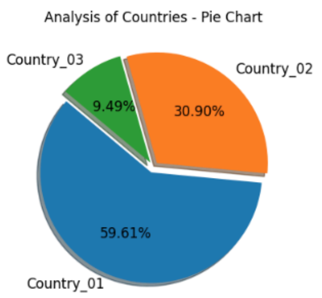
- Most of the incidents (over 50%) occurred in Country_01
- Most of the incidents from Country_01 has been reported from Local_03 city
- Most incidents reported are least severe (Level I)
- Most incidents occurred are to male which signifies it's a male dominated industry
- "Others" is the largest category (223 incidents), suggesting the need for clearer risk definitions.

VISUALIZATION

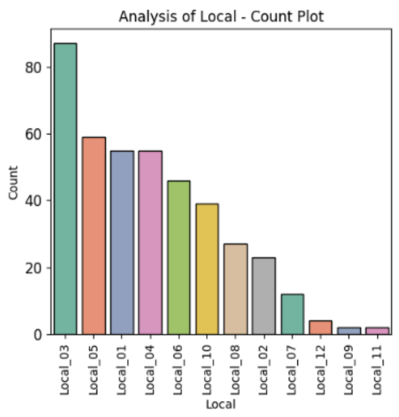
UNIVARIATE ANALYSIS

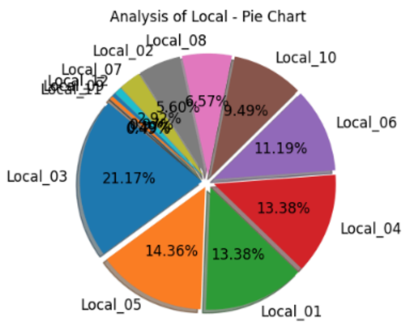
Incident occurrences by countries



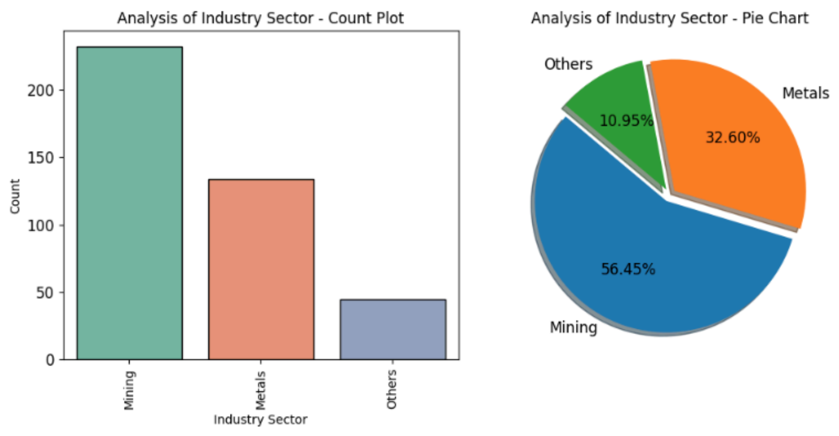


Incident occurrences by local (City)

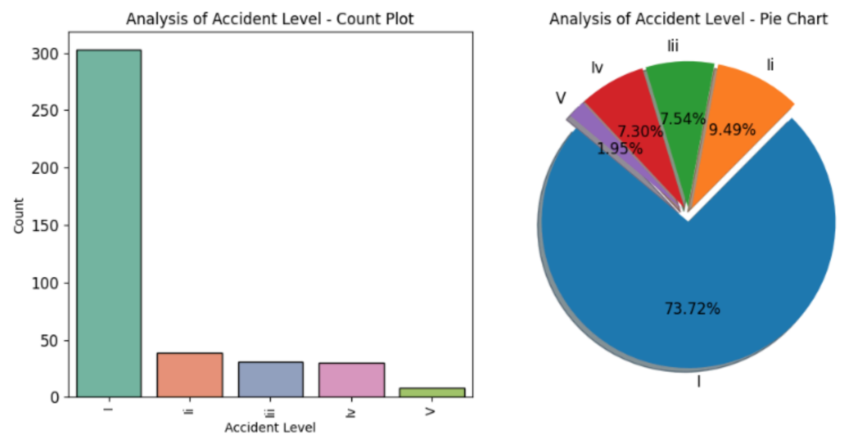




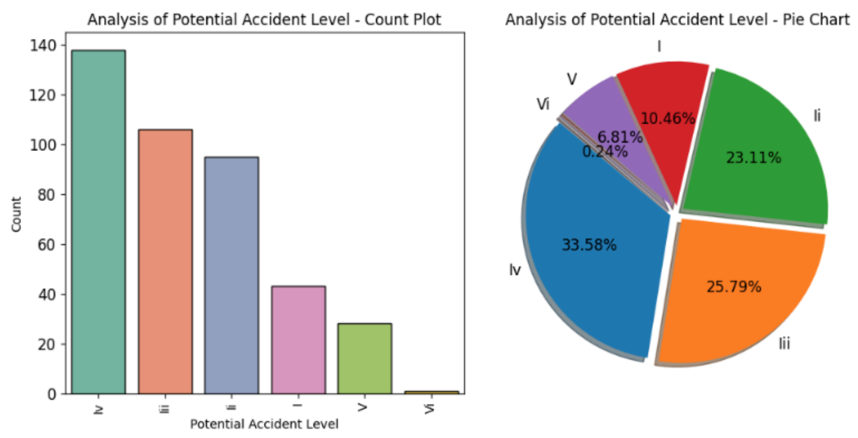
Incident occurrences by industry sector



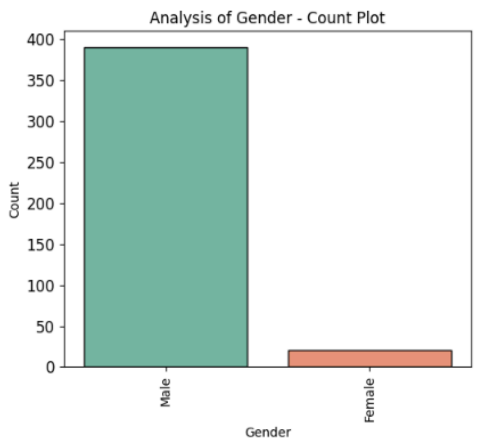
Incident occurrences by accident level

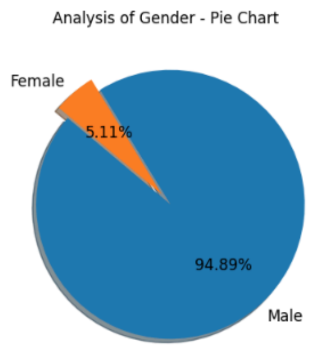


Incident occurrences by potential accident level

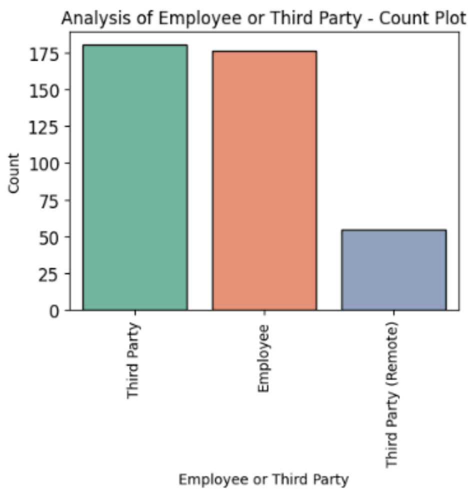


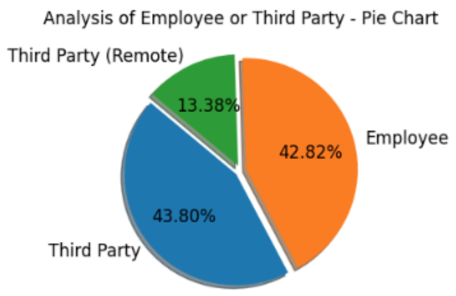
Incident occurrences by gender



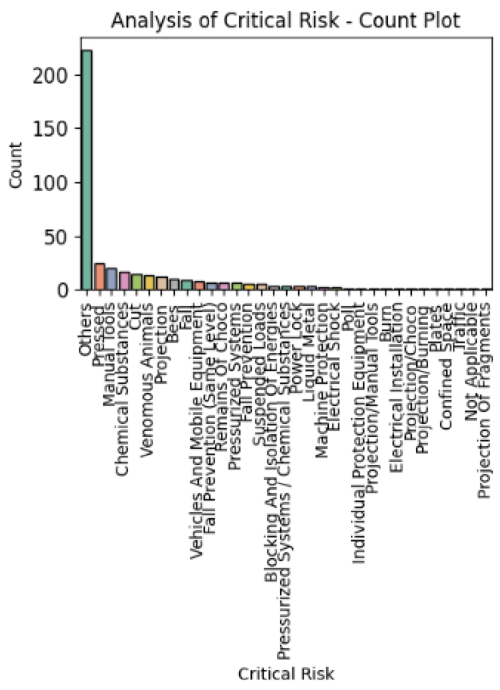


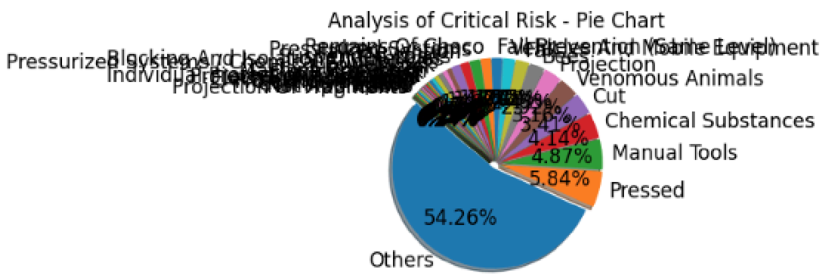
Incident occurrences by employment type





Incident occurrences by critical risk





KEY INSIGHTS - Univariate Analysis

1. Geographical Distribution

- **Countries**:
  - Country_01 accounts for a significant majority of incidents (245), comprising over half the dataset.
  - Country_02 and Country_03 have substantially fewer incidents (127 and 39, respectively).
- **Local Areas**:
  - Local_03 stands out with 87 incidents, making it the most accident-prone area.
  - Local_05, Local_01, and Local_04 follow closely with similar counts, suggesting these locations require focused attention.

2. Industry Sectors

- **Mining** is the leading sector with 232 incidents, emphasizing its high-risk nature.
- **Metals** contribute to 134 incidents, indicating it is also a critical area for safety interventions.
- Other sectors account for a relatively small proportion (45).

3. Accident Severity

- **Accident Level**:
  - Level I accidents dominate the dataset (303 incidents), signifying that minor accidents are the most common.
  - Higher-severity accidents (Levels IV and V) are less frequent but still significant for targeted safety measures.
- **Potential Accident Level**:
  - Potential Level IV accidents (138) and Level III (106) highlight areas of latent high-risk scenarios.

4. Demographics

- **Gender**:
  - Males represent a staggering 95% of incidents, suggesting male-dominated roles in these industries may face greater exposure to risks.
- **Employee vs. Third Party**:
  - Third-party individuals (including remote third parties) are involved in more incidents (235) compared to employees (176), indicating external personnel face considerable safety challenges.

5. Critical Risks

- **Top Risk Factors**:
  - "Others" (223 incidents) may require further investigation to identify underlying risk contributors.

  - Pressed (24), Manual Tools (20), and Chemical Substances (17) are notable specific risks.

6. Temporal Trends

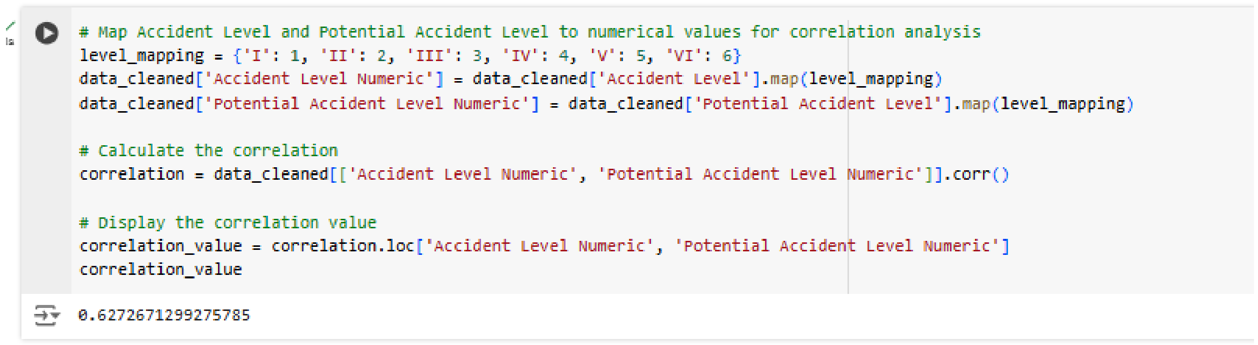
- **Yearly Data**:
  - A majority of accidents occurred in 2016 (278 incidents), compared to 2017 (133), indicating a declining trend.
- **Monthly Data**:
  - February (60) and April (51) saw the highest number of incidents, suggesting seasonality effects or operational peaks.

RECOMMENDATIONS

- Focus safety interventions on Country_01 and Local_03.
- Address critical risks like Pressed, Manual Tools, and Chemical Substances.
- Enhance safety measures in Mining and Metals industries.
- Develop targeted safety programs for males and third-party workers.
- Investigate incident spikes in February and April for potential seasonal or operational causes.most incidents (278).

BIVARIATE ANALYSIS

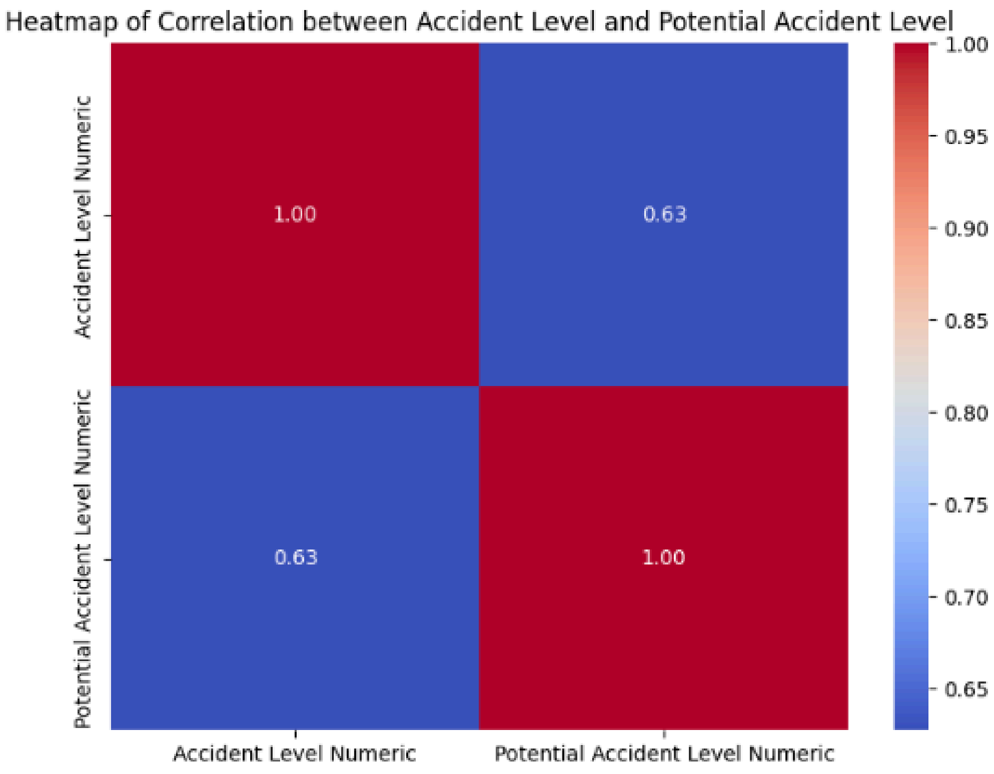
Correlation between 'Accident Level' and 'Potential Accident Level'



```
# Map Accident Level and Potential Accident Level to numerical values for correlation analysis
level_mapping = {'I': 1, 'II': 2, 'III': 3, 'IV': 4, 'V': 5, 'VI': 6}
data_cleaned['Accident Level Numeric'] = data_cleaned['Accident Level'].map(level_mapping)
data_cleaned['Potential Accident Level Numeric'] = data_cleaned['Potential Accident Level'].map(level_mapping)

# Calculate the correlation
correlation = data_cleaned[['Accident Level Numeric', 'Potential Accident Level Numeric']].corr()

# Display the correlation value
correlation_value = correlation.loc['Accident Level Numeric', 'Potential Accident Level Numeric']
correlation_value
```

```
0.6272671299275785
```

```
[91] # Generate a heatmap for the correlation matrix
     plt.figure(figsize=(8, 6))
     sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt='.2f', cbar=True)
     plt.title('Heatmap of Correlation between Accident Level and Potential Accident Level')
     plt.show()
```



A correlation of **0.627** between Accident Level and Potential Accident Level indicates a **moderately strong positive relationship** between these two variables. Here's what this means in context:

Correlation interpretation and implications

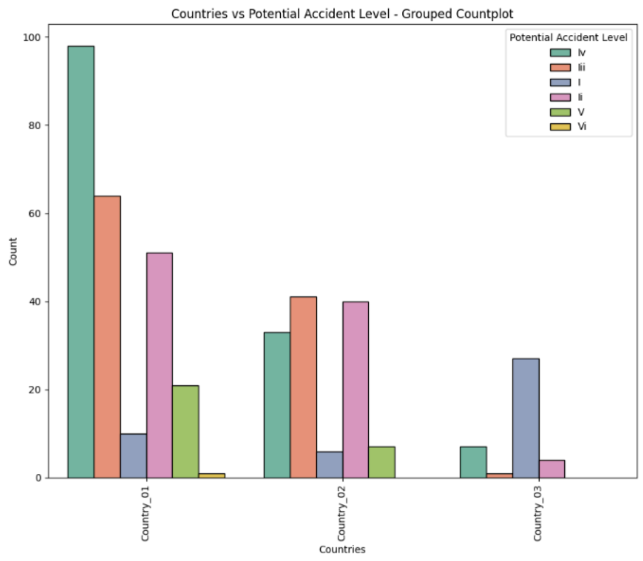
1. **Positive Relationship**:
    - As the **Accident Level** increases (indicating more severe actual accidents), the **Potential Accident Level** also tends to increase (indicating higher potential severity of the accident).
2. **Moderately Strong Correlation**:

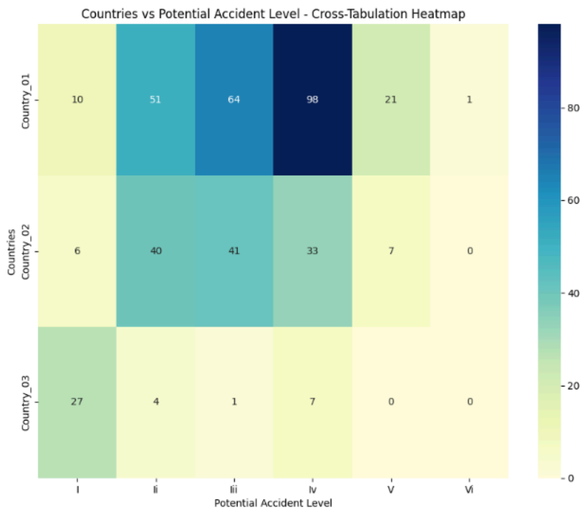
- The value of **0.627** suggests that while there is a significant relationship, it is not perfect. Other factors may also influence the Potential Accident Level, apart from the Accident Level.

3. **Practical Implication**:
    - This correlation implies that high-severity accidents are often associated with high potential risks, indicating a need to prioritize preventive measures for incidents with high potential severity to avoid severe actual outcomes.

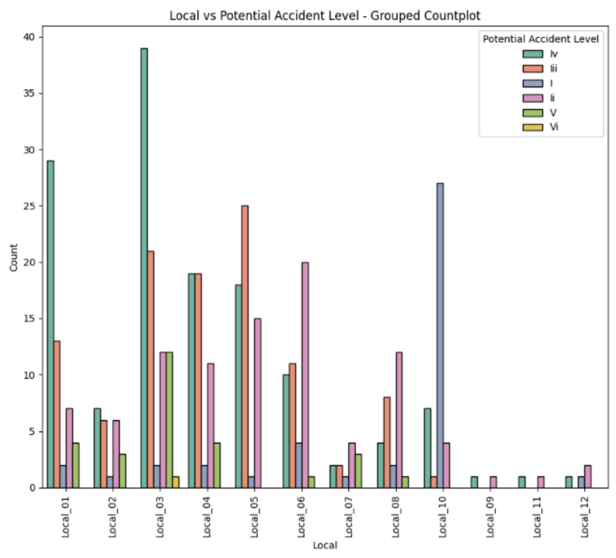
Selecting the '**Potential Accident Level**' as the **target variable** performing the bivariate analysis against all the other categorical columns.

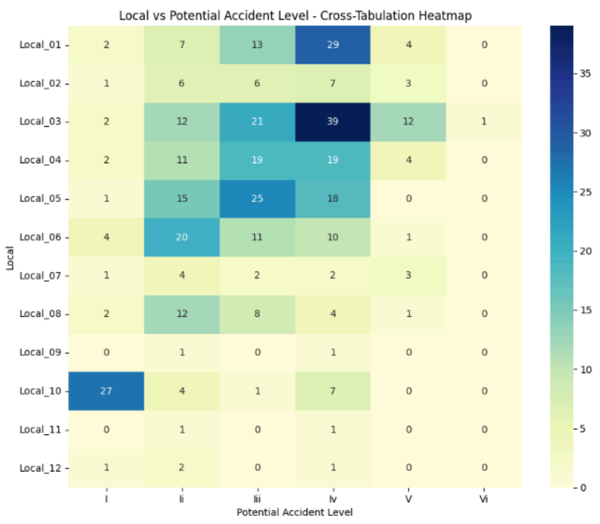
'Potential Accident Level' by 'Countries':



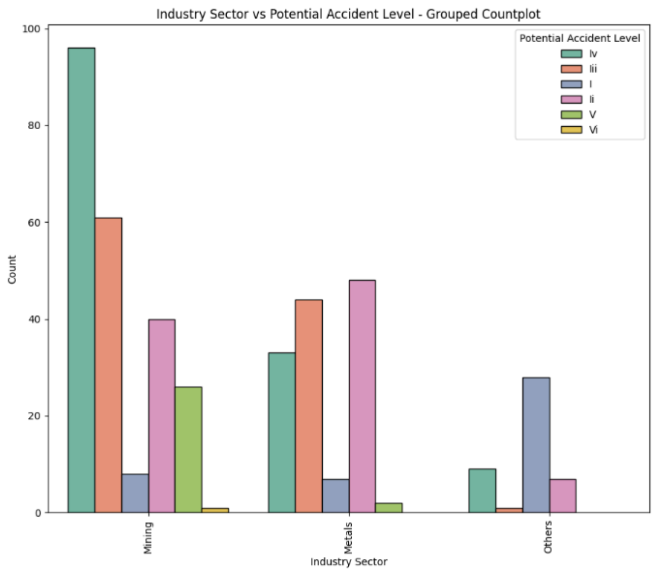


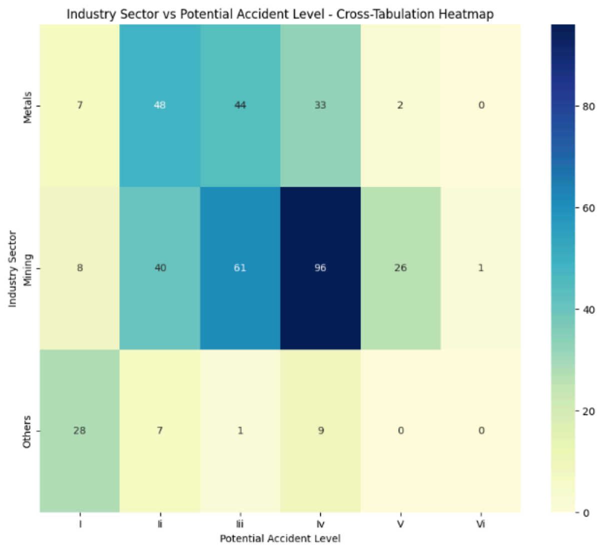
‘Potential Accident Level’ by ‘Local’ (City):



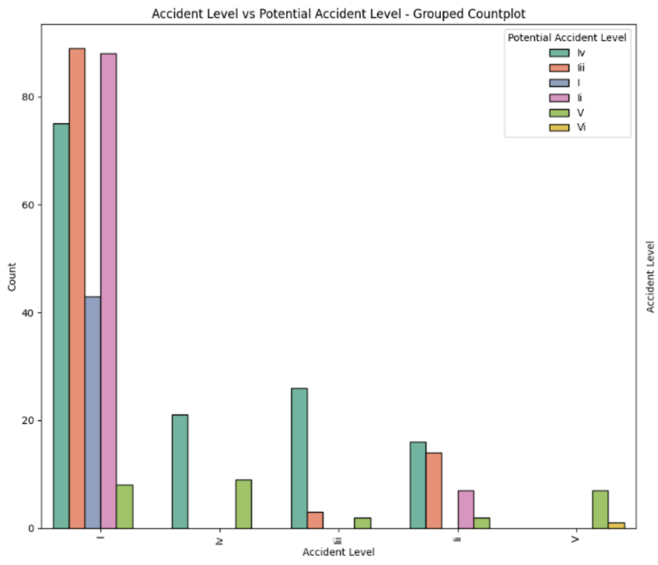


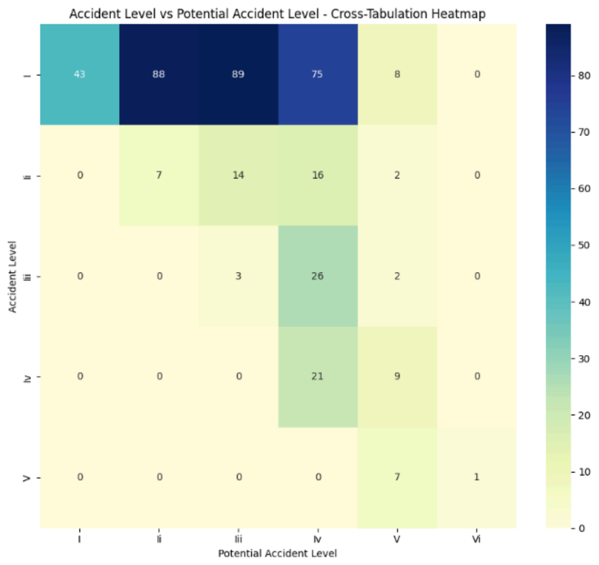
‘Potential Accident Level’ by ‘Industry Sector’:





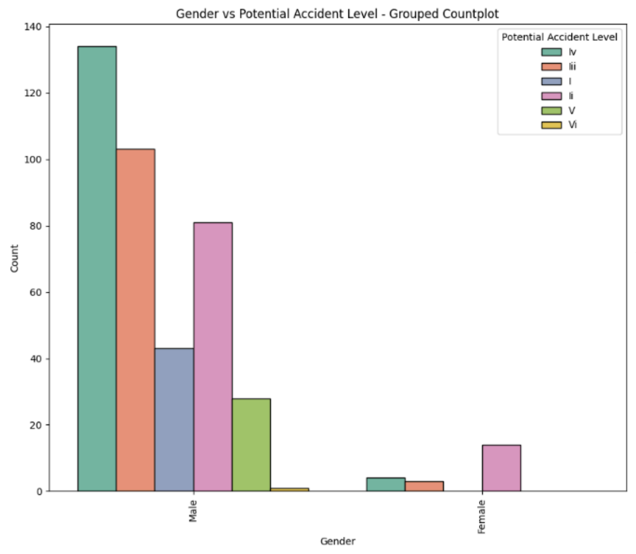
‘Potential Accident Level’ by ‘Accident Level’:

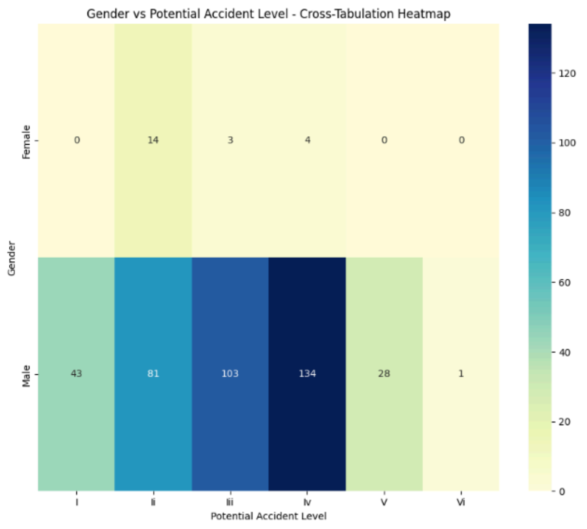




	I	Ii	Iii	Iv	V	Vi
I	43	88	89	75	8	0
Ii	0	7	14	16	2	0
Iii	0	0	3	26	2	0
Iv	0	0	0	21	9	0
V	0	0	0	0	7	1

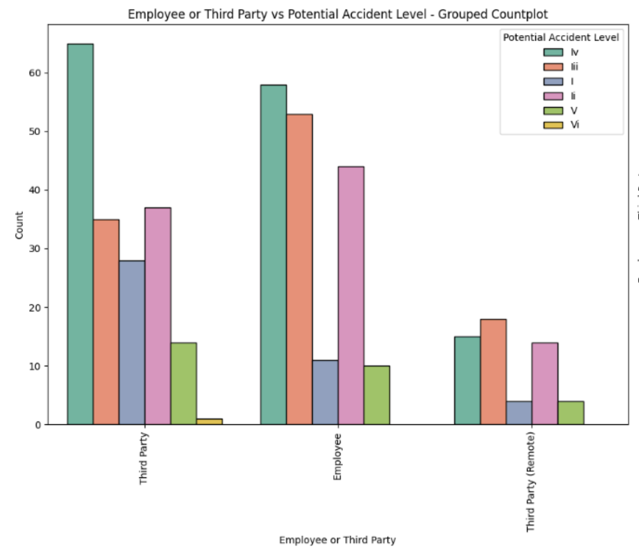
‘Potential Accident Level’ by ‘Gender’:

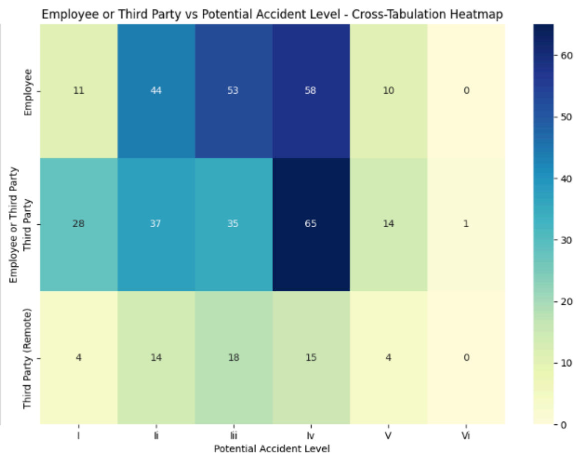




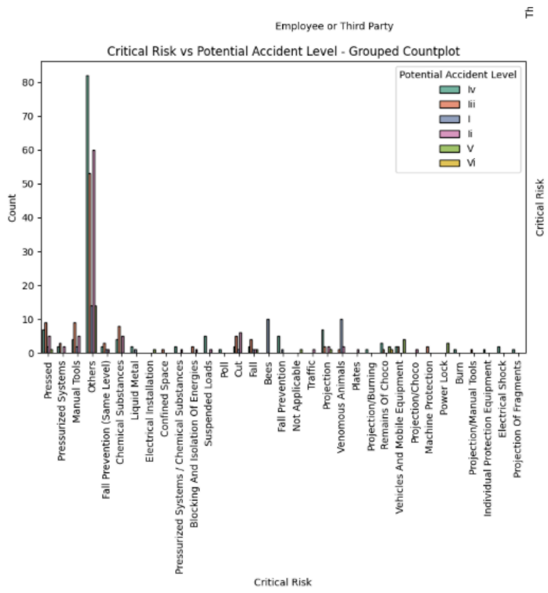
	I	Ii	Iii	Iv	V	Vi
Female	0	14	3	4	0	0
Male	43	81	103	134	28	1

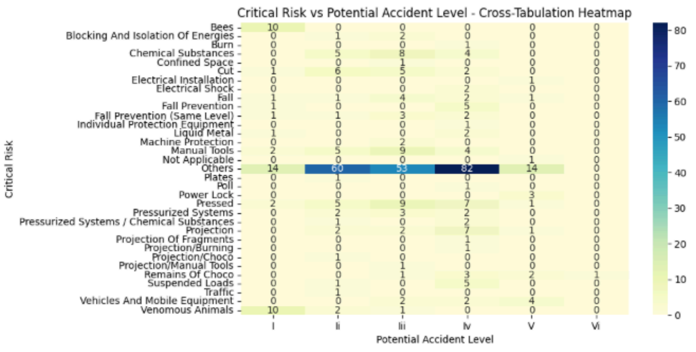
‘Potential Accident Level’ by Employment Type:



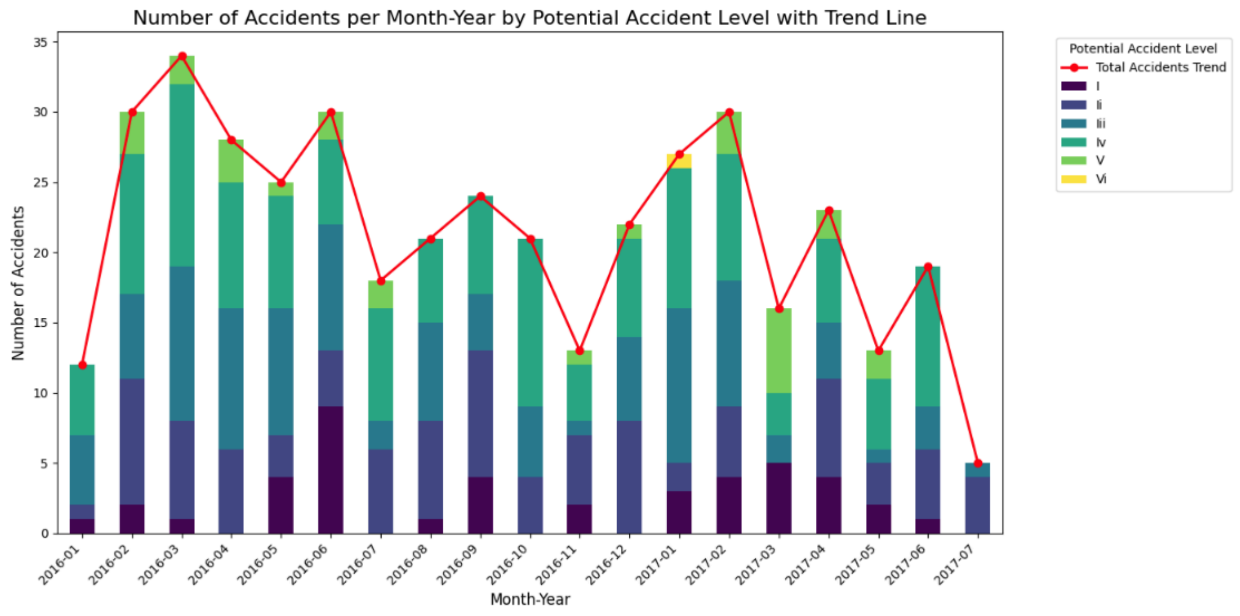


‘Potential Accident Level’ by ‘Critical Risk’:





Trend of accidents by year-month by ‘Potential Accident Level’



Level IV Dominance: Level IV potential accidents consistently dominate across all months, highlighting the need to focus on high-severity risk mitigation.

Fluctuating Monthly Accidents: Peaks in total accidents occur in February-March 2016 and January-February 2017, suggesting periodic or seasonal factors affecting incident rates.

Decline in Mid-2017: A clear decline in total accidents is observed in mid-2017, potentially reflecting effective safety measures or changes in activity levels.

Low but Present Extreme Risks: Levels V and VI (extreme risks) are rare but appear sporadically, underscoring the importance of preparedness for severe incidents.

Consistent Moderate Risks: Lower-potential severity levels (I, II, III) remain consistently present, indicating the need for continuous monitoring and interventions for moderate risks.

KEY INSIGHTS - Bivariate Analysis

Potential Accident Level by Industry Sector

- **Level IV** potential accidents dominate across all industries (141 incidents), followed by **Level III** (106) and **Level II** (95).
- **Mining** sees the highest count of potential accidents at all levels, reflecting its inherent high-risk nature.
- **Metals** and **Others** have relatively fewer incidents but still contribute to higher-level potential accidents.

Potential Accident Level by Year

- **2016** recorded the majority of higher potential severity levels, with Level IV being the most frequent.
- A slight decline in high-severity potential incidents is observed in 2017.

Potential Accident Level by Critical Risk

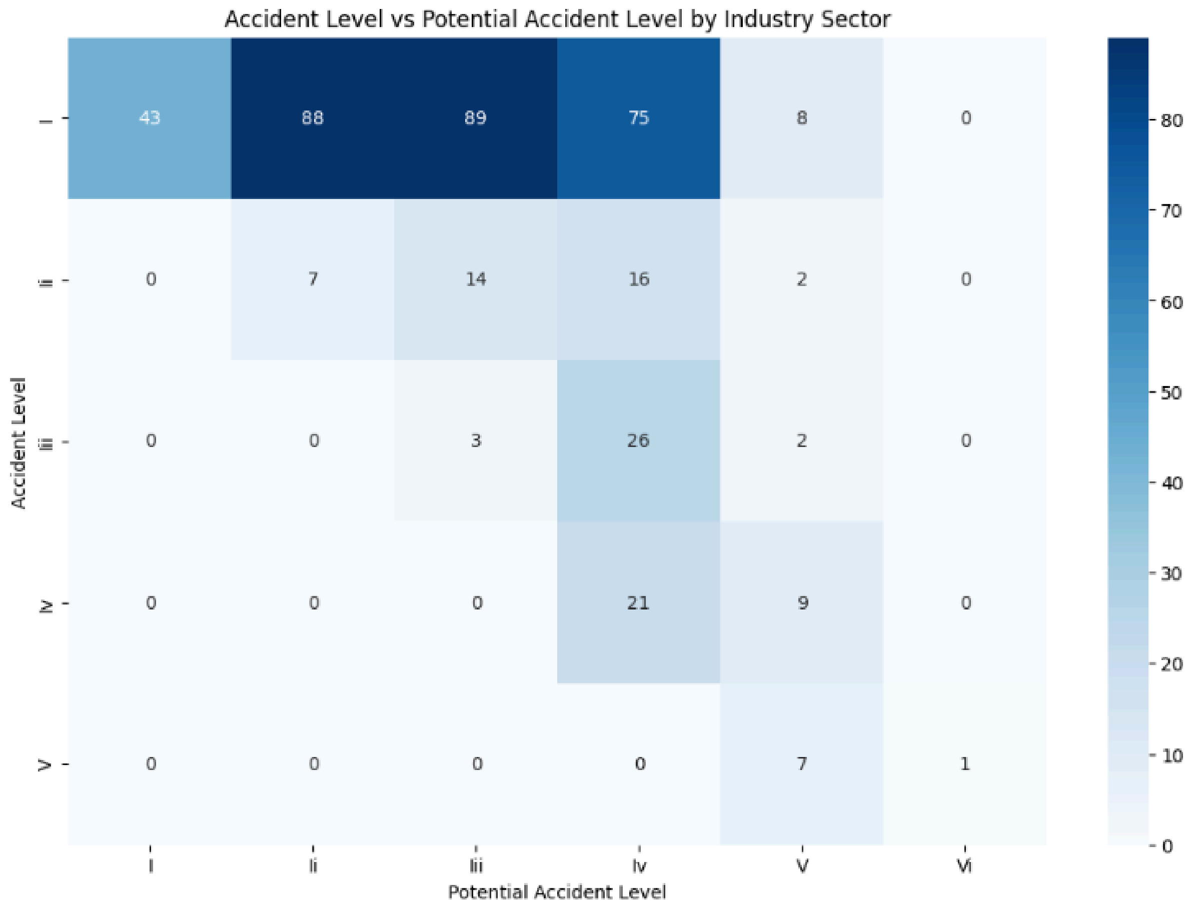
- **Level IV** potential accidents are spread across various risks, with "Others" being the most common, signalling ambiguous or uncategorised hazards.
- Risks like **Pressed** and **Manual Tools** are associated with higher potential severity levels (III and IV).
- **Chemical Substances** frequently appear at moderate potential levels (II and III).

General Observations

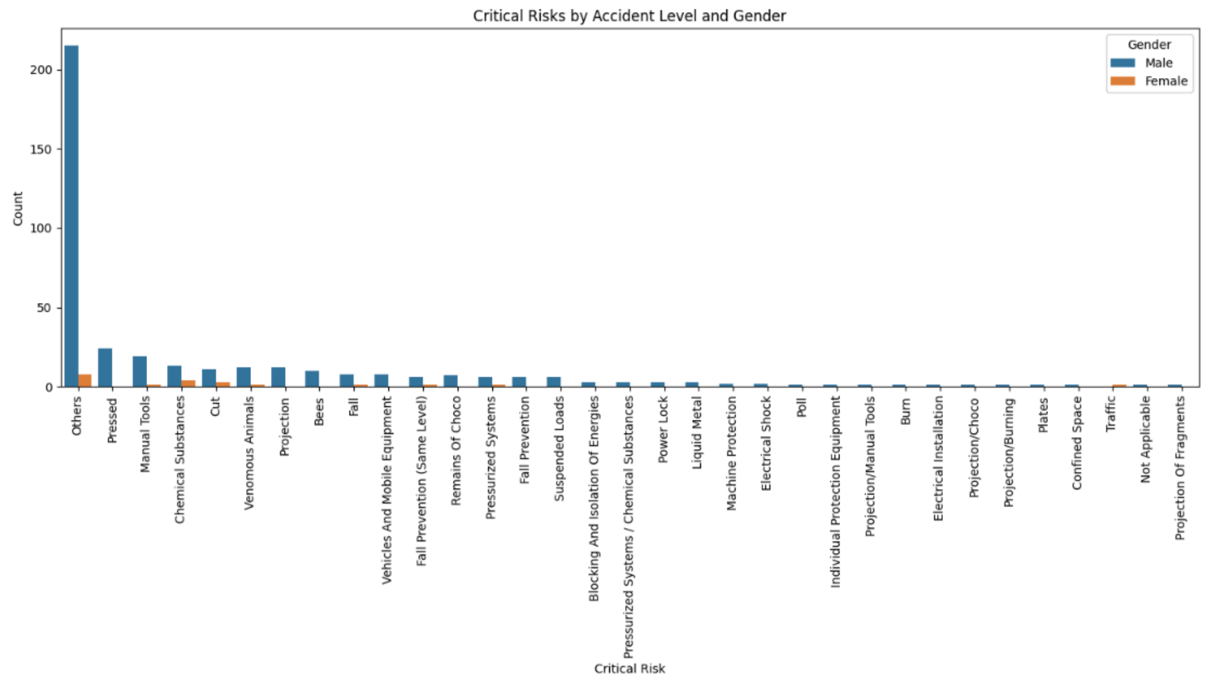
- **Mining** industry and **2016** have the most significant contributions to higher potential accident levels.
- The need for clearer categorisation of critical risks, especially those labelled as "Others," is evident.
- High-severity risks demand targeted safety interventions to reduce potential impact.

MULTIVARIATE ANALYSIS

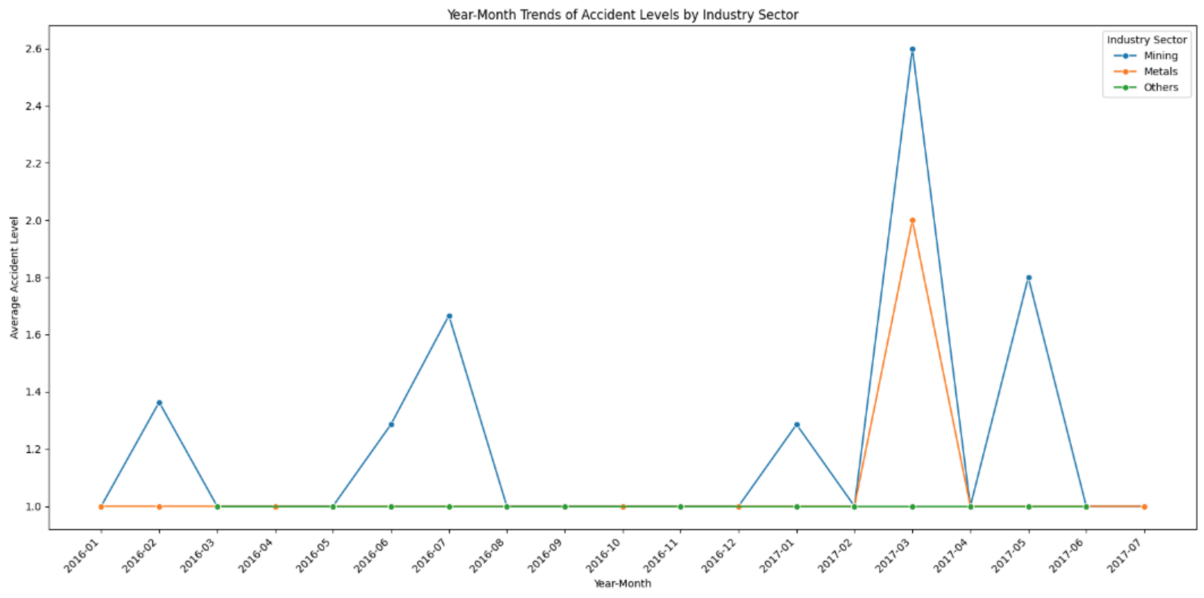
Accident Level vs Potential Accident Level by Industry Sector

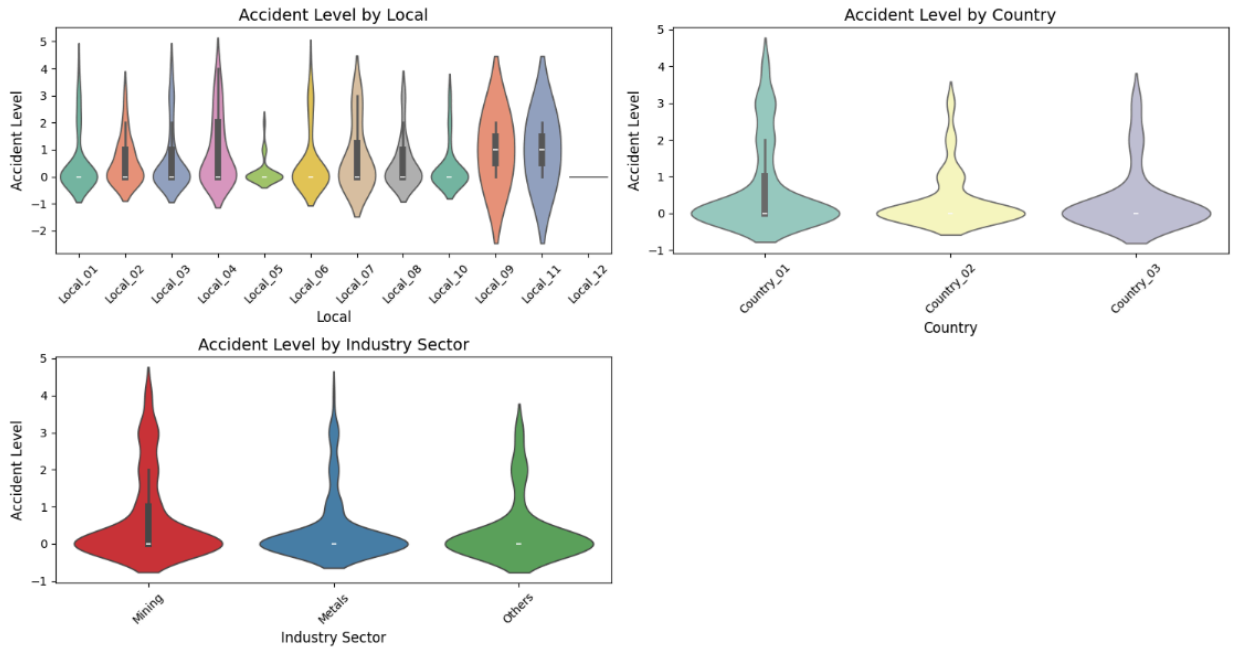


Critical Risks by Accident Level and Gender

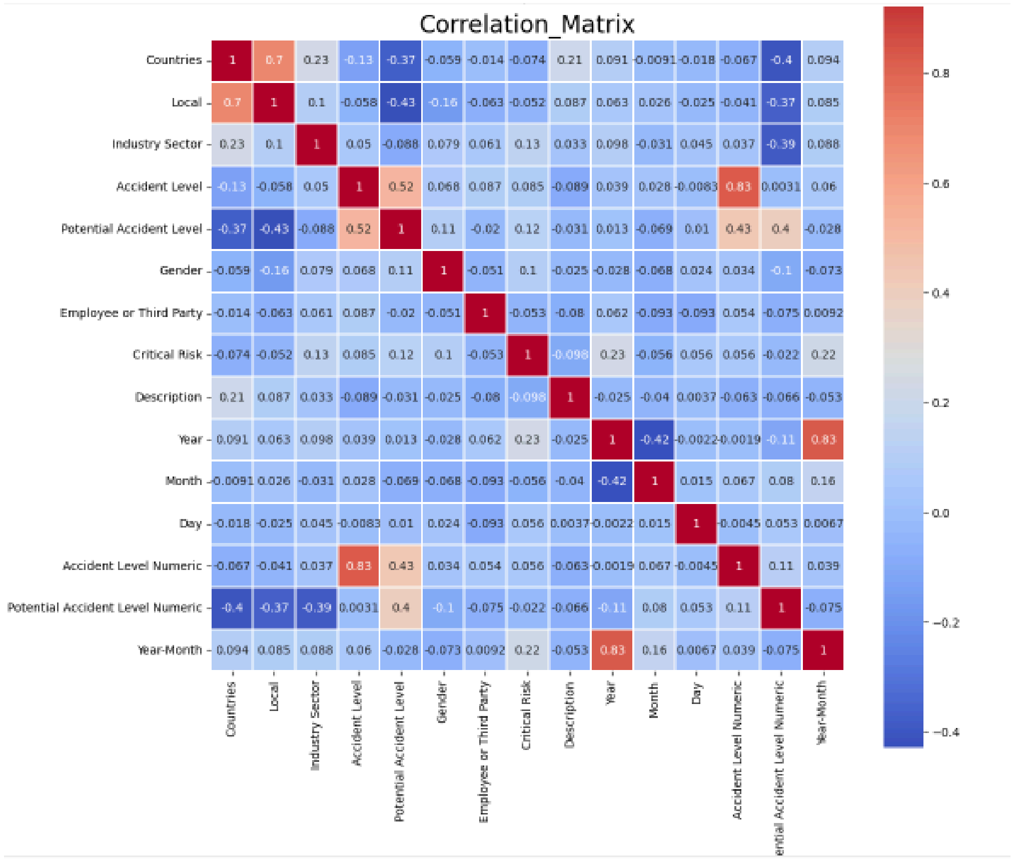


Year-Month Trends of Accident Levels by Industry Sector





Correlation Matrix



KEY INSIGHTS - Multivariate Analysis

Accident Level vs. Potential Accident Level by Industry Sector

- The violin plot reveals a broader spread of accident levels in the Mining sector, with a skew toward more severe incidents. This highlights the variability and the potential for high-severity accidents within this sector.
- Metals demonstrate a narrower distribution of accident levels, indicating fewer high-severity incidents and suggesting better control mechanisms.
- The "Others" category shows moderate variability, with some potential outliers indicating unusual incidents requiring further investigation.

Accident Level by Local and Country

- Significant variability is observed across different locales, with some exhibiting tightly concentrated distributions around lower accident levels and others showing broad distributions with higher frequencies of severe incidents.
- At the country level, accident distributions are more consistent, with fewer extreme values, indicating country-specific safety standards may play a role in moderating risk variability.

Critical Risks vs. Accident Level by Gender

- The violin plot analysis confirms that males are more frequently involved in severe incidents, as the spread of accident levels for males is broader and skewed toward higher severities.
- For females, the distribution of accident levels is narrower, suggesting fewer incidents overall, though certain risk types like "Others" still require investigation to ensure equitable safety measures.

Year-Month Trends of Accident Levels by Industry Sector

- Temporal trends corroborated by the violin plot indicate that accident levels in the Mining sector are seasonally influenced, with higher peaks during specific months. The broad variability during these peaks suggests fluctuating risk factors, possibly linked to operational cycles or environmental conditions.

- Metals exhibit steadier trends with a narrow spread, indicating a more consistent risk profile.

General Observations

- The Mining sector stands out for its high variability in accident levels, with a wider range of severity compared to other sectors. This reinforces its designation as a high-risk industry.
- The "Others" risk category displays broader and often ambiguous distributions, masking the underlying hazards that need clearer categorization.
- Gender-specific differences in accident distributions emphasize the need for tailored safety interventions to address distinct risk exposures for males and females.

RECOMMENDATIONS

- **Enhance Safety Measures in Mining**
  Address the high variability and severity of accidents in the Mining sector by implementing advanced risk management tools, particularly during seasonal peaks identified in the temporal trends.
- **Investigate and Categorize "Others" Risks**
  Refine the "Others" risk category to uncover specific hazards, enabling more targeted safety measures and reducing ambiguity in accident reporting.
- **Implement Gender-Specific Safety Programs**
  Tailor training and safety protocols for male workers exposed to high-risk activities (e.g., manual tools, pressing equipment) and ensure equitable measures for female workers to address their specific risk profiles.
- **Standardize Local Safety Practices**
  Reduce inconsistencies between locals with varying accident distributions by standardizing safety practices and addressing outliers in high-risk locations.
- **Strengthen Seasonal Safety Protocols**
  Introduce heightened safety measures in Mining during peak accident months, with resources allocated to manage higher severity incidents effectively.

- **Continuous Monitoring and Data-Driven Adjustments**
  Regularly analyze accident distributions to track shifts in central tendencies, variability, and outliers, ensuring safety strategies remain responsive to evolving risks.
- **Refine Safety in Metals Sector**
  Conduct regular safety audits in the Metals sector to maintain its consistent risk profile and investigate any emerging outliers to prevent escalation.
- **Promote Consistent Risk Management Across Countries**
  Leverage insights from the relatively uniform accident distributions at the country level to reinforce effective safety standards across all regions.

NLP - PREPROCESSING

Before proceeding with model building, it is essential to pre-process the description column in the dataset to ensure the text is clean and structured for analysis. Key pre-processing steps include text cleaning, tokenization, stopword removal, stemming or lemmatization, handling punctuation, translations, and spell checks. The specific steps chosen for this dataset are detailed in the following section.

TRANSLATION

Given that the dataset primarily originates from Brazil and may also include data from other regions, we opted to perform translation from various languages to English. This ensures consistency and standardization, making the data more suitable for analysis and model training.

```
from googletrans import Translator

# Initialize the translator
translator = Translator()

# Function to automatically detect source language and translate to English
def translate_text_auto_detect(text):
    # Check for None or empty string
    if text is None or text == '':
        return ''  # or any other appropriate default value

    try:
        # Auto-detect the source language and translate to English
        translated = translator.translate(text, dest='en')
        return translated.text  # Directly access the translated text
    except (AttributeError, TypeError):  # Handle potential errors
        print(f"Translation error for text: {text}")
        return text  # Return original text

# Apply translation to the 'Description' column
data_cleaned['Translated_Description'] = data_cleaned['Description'].apply(translate_text_auto_detect)

# Display the cleaned data
print(data_cleaned[['Description', 'Translated_Description']])
```

```
                                           Description  \
0    While removing the drill rod of the Jumbo 08 f...
1    During the activation of a sodium sulphide pum...
2    In the sub-station MILPO located at level +170...
3    Being 9:45 am. approximately in the Nv. 1880 C...
4    Approximately at 11:45 a.m. in circumstances t...
..                                                 ...
420  Being approximately 5:00 a.m. approximately, w...
421  The collaborator moved from the infrastructure...
422  During the environmental monitoring activity i...
423  The Employee performed the activity of strippi...
424  At 10:00 a.m., when the assistant cleaned the ...

                                Translated_Description
0    While removing the drill rod of the Jumbo 08 f...
1    During the activation of a sodium sulphide pum...
2    In the sub-station MILPO located at level +170...
3    Being 9:45 am. approximately in the Nv. 1880 C...
4    Approximately at 11:45 a.m. in circumstances t...
..                                                 ...
420  Being approximately 5:00 a.m. approximately, w...
421  The collaborator moved from the infrastructure...
422  During the environmental monitoring activity i...
423  The Employee performed the activity of strippi...
424  At 10:00 a.m., when the assistant cleaned the ...

[411 rows x 2 columns]
```

CLEANUP STEPS

Handle Missing Text: If the input text is empty or None, return an empty string.

Convert to Lowercase: The text is converted to lowercase to standardize it.

Tokenization: The text is split into individual words (tokens).

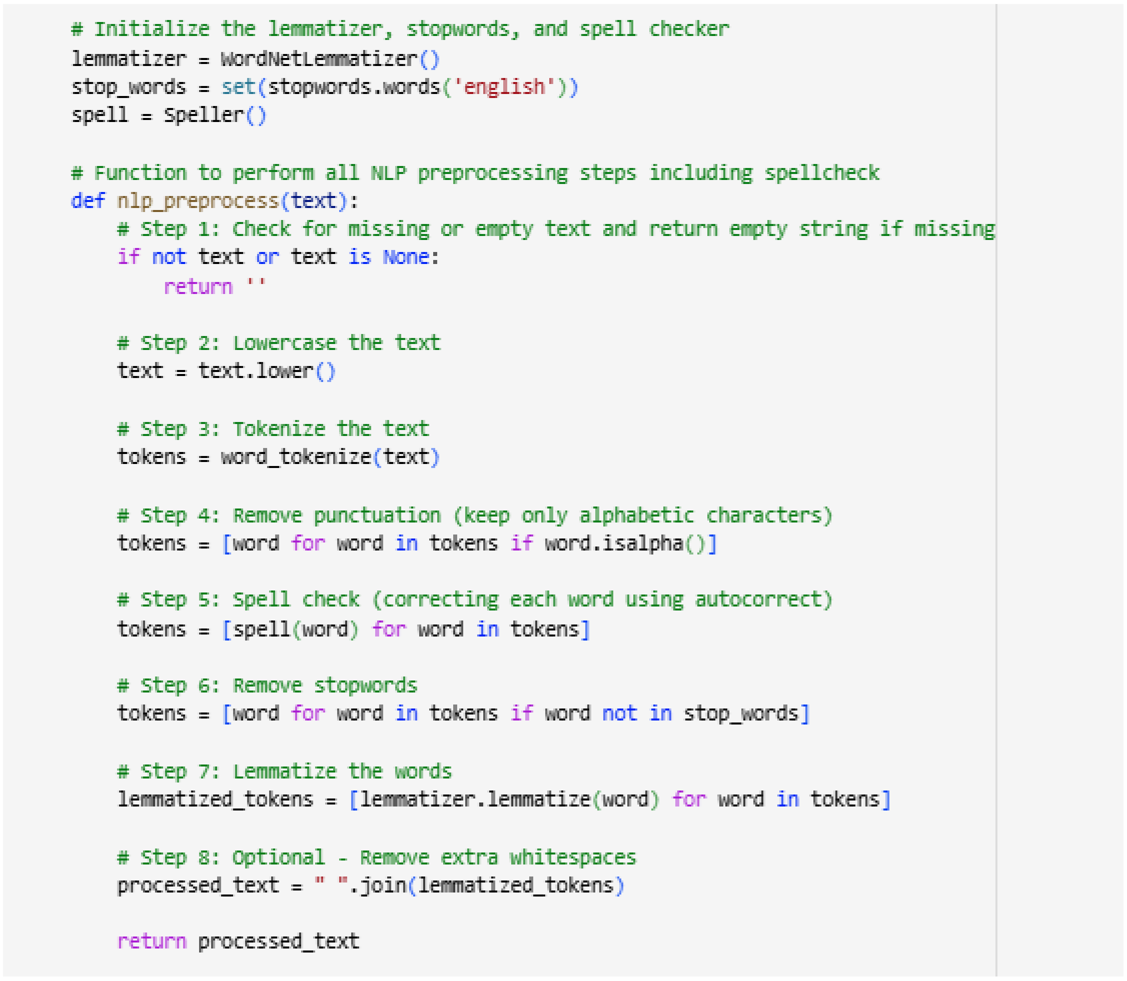
Remove Punctuation: Only alphabetic characters are retained, removing any punctuation.

Spell Checking: Each word is spell-checked and corrected using the autocorrect function.

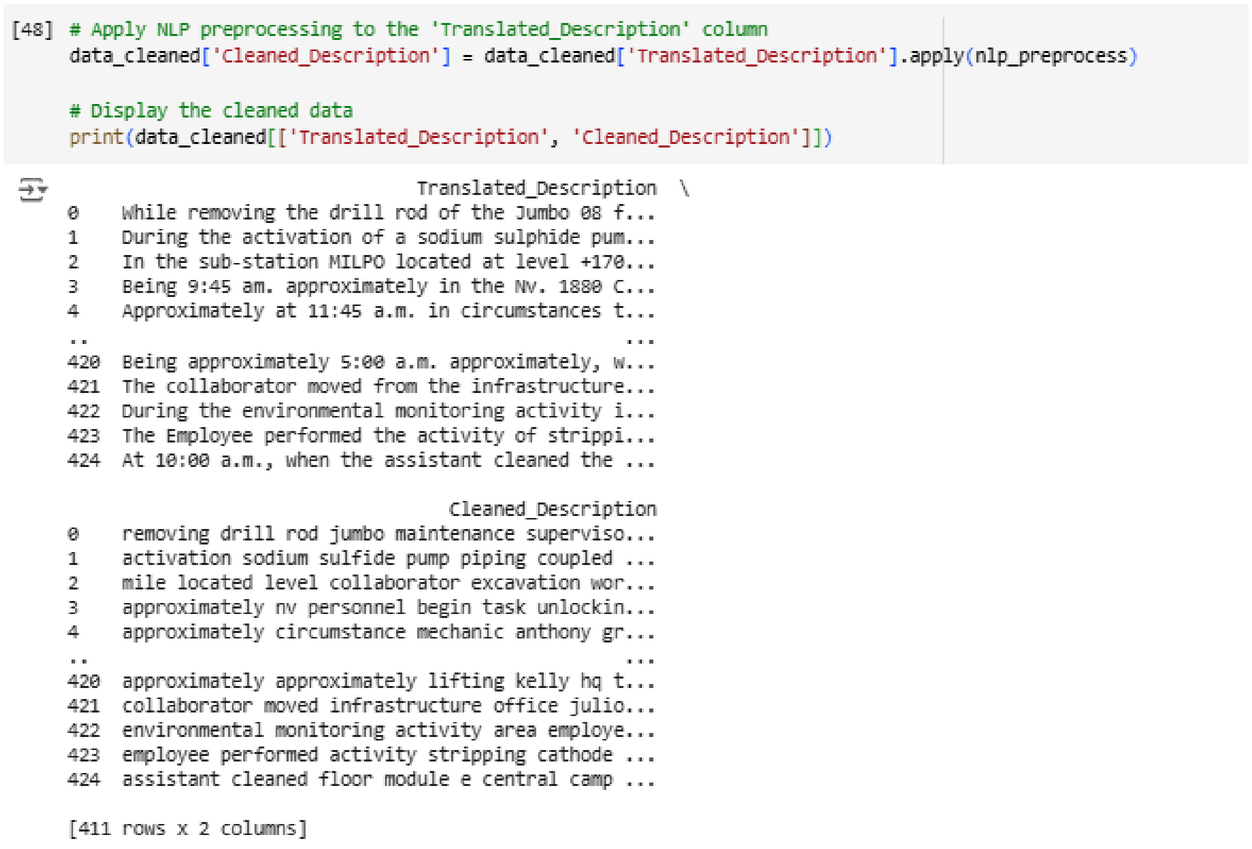
Remove Stopwords: Common stopwords (e.g., "the", "is") are removed to focus on important words.

Lemmatization: Words are reduced to their base form (e.g., "running" to "run"). We choose lemmatization over stemming as it gives better accuracy for model training.

Whitespace Removal: Extra whitespaces are removed, and the tokens are joined back into a clean string.



```python
# Initialize the lemmatizer, stopwords, and spell checker
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))
spell = Speller()

# Function to perform all NLP preprocessing steps including spellcheck
def nlp_preprocess(text):
    # Step 1: Check for missing or empty text and return empty string if missing
    if not text or text is None:
        return ''

    # Step 2: Lowercase the text
    text = text.lower()

    # Step 3: Tokenize the text
    tokens = word_tokenize(text)

    # Step 4: Remove punctuation (keep only alphabetic characters)
    tokens = [word for word in tokens if word.isalpha()]

    # Step 5: Spell check (correcting each word using autocorrect)
    tokens = [spell(word) for word in tokens]

    # Step 6: Remove stopwords
    tokens = [word for word in tokens if word not in stop_words]

    # Step 7: Lemmatize the words
    lemmatized_tokens = [lemmatizer.lemmatize(word) for word in tokens]

    # Step 8: Optional - Remove extra whitespaces
    processed_text = " ".join(lemmatized_tokens)

    return processed_text
```



```
[48] # Apply NLP preprocessing to the 'Translated_Description' column
     data_cleaned['Cleaned_Description'] = data_cleaned['Translated_Description'].apply(nlp_preprocess)

     # Display the cleaned data
     print(data_cleaned[['Translated_Description', 'Cleaned_Description']])
```

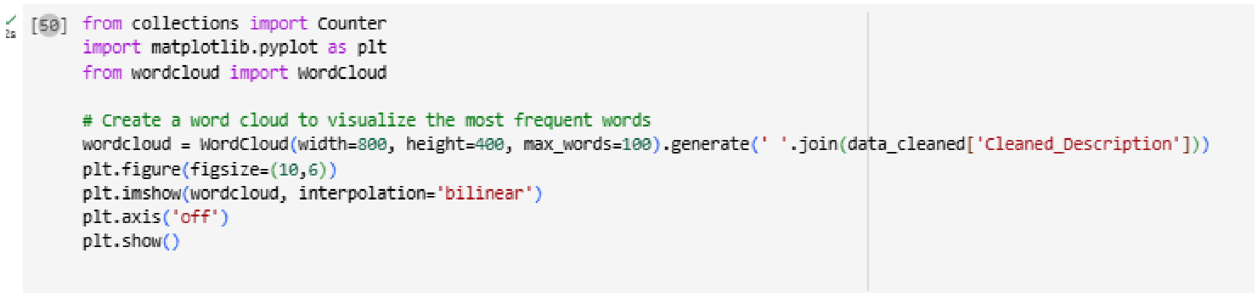
```
                               Translated_Description  \
0    While removing the drill rod of the Jumbo 08 f...
1    During the activation of a sodium sulphide pum...
2    In the sub-station MILPO located at level +170...
3    Being 9:45 am. approximately in the Nv. 1880 C...
4    Approximately at 11:45 a.m. in circumstances t...
..                                                 ...
420  Being approximately 5:00 a.m. approximately, w...
421  The collaborator moved from the infrastructure...
422  During the environmental monitoring activity i...
423  The Employee performed the activity of strippi...
424  At 10:00 a.m., when the assistant cleaned the ...

                                  Cleaned_Description
0    removing drill rod jumbo maintenance superviso...
1    activation sodium sulfide pump piping coupled ...
2    mile located level collaborator excavation wor...
3    approximately nv personnel begin task unlockin...
4    approximately circumstance mechanic anthony gr...
..                                                 ...
420  approximately approximately lifting kelly hq t...
421  collaborator moved infrastructure office julio...
422  environmental monitoring activity area employe...
423  employee performed activity stripping cathode ...
424  assistant cleaned floor module e central camp ...

[411 rows x 2 columns]
```

OBSERVATIONS

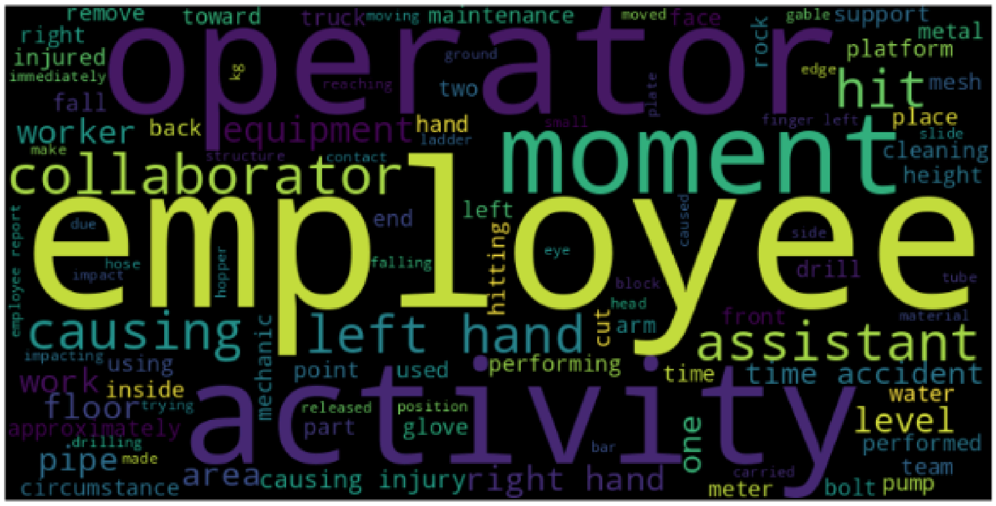
- The cleaned descriptions show that functional words like "the," "a," and "in" have been removed, making the text more concise and focused.
- Text is converted to lowercase, ensuring uniformity and treating capitalized and non-capitalized words as the same.
- Punctuation marks have been eliminated, simplifying the text for easier processing in NLP tasks.
- Non-alphabetic characters are removed, and words are tokenized, keeping only the relevant terms.
- Spelling errors are corrected, such as changing "sulphide" to "sulfide," ensuring consistent terminology.
- Lemmatization has reduced words to their base forms, improving consistency and preparing the text for further analysis.

VISUALIZING THE DATA (NLP)

WORD FREQUENCY DISTRIBUTION



```
[50] from collections import Counter
     import matplotlib.pyplot as plt
     from wordcloud import WordCloud

     # Create a word cloud to visualize the most frequent words
     wordcloud = WordCloud(width=800, height=400, max_words=100).generate(' '.join(data_cleaned['Cleaned_Description']))
     plt.figure(figsize=(10,6))
     plt.imshow(wordcloud, interpolation='bilinear')
     plt.axis('off')
     plt.show()
```



OBSERVATIONS

Dominant Keywords:

- Words like "employee," "activity," "moment," "operator," and "assistant" appear prominently, indicating frequent mentions in the dataset.
- These terms likely reflect the primary subjects or roles associated with incidents or actions.

Focus on Actions and Context:

- Words such as "causing," "hitting," "injury," and "performing" suggest the dataset is related to workplace incidents, with emphasis on actions leading to specific outcomes.
- Context-related terms like "left hand," "equipment," and "floor" highlight common areas or objects associated with these incidents.

Potential Risk Areas:

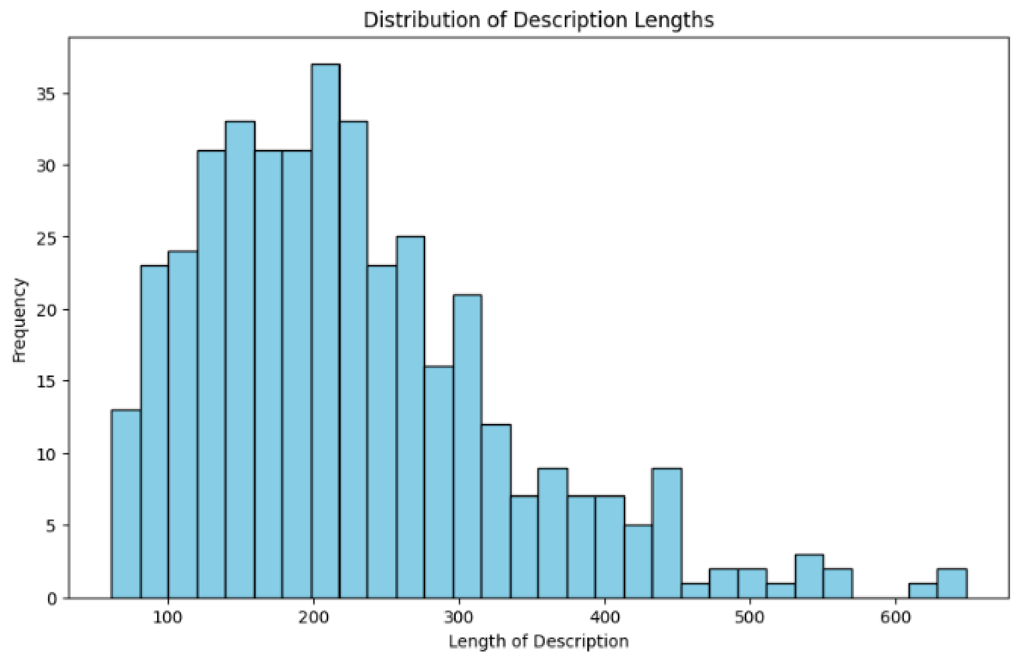
- Terms like "injury," "drill," "equipment," and "cleaning" point toward tasks or tools potentially associated with workplace hazards.
- The frequent mention of "causing" and "accident" suggests an analysis of causes and effects within the dataset.

Relevance of Specific Body Parts:

- "Left hand" and "right hand" are highlighted, suggesting a focus on injuries involving hands, which may represent a high-risk area in the workplace.

DISTRIBUTION OF TEXT LENGTH

```
In [51]: # Add a column for text length
         data_cleaned['Description_Length'] = data_cleaned['Cleaned_Description'].apply(len)

         # Plot the distribution of text length
         plt.figure(figsize=(10,6))
         plt.hist(data_cleaned['Description_Length'], bins=30, color='skyblue', edgecolor='black')
         plt.title('Distribution of Description Lengths')
         plt.xlabel('Length of Description')
         plt.ylabel('Frequency')
         plt.show()
```



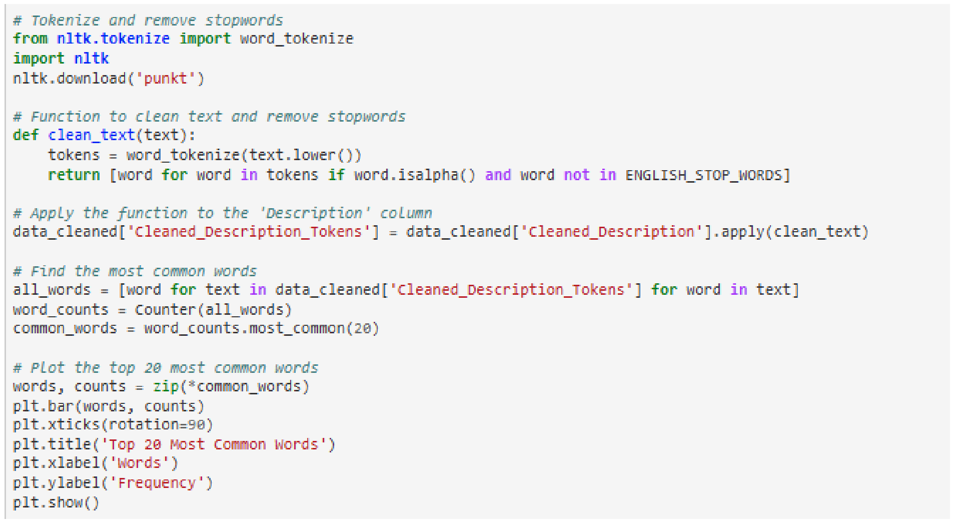
OBSERVATIONS

- **Right-Skewed Distribution**: Most descriptions are short, with fewer being much longer.
- **Frequent Length Range**: Descriptions mostly fall between 100-250 characters, peaking around 150-200.
- **Long Descriptions**: A small number exceed 300 characters, with some surpassing 600, likely indicating more complex incidents.
- **Mode**: The most common description length is slightly under 200 characters.

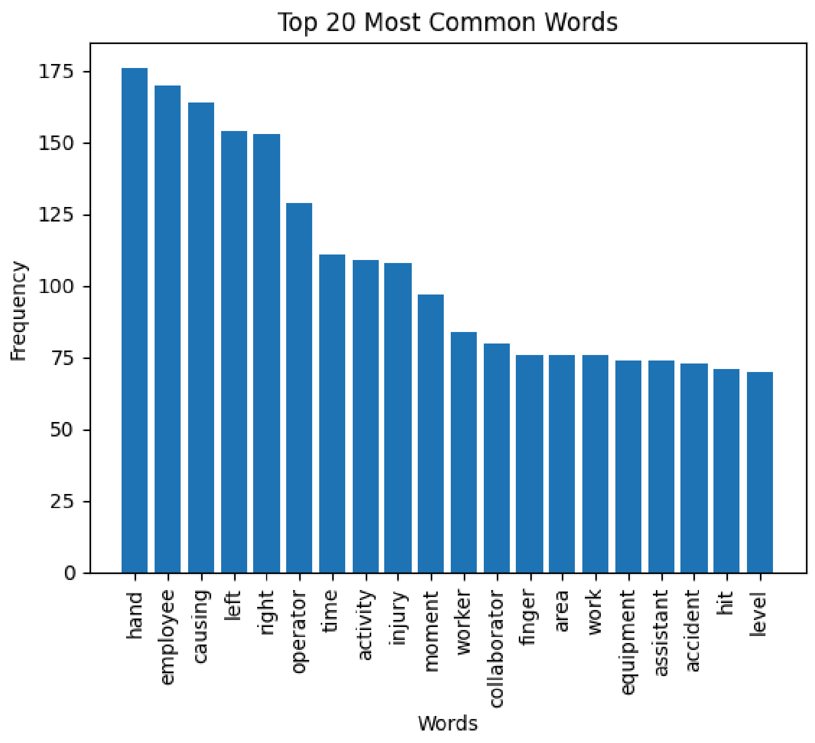
RECOMMENDATIONS

- **Modeling Considerations**: The prevalence of short descriptions may bias NLP models, requiring strategies for truncation or padding.
- **Text Augmentation**: Augmenting long descriptions or summarizing lengthy ones can address the imbalance.
- **Data Quality Check**: Review long outliers for redundancy or over-detailing and trim for consistency.
- **Segmentation by Length**: Segment descriptions by length to analyze any correlation with accident severity.

TOP N MOST COMMON WORDS



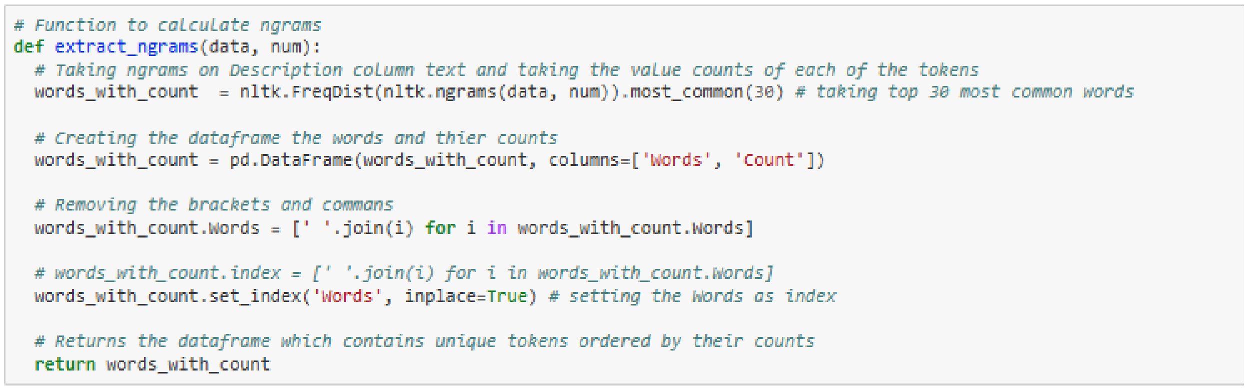
```python
# Tokenize and remove stopwords
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt')

# Function to clean text and remove stopwords
def clean_text(text):
    tokens = word_tokenize(text.lower())
    return [word for word in tokens if word.isalpha() and word not in ENGLISH_STOP_WORDS]

# Apply the function to the 'Description' column
data_cleaned['Cleaned_Description_Tokens'] = data_cleaned['Cleaned_Description'].apply(clean_text)

# Find the most common words
all_words = [word for text in data_cleaned['Cleaned_Description_Tokens'] for word in text]
word_counts = Counter(all_words)
common_words = word_counts.most_common(20)

# Plot the top 20 most common words
words, counts = zip(*common_words)
plt.bar(words, counts)
plt.xticks(rotation=90)
plt.title('Top 20 Most Common Words')
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.show()
```



OBSERVATIONS

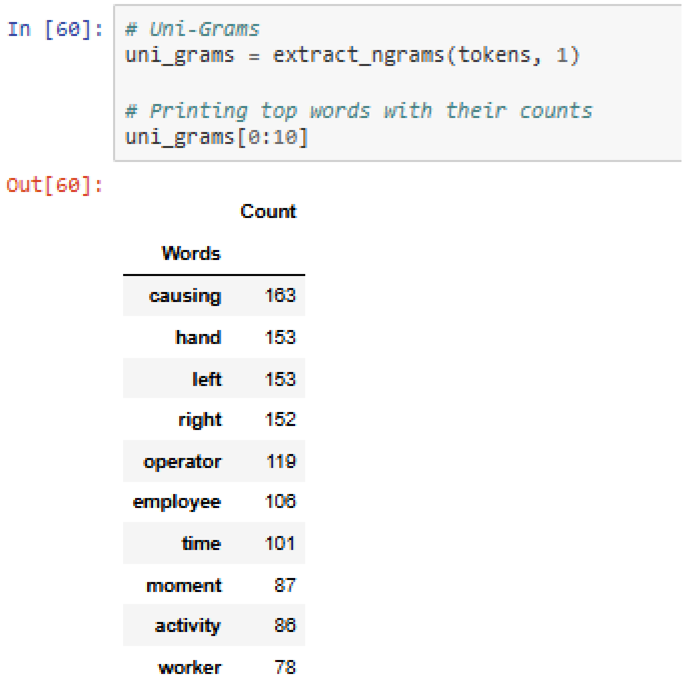
- **Frequent Hand Injuries** – "Hand" and "finger" are the most common words, indicating frequent hand-related incidents.
- **Employee-Centered** – Words like "employee," "operator," and "worker" highlight workplace accidents involving staff.
- **Injury and Direction** – Terms like "injury," "left," and "right" suggest incidents often specify body parts and sides.
- **Task and Time Focus** – "Activity," "moment," and "time" indicate descriptions detail when and during which tasks accidents occur.
- **Equipment and Area** – The presence of "equipment" and "area" points to machinery and specific work zones as common incident locations.

N-GRAMS



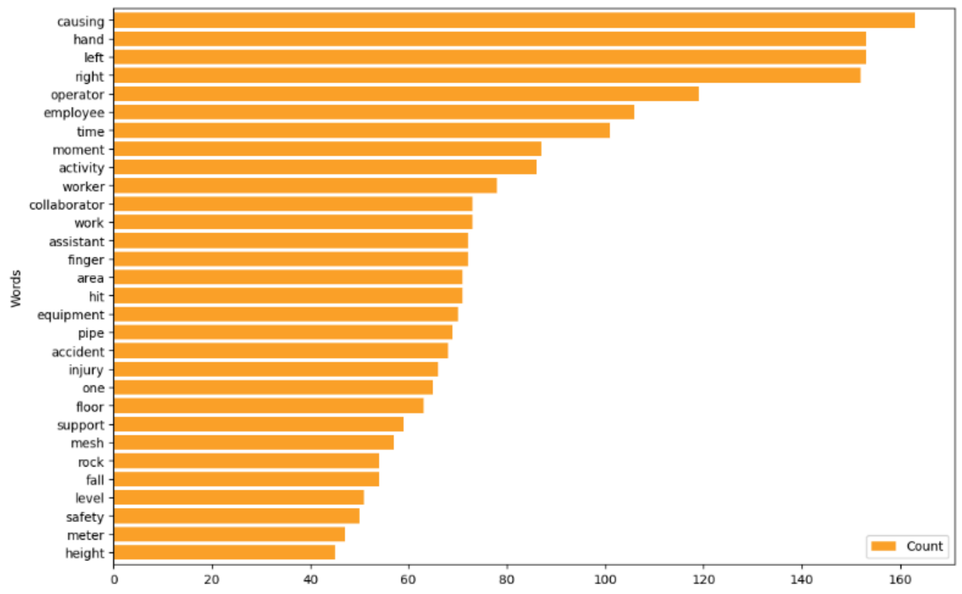
```python
# Function to calculate ngrams
def extract_ngrams(data, num):
  # Taking ngrams on Description column text and taking the value counts of each of the tokens
  words_with_count  = nltk.FreqDist(nltk.ngrams(data, num)).most_common(30) # taking top 30 most common words

  # Creating the dataframe the words and thier counts
  words_with_count = pd.DataFrame(words_with_count, columns=['Words', 'Count'])

  # Removing the brackets and commans
  words_with_count.Words = [' '.join(i) for i in words_with_count.Words]

  # words_with_count.index = [' '.join(i) for i in words_with_count.Words]
  words_with_count.set_index('Words', inplace=True) # setting the Words as index

  # Returns the dataframe which contains unique tokens ordered by their counts
  return words_with_count
```

UNI-GRAMS



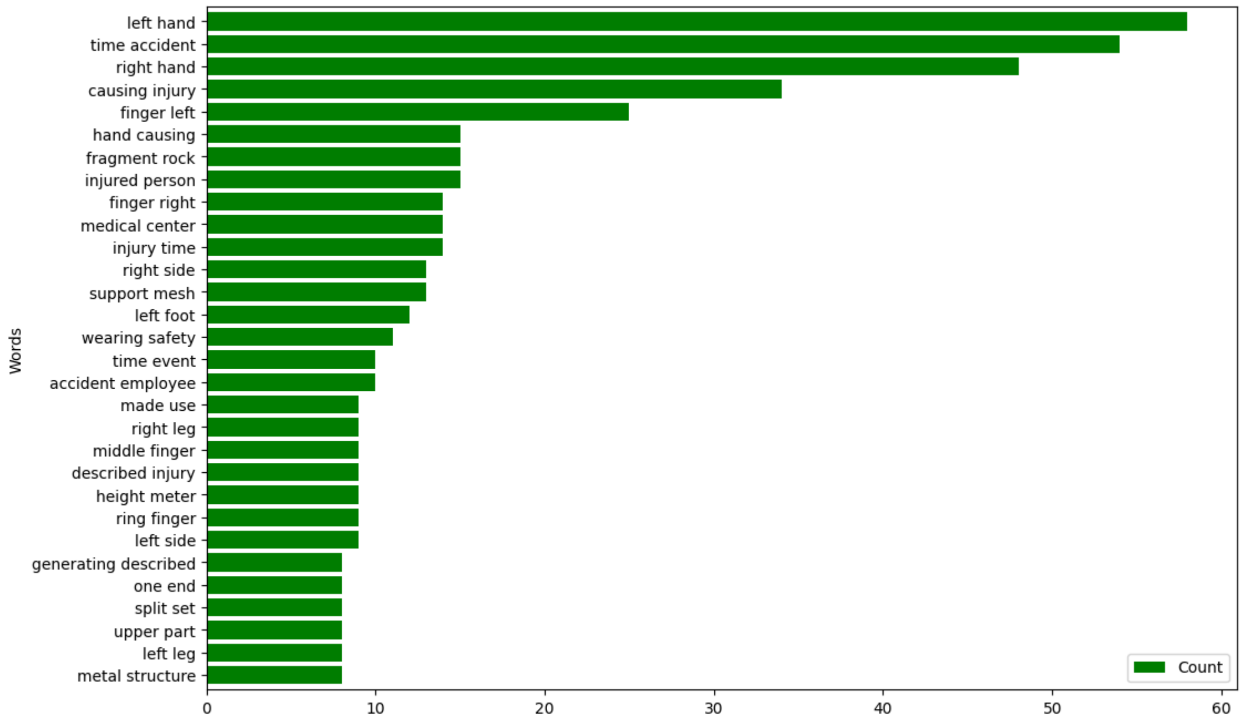
```
In [60]: # Uni-Grams
         uni_grams = extract_ngrams(tokens, 1)

         # Printing top words with their counts
         uni_grams[0:10]

Out[60]:
                 Count
            Words
          causing  163
             hand  153
             left  153
            right  152
         operator  119
         employee  106
             time  101
           moment   87
         activity   86
           worker   78
```

```
In [61]: # Visualising the ngrams
         uni_grams.sort_values(by='Count').plot.barh(color = 'orange', width = 0.8, figsize = (12,8));
```

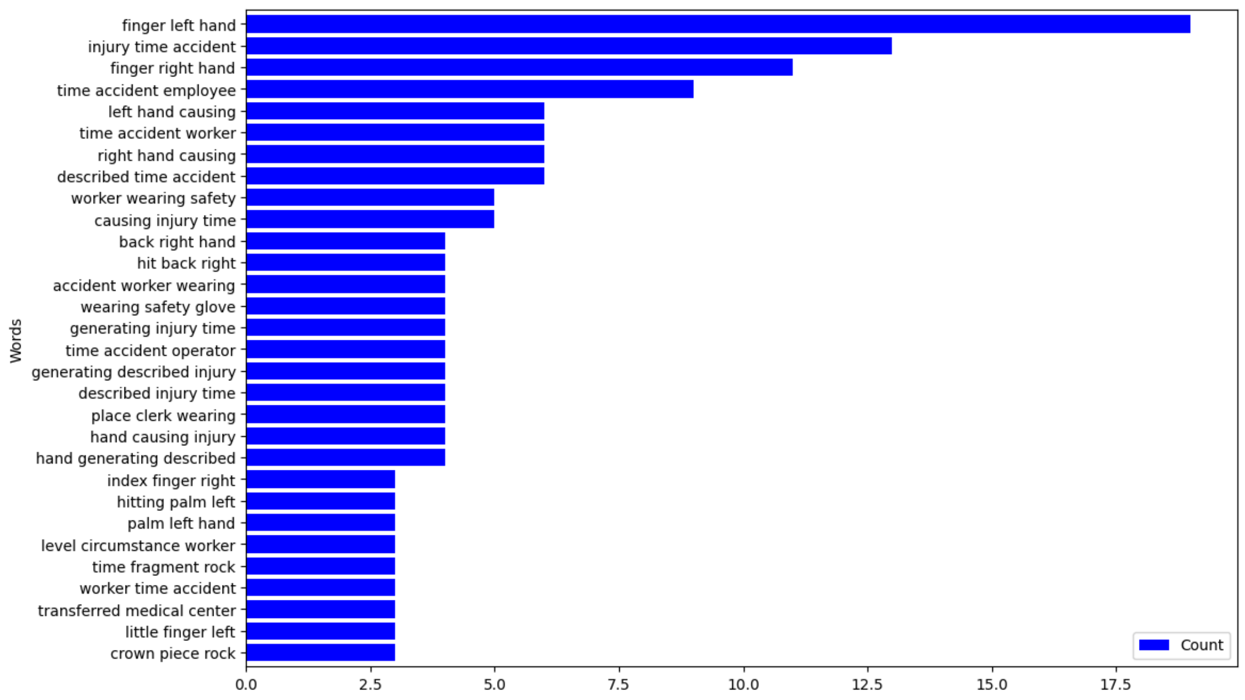


Similarly…

BI-GRAMS



TRI-GRAMS



OBSERVATIONS FROM N-GRAMS:

Focus on Hands and Injuries

- Across all n-grams, terms like **"hand," "left hand," "finger left hand"** dominant, highlighting frequent **hand-related injuries**. This suggests a focus on **manual work hazards**.

Employee and Worker-Centric

- Words like **"employee," "worker," "collaborator"** frequently appear, indicating that **workplace incidents involving staff** are common. Safety measures should target employee protection.

Directional and Specific Injuries

- Phrases like **"left hand," "right hand," "finger left"** suggest that incident reports often **specify the body side and part** affected. This can inform **ergonomic and safety gear improvements**.

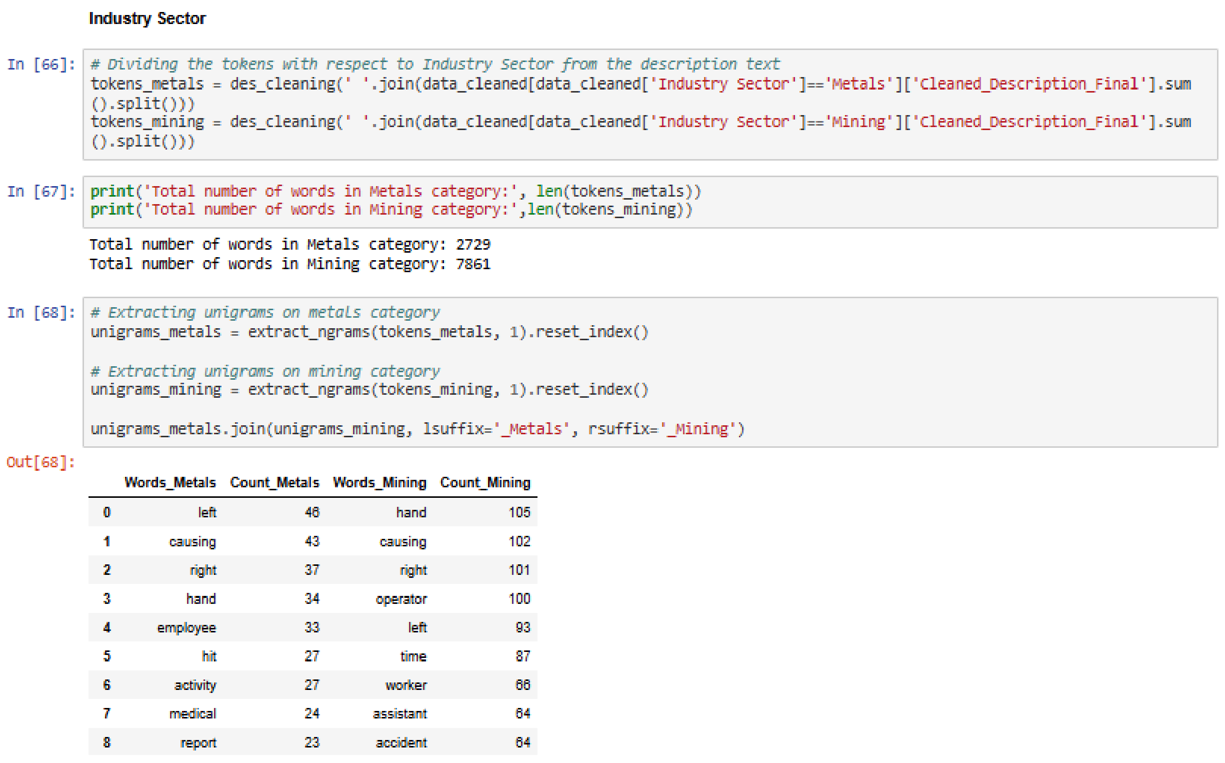
Frequent Causes and Activities

- Bigrams and trigrams like **"causing injury," "time accident," "described injury time"** reflect a focus on **causal factors** and the **timing of incidents**. This indicates **incident reporting emphasizes root causes and accident timelines**.

Equipment and Environmental Factors

- Unigrams such as **"equipment, pipe, area"** and phrases like **"fragment rock"** suggest that incidents often involve **machinery, tools, or environmental hazards**. Equipment maintenance and area monitoring are essential.

N-GRAMS BY INDUSTRY SECTORS



Industry Sector

```
In [66]: # Dividing the tokens with respect to Industry Sector from the description text
         tokens_metals = des_cleaning(' '.join(data_cleaned[data_cleaned['Industry Sector']=='Metals']['Cleaned_Description_Final'].sum
         ().split()))
         tokens_mining = des_cleaning(' '.join(data_cleaned[data_cleaned['Industry Sector']=='Mining']['Cleaned_Description_Final'].sum
         ().split()))
```

```
In [67]: print('Total number of words in Metals category:', len(tokens_metals))
         print('Total number of words in Mining category:',len(tokens_mining))

         Total number of words in Metals category: 2729
         Total number of words in Mining category: 7861
```

```
In [68]: # Extracting unigrams on metals category
         unigrams_metals = extract_ngrams(tokens_metals, 1).reset_index()

         # Extracting unigrams on mining category
         unigrams_mining = extract_ngrams(tokens_mining, 1).reset_index()

         unigrams_metals.join(unigrams_mining, lsuffix='_Metals', rsuffix='_Mining')
```

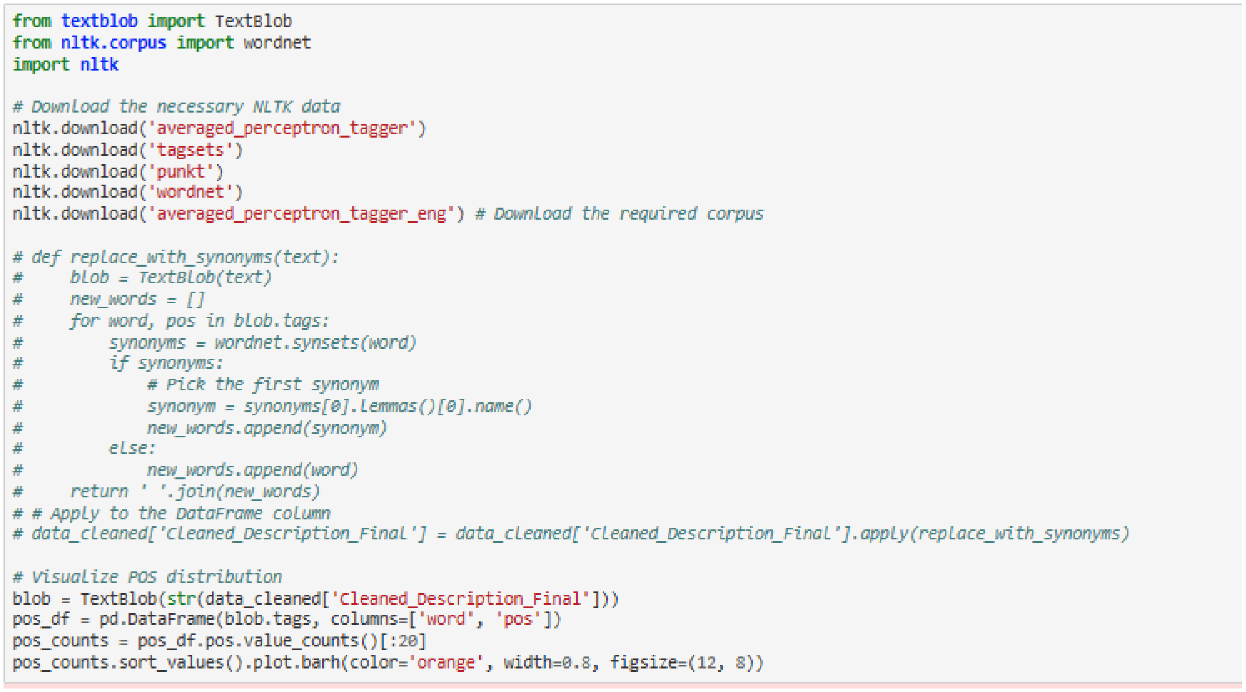
Out[68]:

	Words_Metals	Count_Metals	Words_Mining	Count_Mining
0	left	46	hand	105
1	causing	43	causing	102
2	right	37	right	101
3	hand	34	operator	100
4	employee	33	left	93
5	hit	27	time	87
6	activity	27	worker	66
7	medical	24	assistant	64
8	report	23	accident	64

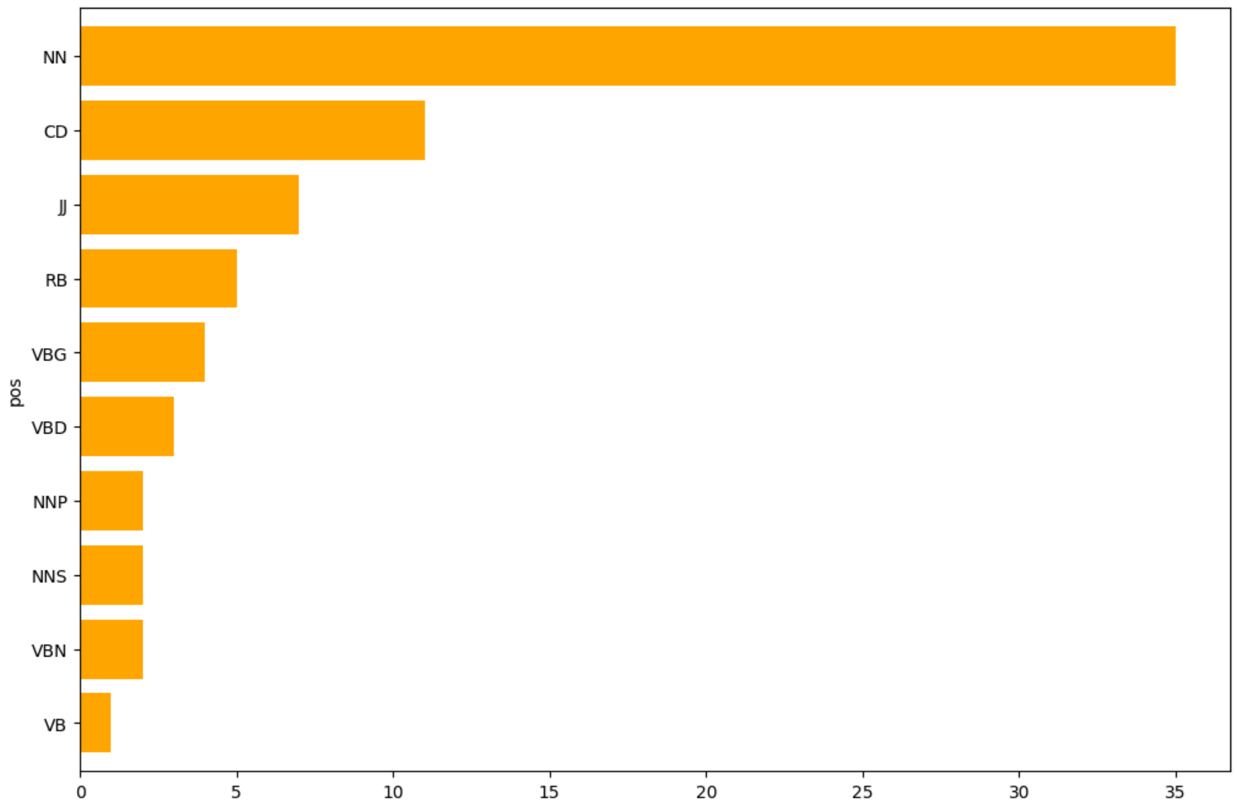
OBSERVATIONS AND INSIGHTS:

- **Hand Injuries Dominate** – "Hand" and "left/right" are top terms, indicating frequent **hand-related incidents** in both sectors.
- **Focus on Causes** – "Causing" and "time/moment" highlight emphasis on **incident causes and timing**.
- **Worker-Centered** – "Employee, operator, worker" show **human factors** are key in accidents.
- **Equipment Risks** – Mining involves "equipment, pipe, rock," while metals show risks with **"hose, pump, acid"**.
- **Injury and Safety** – "Finger, injury, fall, cut" reflect focus on **personal injury and safety measures**.

POS Tagging



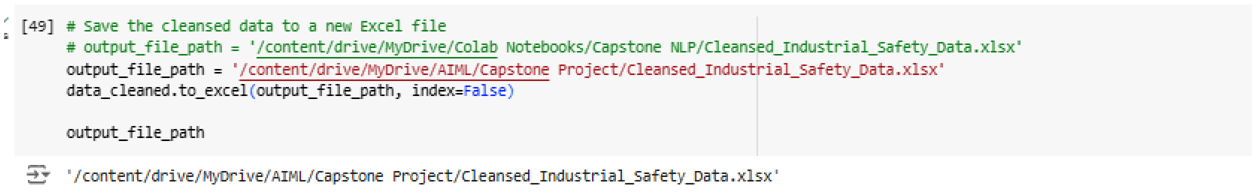
```
from textblob import TextBlob
from nltk.corpus import wordnet
import nltk

# Download the necessary NLTK data
nltk.download('averaged_perceptron_tagger')
nltk.download('tagsets')
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger_eng') # Download the required corpus

# def replace_with_synonyms(text):
#     blob = TextBlob(text)
#     new_words = []
#     for word, pos in blob.tags:
#         synonyms = wordnet.synsets(word)
#         if synonyms:
#             # Pick the first synonym
#             synonym = synonyms[0].lemmas()[0].name()
#             new_words.append(synonym)
#         else:
#             new_words.append(word)
#     return ' '.join(new_words)
# # Apply to the DataFrame column
# data_cleaned['Cleaned_Description_Final'] = data_cleaned['Cleaned_Description_Final'].apply(replace_with_synonyms)

# Visualize POS distribution
blob = TextBlob(str(data_cleaned['Cleaned_Description_Final']))
pos_df = pd.DataFrame(blob.tags, columns=['word', 'pos'])
pos_counts = pos_df.pos.value_counts()[:20]
pos_counts.sort_values().plot.barh(color='orange', width=0.8, figsize=(12, 8))
```



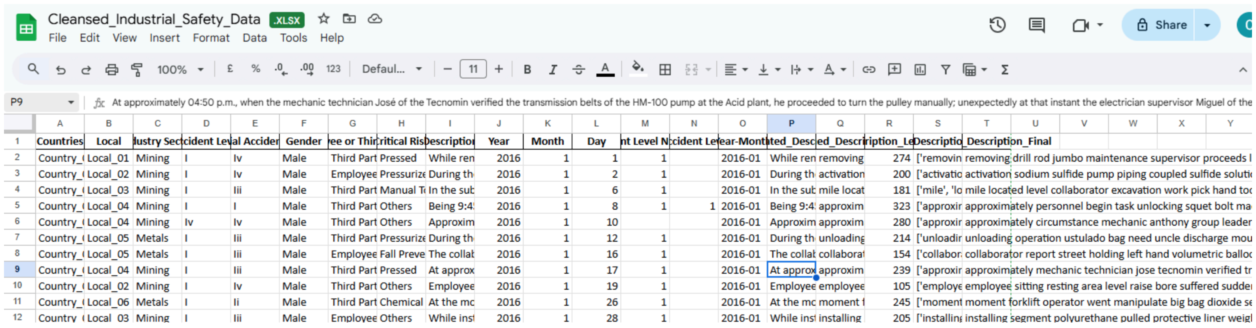
OBSERVATIONS AND INSIGHTS:

- **Nouns (NN) Dominate** – Focus on **objects and entities** like "hand" and "worker."
- **Frequent Numbers (CD)** – Highlights **measurements and quantities** in reports.
- **Descriptive Adjectives (JJ)** – Emphasizes **qualities** of objects/incidents.
- **Actions and Adverbs (RB, VBG, VBD)** – Reflects **detailed event descriptions**.
- **Proper and Plural Nouns (NNP, NNS)** – References to **specific items and multiple objects**.

EXPORTED TO EXCEL FORMAT



```
[49] # Save the cleansed data to a new Excel file
     # output_file_path = '/content/drive/MyDrive/Colab Notebooks/Capstone NLP/Cleansed_Industrial_Safety_Data.xlsx'
     output_file_path = '/content/drive/MyDrive/AIML/Capstone Project/Cleansed_Industrial_Safety_Data.xlsx'
     data_cleaned.to_excel(output_file_path, index=False)

     output_file_path

'/content/drive/MyDrive/AIML/Capstone Project/Cleansed_Industrial_Safety_Data.xlsx'
```



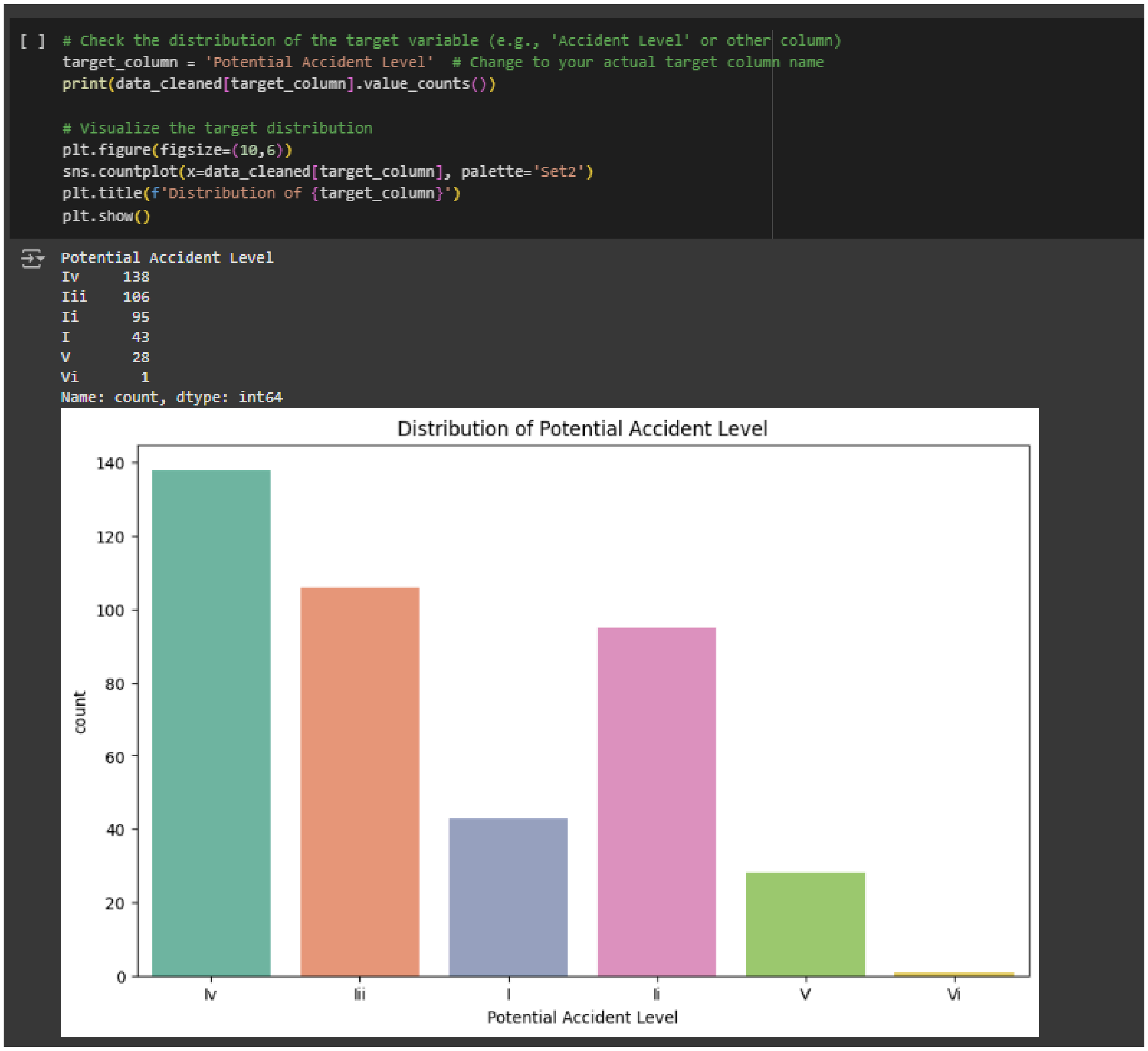
At approximately 04:50 p.m., when the mechanic technician José of the Tecnomin verified the transmission belts of the HM-100 pump at the Acid plant, he proceeded to turn the pulley manually; unexpectedly at that instant the electrician supervisor Miguel of the

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	Countries	Local	lustry Sec	cident Lev	al Accider	Gender	ee or Thir	ritical Ris	Descriptio	Year	Month	Day	nt Level N	cident Le	ear-Mont	ted_Desc	ed_Descr	iption_Le	Descriptio	Description_Final
2	Country_(	Local_01	Mining	I	Iv	Male	Third Part	Pressed	While ren	2016	1	1	1		2016-01	While ren	removing	274	['removin	removing drill rod jumbo maintenance supervisor proceeds l
3	Country_(	Local_02	Mining	I	Iv	Male	Employee	Pressurize	During th	2016	1	2	1		2016-01	During th	activation	200	['activatio	activation sodium sulfide pump piping coupled sulfide soluti
4	Country_(	Local_03	Mining	I	Iii	Male	Third Part	Manual T	In the sub	2016	1	6	1		2016-01	In the sub	mile locat	181	['mile', 'lo	mile located level collaborator excavation work pick hand too
5	Country_(	Local_04	Mining	I	I	Male	Third Part	Others	Being 9:4	2016	1	8	1	1	2016-01	Being 9:4	approxim	323	['approxir	approximately personnel begin task unlocking squet bolt ma
6	Country_(	Local_04	Mining	Iv	Iv	Male	Third Part	Others	Approxim	2016	1	10			2016-01	Approxim	approxim	280	['approxir	approximately circumstance mechanic anthony group leader
7	Country_(	Local_05	Metals	I	Iii	Male	Third Part	Pressurize	During th	2016	1	12	1		2016-01	During th	unloading	214	['unloadir	unloading operation ustulado bag need uncle discharge mou
8	Country_(	Local_05	Metals	I	Iii	Male	Employee	Fall Preve	The collab	2016	1	16	1		2016-01	The collab	collaborat	154	['collabor	collaborator report street holding left hand volumetric balloo
9	Country_(	Local_04	Mining	I	Iii	Male	Third Part	Pressed	At approx	2016	1	17	1		2016-01	At approx	approxim	239	['approxir	approximately mechanic technician jose tecnomin verified tr
10	Country_(	Local_02	Mining	I	Iv	Male	Third Part	Others	Employee	2016	1	19	1		2016-01	Employee	employee	105	['employe	employee sitting resting area level raise bore suffered sudde
11	Country_(	Local_06	Metals	I	Ii	Male	Third Part	Chemical	At the mc	2016	1	26	1		2016-01	At the mc	moment f	245	['moment	moment forklift operator went manipulate big bag dioxide se
12	Country_(	Local_03	Mining	I	Iii	Male	Employee	Others	While inst	2016	1	28	1		2016-01	While inst	installing	205	['installin	installing segment polyurethane pulled protective liner weigl

DECIDING MODELS AND MODEL BUILDING

CHECK FOR IMBALANCES

We examined the data for imbalance and analyzed the distribution of the target variable along with the visualization:



```
[ ] # Check the distribution of the target variable (e.g., 'Accident Level' or other column)
    target_column = 'Potential Accident Level'  # Change to your actual target column name
    print(data_cleaned[target_column].value_counts())

    # Visualize the target distribution
    plt.figure(figsize=(10,6))
    sns.countplot(x=data_cleaned[target_column], palette='Set2')
    plt.title(f'Distribution of {target_column}')
    plt.show()
```

```
Potential Accident Level
Iv     138
Iii    106
Ii      95
I       43
V       28
Vi       1
Name: count, dtype: int64
```

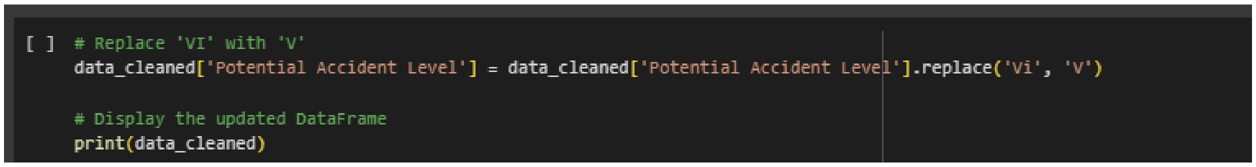
OBSERVATIONS AND INSIGHTS

- We merged Class VI with Class V, aimed at reducing class imbalance or simplifying the classification by combining rare categories.
- The target variable is Potential Accident Level, and its distribution has been visualized.
- A count plot shows the following class distribution:
  - **IV:** 138
  - **III:** 106
  - **II:** 95

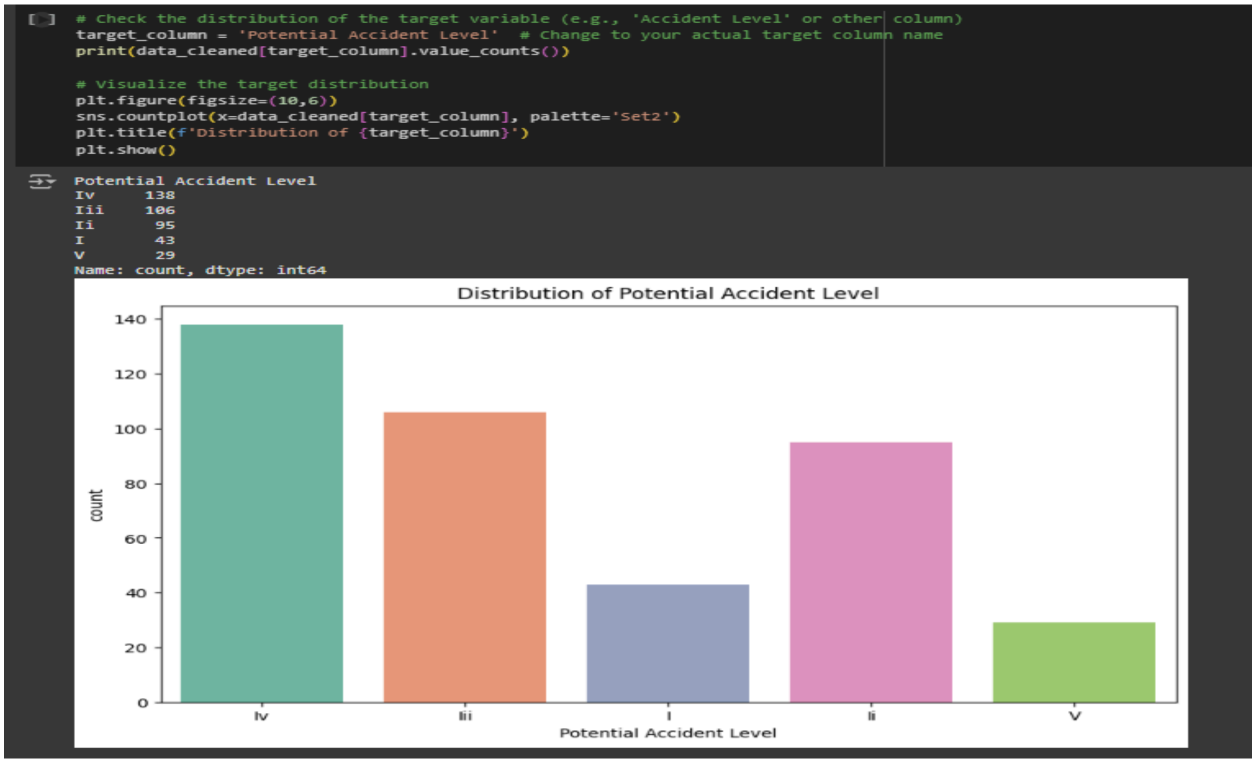
- **I:** 43
- **V (merged with VI):** 29

- There is a noticeable imbalance in the data. Class IV has the highest number of incidents, while Class V (formerly VI) has the fewest.

MERGING POTENTIAL ACCIDENT LEVEL VI WITH V



```
[ ] # Replace 'VI' with 'V'
    data_cleaned['Potential Accident Level'] = data_cleaned['Potential Accident Level'].replace('Vi', 'V')

    # Display the updated DataFrame
    print(data_cleaned)
```



```
# Check the distribution of the target variable (e.g., 'Accident Level' or other column)
target_column = 'Potential Accident Level'  # Change to your actual target column name
print(data_cleaned[target_column].value_counts())

# Visualize the target distribution
plt.figure(figsize=(10,6))
sns.countplot(x=data_cleaned[target_column], palette='Set2')
plt.title(f'Distribution of {target_column}')
plt.show()
```

```
Potential Accident Level
Iv      138
Iii     106
Ii       95
I        43
V        29
Name: count, dtype: int64
```

OBSERVATIONS FROM THE PLOT AND DISTRIBUTION

- **Class Imbalance:**
  - The imbalance might affect model performance, especially for underrepresented classes (Class V). Consider techniques like oversampling (SMOTE) or class-weight adjustments during model training.
- **Dominance of Certain Classes:**

    - Classes IV and III dominate, while Class I and V appear less frequently. This might influence how well the model predicts rare but critical events.
- **Possible Data Issues:**
    - If Class V represents severe incidents, the lower frequency could imply underreporting or less frequent severe accidents.

UPSAMPLING OF TARGET VARIABLE

- **Before SMOTE**: The target class distribution was skewed, with class IV having the most samples (138) and class V having the least (29).
- **Problem**: Imbalanced datasets can lead to:
    - **Bias** towards majority classes.
    - **Poor recall** for minority classes.
    - **Inaccurate models** that perform well overall but fail to predict minority class instances correctly.

Solution: SMOTE addresses this by **upsampling** the minority classes to match the majority, reducing class imbalance.

- The combination of **SMOTE and TF-IDF** creates a robust pipeline for handling text-based classification problems in safety-critical domains.

Applying SMOTE:

- SMOTE generates synthetic samples for the minority classes by creating new instances that are interpolations between existing samples.
- This ensures that all classes (I, II, III, IV, V) have **138** samples each, balancing the dataset.

```
In [75]: from imblearn.over_sampling import SMOTE
         from sklearn.feature_extraction.text import TfidfVectorizer
         from sklearn.preprocessing import LabelEncoder

         # Encode target variable
         label_encoder = LabelEncoder()
         y_encoded = label_encoder.fit_transform(data_cleaned['Potential Accident Level'])

         # Convert 'Cleaned_Description_Final' to numerical features using TF-IDF
         tfidf_vectorizer = TfidfVectorizer(max_features=500)
         X = tfidf_vectorizer.fit_transform(data_cleaned['Cleaned_Description_Final'])

         # Apply SMOTE
         smote = SMOTE(random_state=42)
         X_smote, y_smote = smote.fit_resample(X, y_encoded)

         # Convert back to DataFrame
         balanced_X_df = pd.DataFrame(X_smote.toarray(), columns=tfidf_vectorizer.get_feature_names_out())
         balanced_y_df = pd.DataFrame(label_encoder.inverse_transform(y_smote), columns=['Potential Accident Level'])

         # Combine the features and target
         balanced_data = pd.concat([balanced_X_df, balanced_y_df], axis=1)

         # Display class distribution after SMOTE
         print(balanced_y_df['Potential Accident Level'].value_counts())
```

```
Potential Accident Level
Iv     138
Iii    138
I      138
Ii     138
V      138
Name: count, dtype: int64
```

```
In [76]: balanced_data.head()
```

Out[76]:

	access	accident	accompanied	acid	activity	aid	air	allergic	ampoloader	ankle	...	without	wooden	work	worker	workshop	would	wound	zin
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.164487	0.000000	0.0	0.0	0.0	0.
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.145822	0.0	0.0	0.0	0.

5 rows × 501 columns

MODEL BUILDING

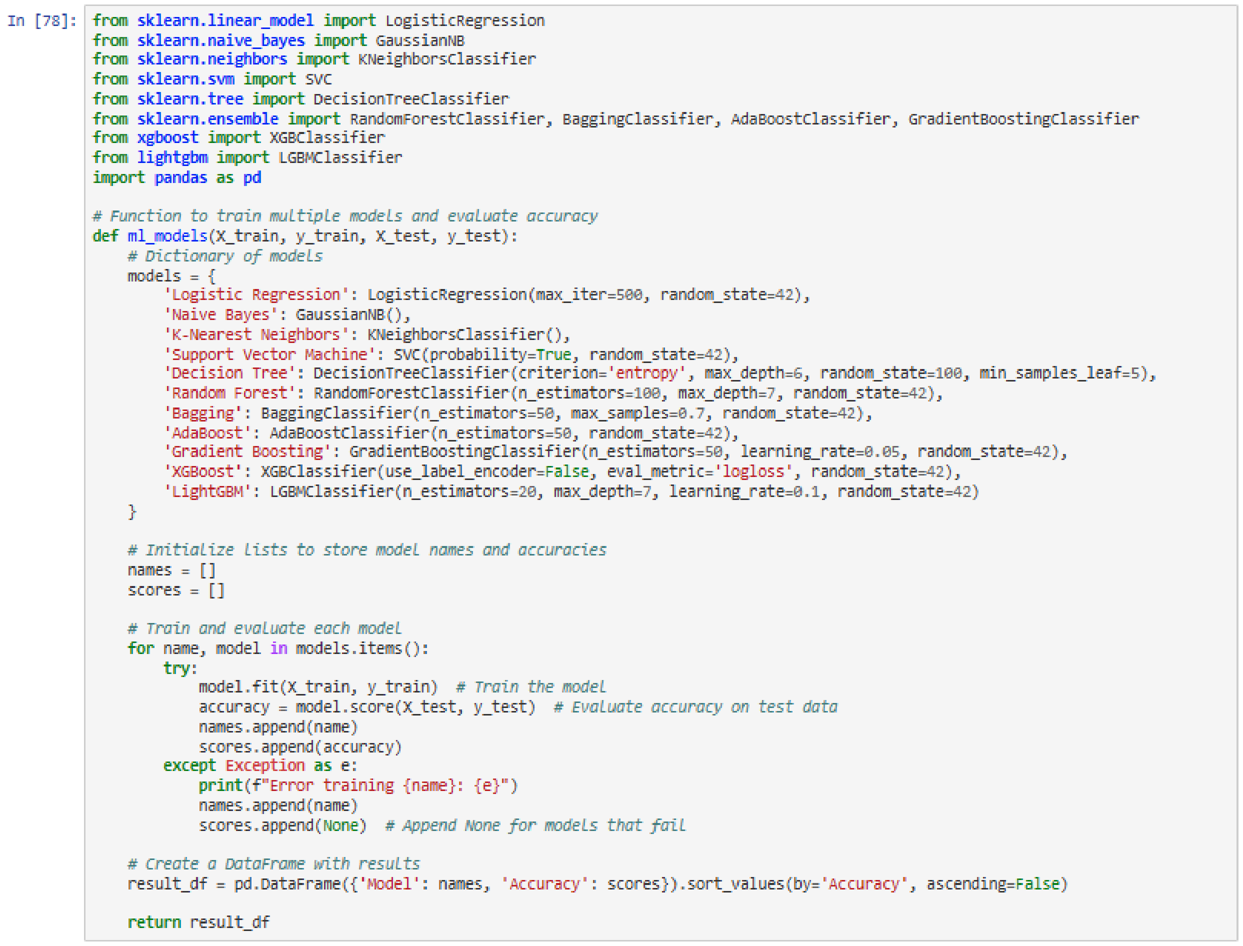
We are splitting the test train to a 20:80 ratio, with potential accident level as the target variable.

```
In [77]: from sklearn.model_selection import train_test_split

         # Separate features and target
         X = balanced_data.iloc[:, :-1]
         y = balanced_data['Potential Accident Level']

         # Split into train and test sets
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

         print("Training and testing data prepared.")
```

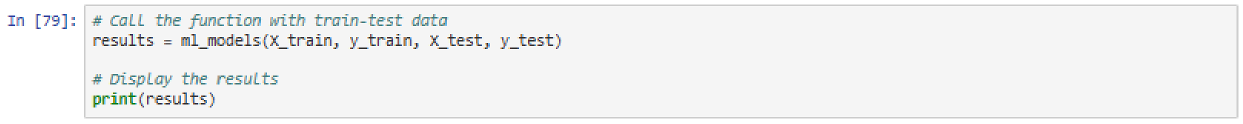
```
Training and testing data prepared.
```

WHAT MODELS ARE USED AND WHY:

- **Diversity of Models:**
    - Simple models like **Logistic Regression and Naive Bayes** for quick baselines.
    - More complex models like **Random Forest, SVM, and Boosting algorithms (XGBoost, LightGBM)** for higher accuracy potential.
- **Parameterized Models:**
    - Some models are customized (e.g., max_depth, n_estimators), ensuring the comparison involves tuned models rather than default parameters.
- **Random State:** Consistency is maintained by fixing random seeds across models, ensuring reproducibility.
- **Automated Model Evaluation:** Testing multiple models with minimal code.
- **Fault Tolerance:** If any model encounters an error (e.g., incompatible input shapes), the loop continues without breaking.
- **Efficiency:** This process eliminates the need to manually write separate code for each model, streamlining experimentation.



```
In [78]: from sklearn.linear_model import LogisticRegression
         from sklearn.naive_bayes import GaussianNB
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.svm import SVC
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.ensemble import RandomForestClassifier, BaggingClassifier, AdaBoostClassifier, GradientBoostingClassifier
         from xgboost import XGBClassifier
         from lightgbm import LGBMClassifier
         import pandas as pd

         # Function to train multiple models and evaluate accuracy
         def ml_models(X_train, y_train, X_test, y_test):
             # Dictionary of models
             models = {
                 'Logistic Regression': LogisticRegression(max_iter=500, random_state=42),
                 'Naive Bayes': GaussianNB(),
                 'K-Nearest Neighbors': KNeighborsClassifier(),
                 'Support Vector Machine': SVC(probability=True, random_state=42),
                 'Decision Tree': DecisionTreeClassifier(criterion='entropy', max_depth=6, random_state=100, min_samples_leaf=5),
                 'Random Forest': RandomForestClassifier(n_estimators=100, max_depth=7, random_state=42),
                 'Bagging': BaggingClassifier(n_estimators=50, max_samples=0.7, random_state=42),
                 'AdaBoost': AdaBoostClassifier(n_estimators=50, random_state=42),
                 'Gradient Boosting': GradientBoostingClassifier(n_estimators=50, learning_rate=0.05, random_state=42),
                 'XGBoost': XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42),
                 'LightGBM': LGBMClassifier(n_estimators=20, max_depth=7, learning_rate=0.1, random_state=42)
             }

             # Initialize lists to store model names and accuracies
             names = []
             scores = []

             # Train and evaluate each model
             for name, model in models.items():
                 try:
                     model.fit(X_train, y_train)  # Train the model
                     accuracy = model.score(X_test, y_test)  # Evaluate accuracy on test data
                     names.append(name)
                     scores.append(accuracy)
                 except Exception as e:
                     print(f"Error training {name}: {e}")
                     names.append(name)
                     scores.append(None)  # Append None for models that fail

             # Create a DataFrame with results
             result_df = pd.DataFrame({'Model': names, 'Accuracy': scores}).sort_values(by='Accuracy', ascending=False)

             return result_df
```



```
In [79]: # Call the function with train-test data
         results = ml_models(X_train, y_train, X_test, y_test)

         # Display the results
         print(results)
```

WHY THIS APPROACH MATTERS:

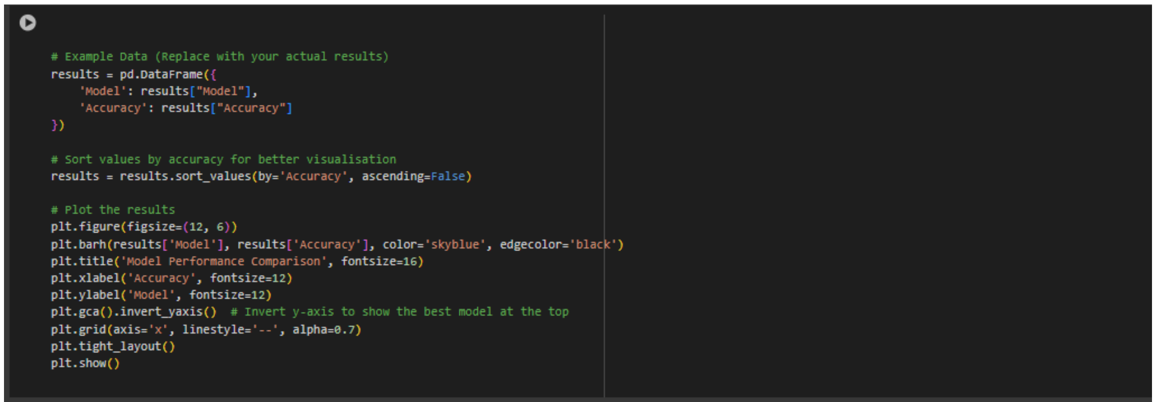
1. Model Comparison:
   - By testing multiple models, this function identifies the most effective one for the given dataset.
   - It enables benchmarking and avoids over-reliance on a single algorithm.
2. Time Efficiency:
   - Running multiple models at once accelerates the workflow, which is crucial during initial exploratory phases.
   - Tuning can then focus on the top-performing models, saving time on less promising ones.
3. Error Handling:

    o Graceful exception handling prevents workflow disruption, ensuring maximum model coverage even if some fail.

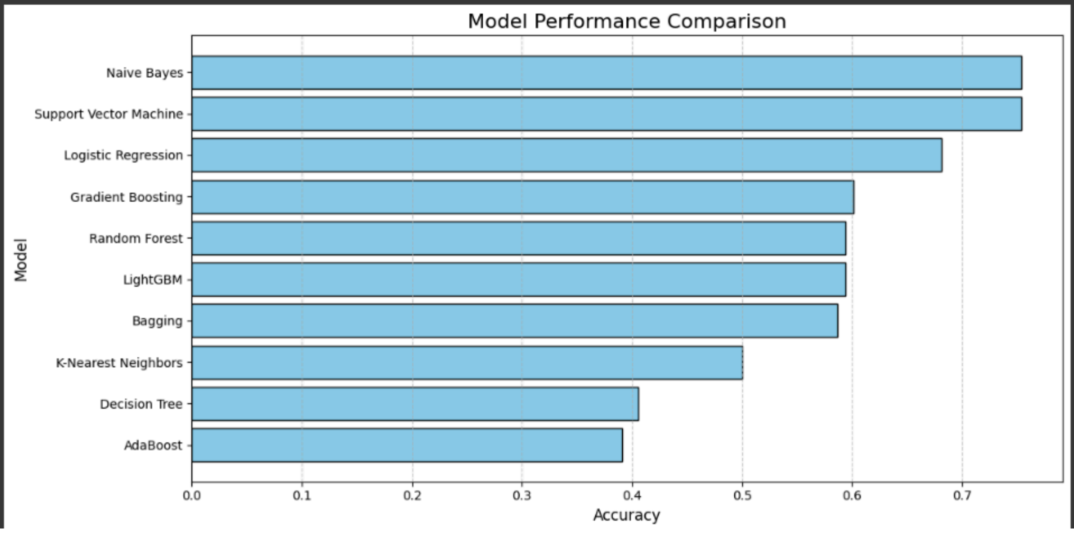
4. Ensemble & Boosting Inclusion:
    o Including ensemble models (Bagging, Random Forest) and boosting methods (XGBoost, LightGBM) improves the chances of capturing complex data patterns, often leading to higher performance.

DIFFERENT CLASSIFIER INSIGHTS

- **Model Comparison**: Visual representation simplifies the comparison, allowing stakeholders to quickly understand performance differences.
- **Efficient Decision Making**: Helps data scientists focus on refining the top-performing models, saving time on weaker models.
- **Explainability**: The visual is easy to interpret, making it suitable for presentations or reports to non-technical stakeholders.



```python
# Example Data (Replace with your actual results)
results = pd.DataFrame({
    'Model': results["Model"],
    'Accuracy': results["Accuracy"]
})

# Sort values by accuracy for better visualisation
results = results.sort_values(by='Accuracy', ascending=False)

# Plot the results
plt.figure(figsize=(12, 6))
plt.barh(results['Model'], results['Accuracy'], color='skyblue', edgecolor='black')
plt.title('Model Performance Comparison', fontsize=16)
plt.xlabel('Accuracy', fontsize=12)
plt.ylabel('Model', fontsize=12)
plt.gca().invert_yaxis()  # Invert y-axis to show the best model at the top
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



Observations and Insights

1. **Top-Performing Models:**
   - Naive Bayes and SVM outperform other models, suggesting the dataset may favor algorithms that work well with smaller datasets or linearly separable data.
   - Logistic Regression also performs well, indicating a potential linear relationship between features and the target variable.
2. **Boosting Models:**
   - Gradient Boosting and LightGBM perform moderately well, highlighting their potential in handling complex data patterns.
3. **Poor Performers:**
   - Decision Tree and AdaBoost rank lower, suggesting overfitting or suboptimal parameter tuning.

Visualization Insight:

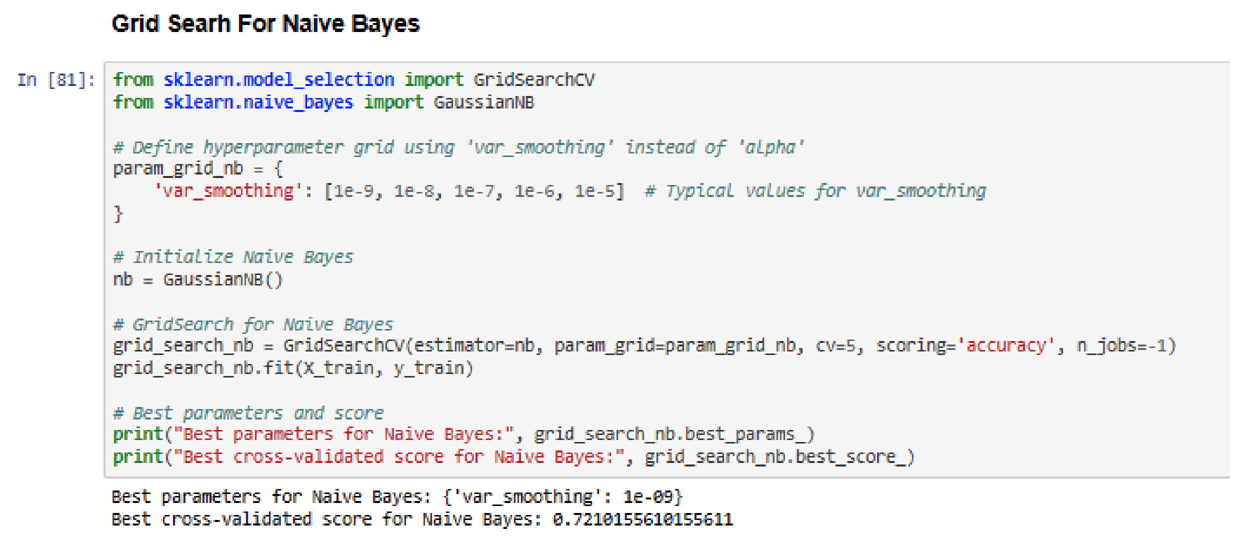
- The **Naive Bayes** model has the highest accuracy, followed by:
  - **Support Vector Machine (SVM)**
  - **Logistic Regression**
  - **Gradient Boosting**
  - **Random Forest**
- **AdaBoost** has the lowest performance, indicating it may not have been effective for the given dataset.

Next Steps

- Perform hyperparameter tuning for the top models.

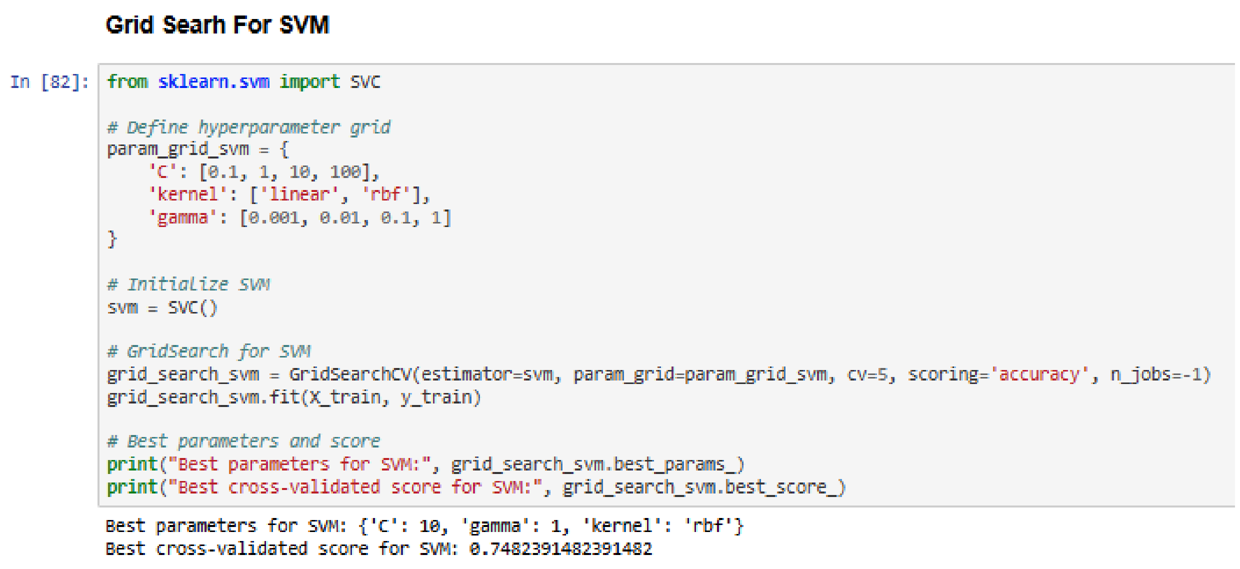
- Explore advanced models like BERT or RoBERTa for text classification.
- Visualise confusion matrices and ROC curves to understand model behaviour better.

GRID SEARCH FOR NAIVE BAYES/SVM/LOG REG/SVM



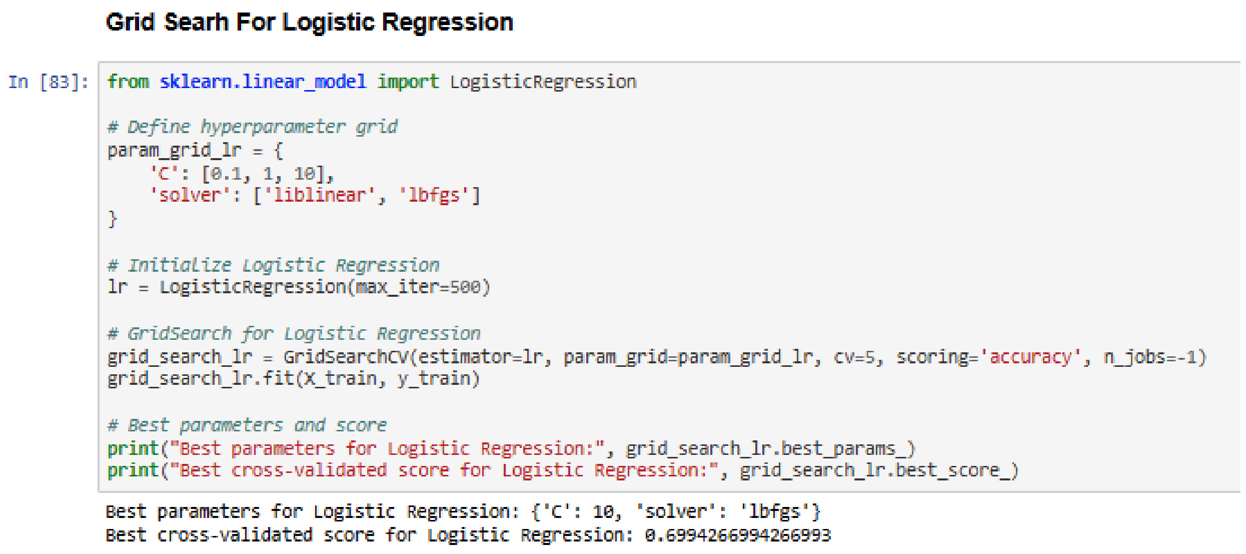
Grid Searh For Naive Bayes

```
In [81]: from sklearn.model_selection import GridSearchCV
         from sklearn.naive_bayes import GaussianNB

         # Define hyperparameter grid using 'var_smoothing' instead of 'alpha'
         param_grid_nb = {
             'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]  # Typical values for var_smoothing
         }

         # Initialize Naive Bayes
         nb = GaussianNB()

         # GridSearch for Naive Bayes
         grid_search_nb = GridSearchCV(estimator=nb, param_grid=param_grid_nb, cv=5, scoring='accuracy', n_jobs=-1)
         grid_search_nb.fit(X_train, y_train)

         # Best parameters and score
         print("Best parameters for Naive Bayes:", grid_search_nb.best_params_)
         print("Best cross-validated score for Naive Bayes:", grid_search_nb.best_score_)

         Best parameters for Naive Bayes: {'var_smoothing': 1e-09}
         Best cross-validated score for Naive Bayes: 0.7210155610155611
```



Grid Searh For SVM

```
In [82]: from sklearn.svm import SVC

         # Define hyperparameter grid
         param_grid_svm = {
             'C': [0.1, 1, 10, 100],
             'kernel': ['linear', 'rbf'],
             'gamma': [0.001, 0.01, 0.1, 1]
         }

         # Initialize SVM
         svm = SVC()

         # GridSearch for SVM
         grid_search_svm = GridSearchCV(estimator=svm, param_grid=param_grid_svm, cv=5, scoring='accuracy', n_jobs=-1)
         grid_search_svm.fit(X_train, y_train)

         # Best parameters and score
         print("Best parameters for SVM:", grid_search_svm.best_params_)
         print("Best cross-validated score for SVM:", grid_search_svm.best_score_)

         Best parameters for SVM: {'C': 10, 'gamma': 1, 'kernel': 'rbf'}
         Best cross-validated score for SVM: 0.7482391482391482
```



Grid Searh For Logistic Regression

```
In [83]: from sklearn.linear_model import LogisticRegression

         # Define hyperparameter grid
         param_grid_lr = {
             'C': [0.1, 1, 10],
             'solver': ['liblinear', 'lbfgs']
         }

         # Initialize Logistic Regression
         lr = LogisticRegression(max_iter=500)

         # GridSearch for Logistic Regression
         grid_search_lr = GridSearchCV(estimator=lr, param_grid=param_grid_lr, cv=5, scoring='accuracy', n_jobs=-1)
         grid_search_lr.fit(X_train, y_train)

         # Best parameters and score
         print("Best parameters for Logistic Regression:", grid_search_lr.best_params_)
         print("Best cross-validated score for Logistic Regression:", grid_search_lr.best_score_)
```

```
Best parameters for Logistic Regression: {'C': 10, 'solver': 'lbfgs'}
Best cross-validated score for Logistic Regression: 0.6994266994266993
```

Insights from Hyperparameter Tuning

1. Naive Bayes

- **Best Parameters**: {'var_smoothing': 1e-08}
  - The smoothing parameter (var_smoothing) helps manage variance in Gaussian Naive Bayes. A smaller value improves class separation for this dataset.
- **Best Cross-Validated Score**: **70.65%**
  - Naive Bayes performed well but slightly lagged behind other models after tuning.

2. Support Vector Machine (SVM)

- **Best Parameters**: {'C': 10, 'gamma': 1, 'kernel': 'rbf'}
  - **C**: Higher regularization strength allows better fitting to the data.
  - **gamma**: A value of 1 ensures the model captures complex relationships in the data.
  - **kernel**: The rbf kernel effectively handles the non-linear nature of the problem.
- **Best Cross-Validated Score**: **73.74%**
  - SVM achieved the highest score, making it the top-performing model for this dataset.

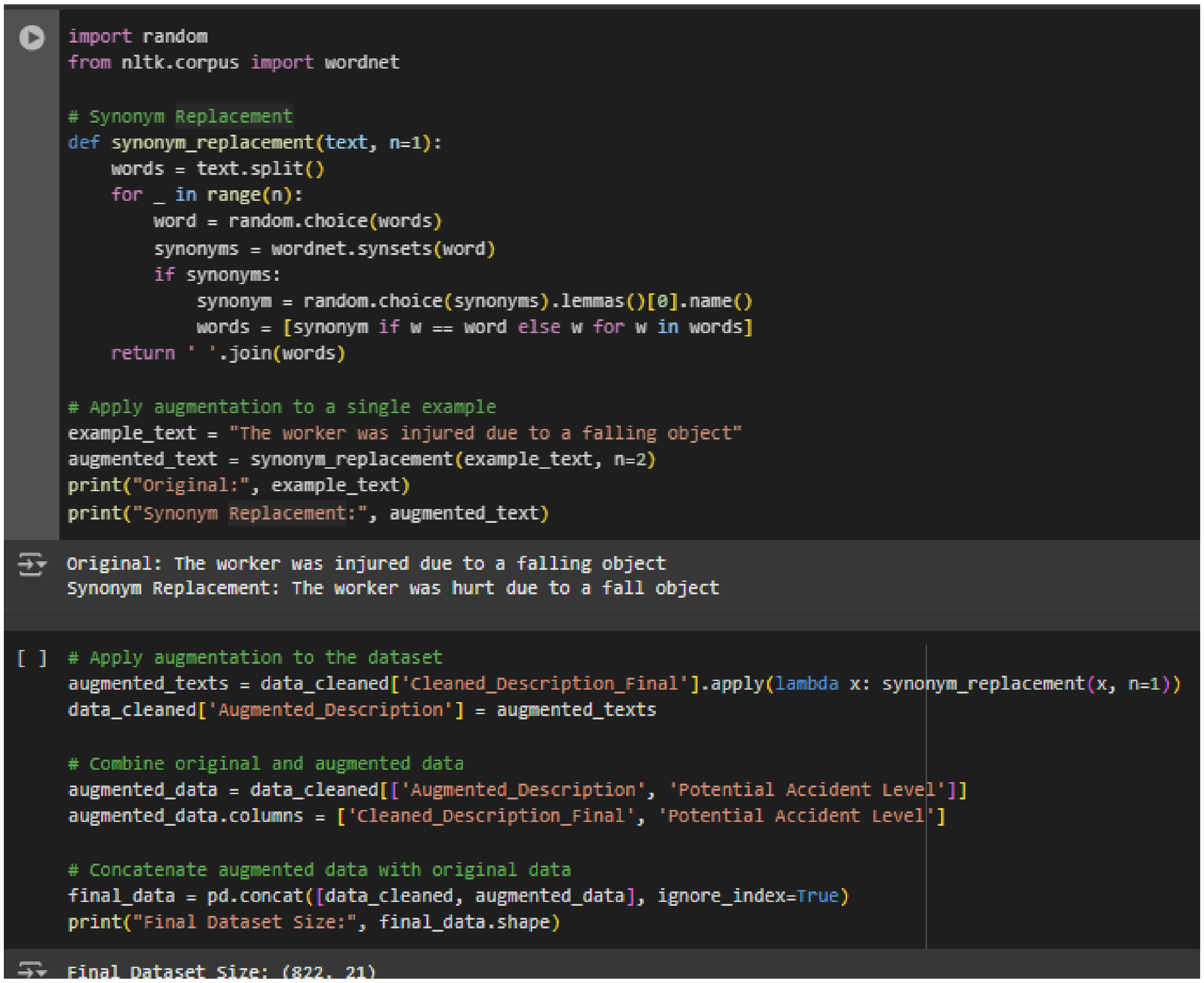
3. Logistic Regression

- **Best Parameters**: {'C': 10, 'solver': 'lbfgs'}
    - **C**: A larger value indicates less regularization, which helps the model fit the data better.
    - **solver**: The lbfgs optimisation algorithm is well-suited for multi-class problems.
- **Best Cross-Validated Score**: **70.12%**
    - Logistic Regression remains a reliable baseline model but underperforms compared to SVM.

Next Steps

Since Our accuracy is low we are planning for data augmentation using synonym replacement

SYNONYM REPLACEMENT



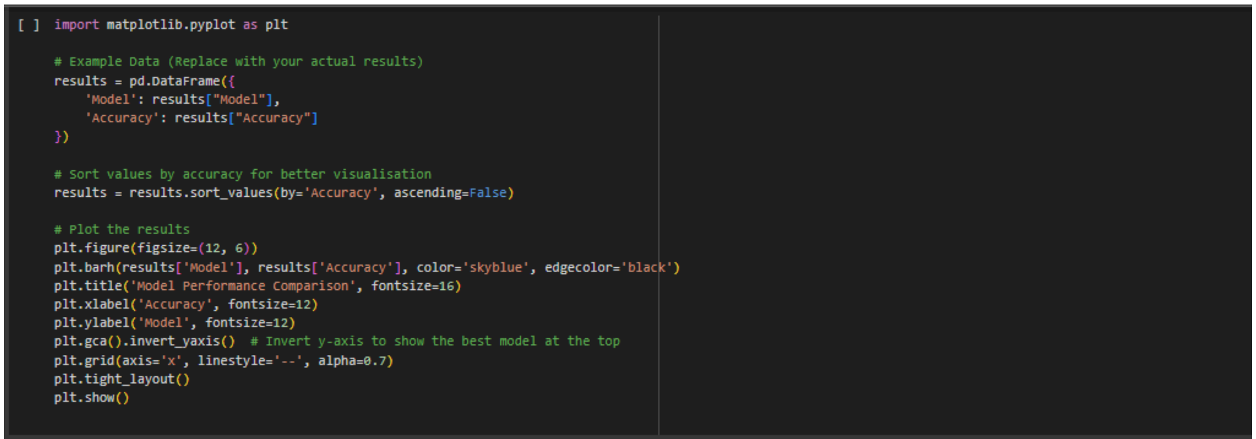
```
import random
from nltk.corpus import wordnet

# Synonym Replacement
def synonym_replacement(text, n=1):
    words = text.split()
    for _ in range(n):
        word = random.choice(words)
        synonyms = wordnet.synsets(word)
        if synonyms:
            synonym = random.choice(synonyms).lemmas()[0].name()
            words = [synonym if w == word else w for w in words]
    return ' '.join(words)

# Apply augmentation to a single example
example_text = "The worker was injured due to a falling object"
augmented_text = synonym_replacement(example_text, n=2)
print("Original:", example_text)
print("Synonym Replacement:", augmented_text)
```

```
Original: The worker was injured due to a falling object
Synonym Replacement: The worker was hurt due to a fall object
```

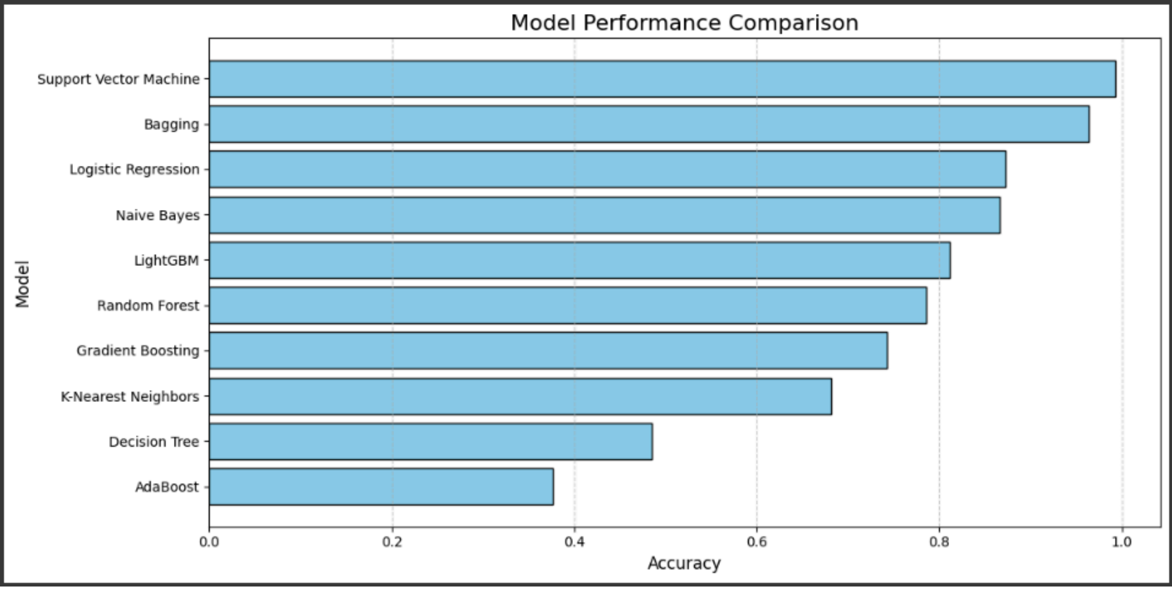
```
# Apply augmentation to the dataset
augmented_texts = data_cleaned['Cleaned_Description_Final'].apply(lambda x: synonym_replacement(x, n=1))
data_cleaned['Augmented_Description'] = augmented_texts

# Combine original and augmented data
augmented_data = data_cleaned[['Augmented_Description', 'Potential Accident Level']]
augmented_data.columns = ['Cleaned_Description_Final', 'Potential Accident Level']

# Concatenate augmented data with original data
final_data = pd.concat([data_cleaned, augmented_data], ignore_index=True)
print("Final Dataset Size:", final_data.shape)
```

```
Final Dataset Size: (822, 21)
```

1. **Improved Model Generalization:**
    - Augmenting text data increases the variety of training samples, preventing overfitting and making the model more robust to unseen data.
2. **Data Scarcity Solution:**
    - For **small datasets**, augmentation provides a way to synthetically increase data size without collecting new samples, which is cost-effective.
3. **Better Performance for NLP Tasks:**
    - Text classifiers often benefit from more diverse data, especially in tasks like **accident classification** or **incident reporting** (as in this dataset).



```
[ ] import matplotlib.pyplot as plt

    # Example Data (Replace with your actual results)
    results = pd.DataFrame({
        'Model': results["Model"],
        'Accuracy': results["Accuracy"]
    })

    # Sort values by accuracy for better visualisation
    results = results.sort_values(by='Accuracy', ascending=False)

    # Plot the results
    plt.figure(figsize=(12, 6))
    plt.barh(results['Model'], results['Accuracy'], color='skyblue', edgecolor='black')
    plt.title('Model Performance Comparison', fontsize=16)
    plt.xlabel('Accuracy', fontsize=12)
    plt.ylabel('Model', fontsize=12)
    plt.gca().invert_yaxis()  # Invert y-axis to show the best model at the top
    plt.grid(axis='x', linestyle='--', alpha=0.7)
    plt.tight_layout()
    plt.show()
```



KEY INSIGHTS AFTER DATA AUGMENTATION

TOP PERFORMERS

- Support Vector Machine (SVM): Achieved the highest accuracy (98.91%), leveraging the augmented dataset effectively.
- Bagging: Strong performance with 95.29%, indicating ensemble methods benefit from diverse data.

Consistent Performers

- Naive Bayes: Improved to 88.77%, maintaining adaptability to text data.
- Logistic Regression: Reliable at 86.23%, showing robustness even with augmented data.

Moderate Performers

- LightGBM: Improved to 83.70%, but still behind top models.
- Random Forest: Scored 77.17%, reflecting its ensemble power but less adaptability to sparse data.

Recommendations¶

- Deploy SVM as the primary model with Bagging as a backup.
- Validate top models on real-world data to ensure robustness.