NLP
Industrial Safety – NLP Based Chatbot

A Capstone Project

Submitted by:

Adittya S
Animesh Johri
Praburam Krishnamurthy
Shree Maya G
Swapnil Lokaksha

In fulfillment of the requirements for the award of
Post Graduate Program in
Artificial Intelligence & Machine Learning

from:

Great Learning
&
TEXAS McCombs
The University of Texas at Austin

Mentor:
Aniket Chabra
November 2024

Contents:

Introduction

In industries worldwide, ensuring employee safety remains a paramount concern, especially in high-risk environments like mining, manufacturing, and heavy metal processing. Despite the implementation of stringent safety protocols and continuous advancements in industrial safety measures, incidents resulting in injuries or fatalities still occur. The persistent occurrence of such accidents underscores the urgent need for industries and companies to better understand and predict the risk factors associated with workplace accidents.

This report focuses on analyzing industry safety data provided by one of the largest industries in Brazil and globally. The primary objective of this report is to predict accident severity using machine learning (ML) models based on historical accident descriptions. The target variables for prediction are "Accident Level" or "Potential Accident Level," representing the actual severity of accidents and their potential risk, respectively. To achieve this, we will leverage Natural Language Processing (NLP) techniques to analyze accident descriptions and extract valuable insights from textual data. This approach allows us to interpret the unstructured descriptions of incidents and use them to predict accident severity.

In this analysis, we will use the accident descriptions as the primary source of information for predicting accident levels. By employing NLP techniques such as tokenization, stopword removal, lemmatization, and TF-IDF (Term Frequency-Inverse Document Frequency) feature extraction, we aim to convert the unstructured text data into meaningful numerical features. These features will then be utilized to train machine learning models for predicting accident levels.

This analysis is crucial not only for mitigating risks and enhancing safety protocols within the organization but also for setting a precedent for industries globally in the pursuit of a safer working environment for all employees.

NOTE: We have **attempted** a **total of 20 models**. Total of 5 models were created for classifying "Potential Accident Level" and 15 models(3 different variations of model for each of the 5 classifiers) for classifying "Accident Level".

Abstract:

(Summary of problem statement(objective), data, findings, approach to EDA, Pre-processing and model building and selection):

Objective:

The objective of this project is to leverage ML techniques to predict accident severity by classifying the target variables "Accident Level" or "Potential Accident Level." A key aim is to build a Natural Language Processing (NLP)-based predictive model that can be integrated into an intelligent chatbot utility, offering real-time severity assessments based on incident descriptions. This proactive approach is designed to enhance workplace safety and risk management.

Data:

The dataset consists of accident records collected from 12 cities in 3 countries, spanning 3 industry types: Mining, Metals, and Others. Each record captures a range of features, including 5 accident levels (I to V) and 6 potential accident levels (I to VI). Additionally, the dataset categorizes employees into three groups: direct employees, third-party employees, and third-party employees working remotely. A "Critical Risk" column categorizes the nature of accidents, while the "Description" column contains detailed accident descriptions.

The dataset is primarily categorical, and the cleaned data columns used for analysis include Date, Country, City, Industry Sector, Accident Level, Potential Accident Level, Gender, Employee Type, Critical Risk, and Description.

Findings and Approach:

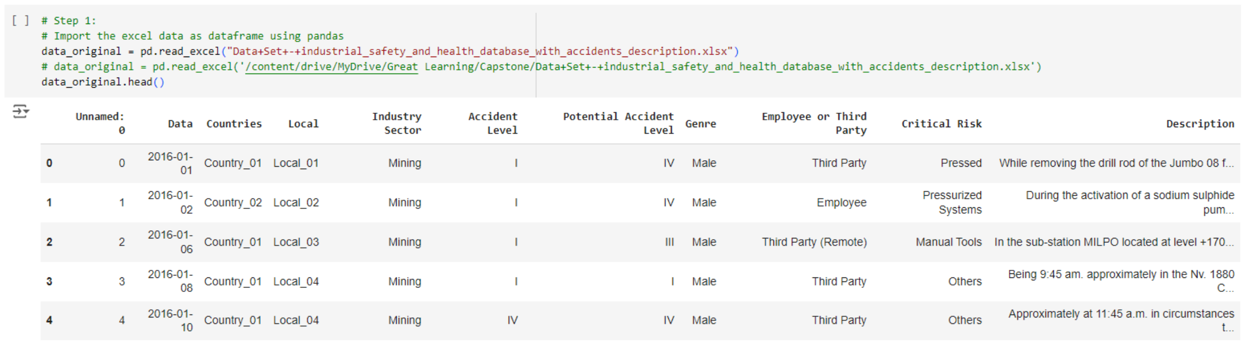
1. **Data Cleaning:** The dataset was provided in Excel format and was imported into a Pandas DataFrame. Initial data cleaning involved renaming columns for clarity, dropping irrelevant and duplicate records, and standardizing the dataset.
2. **Exploratory Data Analysis (EDA):** Univariate analyses were conducted on all categorical columns to gain insights into all features, bivariate, and multivariate analyses were conducted to gain insights into feature relationships and interactions. Key findings were documented.
3. **Feature Identification:** The "Description" feature was identified as the most critical variable for classifying the dependent variables "Accident Level" or "Potential Accident Level."
4. **NLP Pre-processing:** The "Description" feature was pre-processed to generate a new feature, "Preprocessed_Description." Pre-processing steps included lowercasing, removing stopwords, punctuation, special characters, and numbers, along with tokenization and lemmatization.

5. **Data Preparation:** The cleaned and pre-processed data was exported as a CSV file and re-imported as a DataFrame for modeling. The TF-IDF vectorizer was applied to "Preprocessed_Description," extracting the top 1,000 features to be used as independent variables.
6. **Modeling Setup:** The dataset was split into training and testing sets in an 80:20 ratio using stratification on the dependent variable. We considered "Preprocessed_Description" as the independent variable and "Accident Level" or "Potential Accident Level" as the dependent variables.
7. **Model Selection:** Several machine learning algorithms were tested, including Naive Bayes, Logistic Regression, Support Vector Machines (SVM), Random Forest, and Gradient Boosting. Initial base models were trained to predict both "Accident Level" and "Potential Accident Level."
8. **Target Variable Refinement:** After analyzing model performance, "Accident Level" was chosen as the primary target variable due to consistently better results compared to "Potential Accident Level."
9. **Handling Imbalance**: The data exhibited a class imbalance, with approximately 74% of records falling under Accident Level I. To address this, the training data was balanced using the SMOTE technique.
10. **Model Evaluation and Tuning:** We evaluated three types of models for each classifier: base models trained on unbalanced data, base models trained on balanced data, and hyper-parameter-tuned models trained on balanced data. Performance metrics and relevance were considered to select the best-performing model.

The chosen model will be utilized to support real-time accident severity predictions, enabling proactive safety measures and more effective risk management within the organization.

Data Cleaning

Exploring Original Data



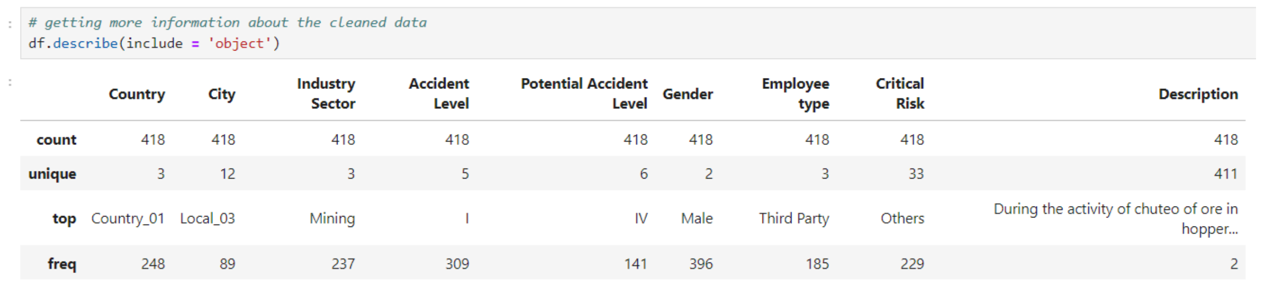
```
[ ] # Step 1:
    # Import the excel data as dataframe using pandas
    data_original = pd.read_excel("Data+Set+-+industrial_safety_and_health_database_with_accidents_description.xlsx")
    # data_original = pd.read_excel('/content/drive/MyDrive/Great Learning/Capstone/Data+Set+-+industrial_safety_and_health_database_with_accidents_description.xlsx')
    data_original.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...

<u>Insights and Data Cleaning approach:</u>

1. "Unnamed" column can be removed as it does not seem to have any significance.
2. "Data" column is of datetime type and contains date, this column should be renamed as "Date"
3. The categorical columns such as "Countries", "Local", "Industry Sector", "Accident Level", "Potential Accident Level", "Genre", "Employee or Third Party", "Critical Risk" and "Description" are all relevant columns.
4. "Genre" column contains information about Gender, hence it can be renamed as "Gender"
5. "Employee or Third Party" column contains information about employee type hence it can be renamed as "Employee Type"
6. "Local" column contains information about anonymized city names, therefore, it can be renamed as "City"
7. "Countries" is a plural term; it can be renamed as a standard singular name like "Country"

Cleaned Data:



```
# getting more information about the cleaned data
df.describe(include = 'object')
```

	Country	City	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Description
count	418	418	418	418	418	418	418	418	418
unique	3	12	3	5	6	2	3	33	411
top	Country_01	Local_03	Mining	I	IV	Male	Third Party	Others	During the activity of chuteo of ore in hopper...
freq	248	89	237	309	141	396	185	229	2

<u>Insights on cleaned data:</u>

- There are 3 countries in the dataset and "Country_01" has more than 50% of the total industrial incidents.
- There are 12 cities in the dataset and "Local_03" which belongs to "Country_01" has the greatest number of industrial incidents compared to other cities.
- There are 3 categories of Industrial sector and "Mining" has the largest and more than 50% of the total number of incidents.
- There are 5 unique accident levels and there are around 75% of accidents at accident level 1, which means the majority of the incidents are not severe.
- There are 6 unique potential accident levels and the majority of the accidents have a potential accident level of 4.
- Around 95% of the workers are males in all the mentioned accidents.
- There are 3 types of employees. The majority of employees i.e. just under 50% of them are third-party employees.
- Majority of the risk category under the "Critical Risk" column is mentioned as "Others".
- There are 411 unique descriptions out of a total 418 accident descriptions which means some accident descriptions exactly match each other.
- To conclude, there is a high chance of the incidents occurring for the following profile: country as country_01, Local_03 city, mining industrial sector, third-party employee, male worker. We can investigate further on this.

EDA Summary:

Country

Country_01 has the highest incidents (59.33%), followed by Country_02 (30.86%) and Country_03 (9.81%).

City

Local_03 accounts for the largest share of incidents (21.29%), while other cities like Local_05 and Local_01 also show notable counts. Local_12, Local_09 and Local_11 cities have the least number of accidents.

Industry Sector

Mining dominates with 56.7% of incidents, while Metals accounts for 32.06% and Others for 11.24%.

Accident Level Most incidents have accident level I with 73.92% of incidents, followed by level II,III, IV and V. As the accidents levels are increasing the number on accidents are decreasing.

Potential Accident Level

Most incidents are in the highest potential accident level IV with 33.73% of incidents, followed by level III at 25.36%.

Gender

Males are overwhelmingly involved in incidents, representing 94.74% of cases.

Employee Type

Third parties represent 44.26% of incidents, closely followed by employees (42.58%), and third-party remote workers (13.16%).

Critical Risk

The 'Others' category dominates, followed by smaller contributions from risks like "Pressed," "Manual Tools," and "Chemical Substances."

Incidents per Day/Week/Month/Year/Season

The data spans from January 2016 to July 2017, covering about 1.5 years. Incidents fluctuate over time, peaking at certain points in months and years, with steady trends over weeks. The number of incidents per month has a slight declining trend as the year progresses from February to December. More incidents occurred in cooler weather compared to warmer weather.

Accident Level by Country

Country_01 shows the highest accident level I incidents, followed by lower counts in other levels for all countries.

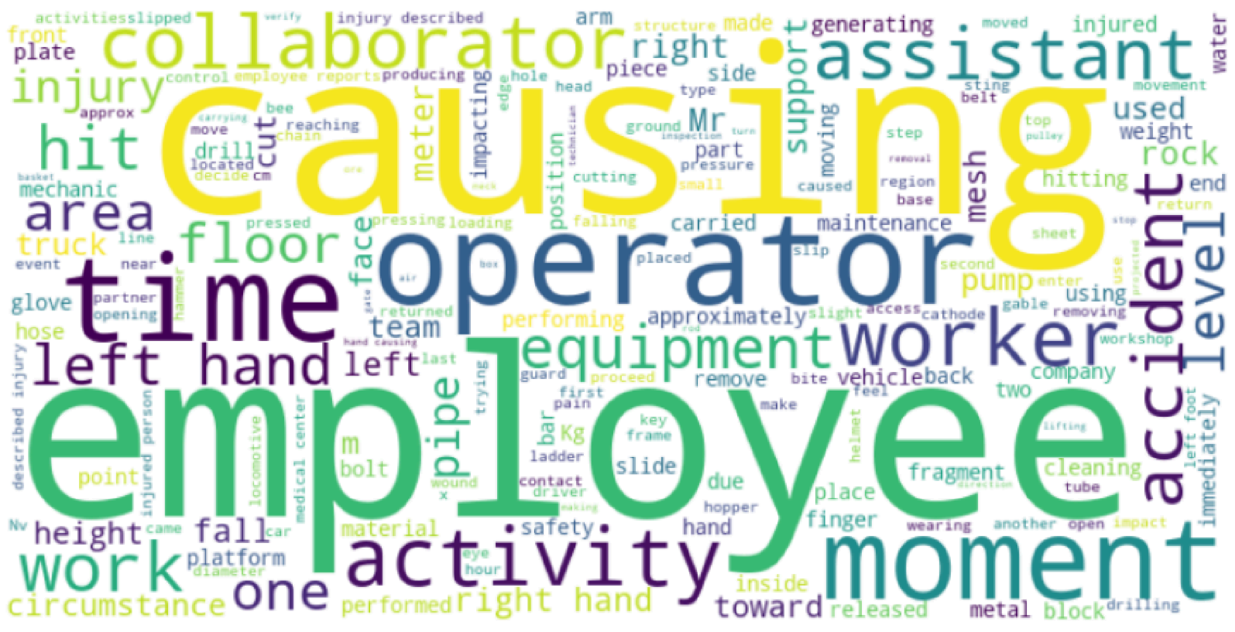
Accident Level by Critical Risk

Level I incidents dominate across all critical risks, with a few cases in Levels II-V across various risk types.

Accident Level and Potential Accident Level Accident Level I has the highest occurrence, especially with potential severity levels up to IV. This indicates that while many accidents are classified as minor (I), they have the potential to escalate into more serious incidents if not properly managed

Industry Sectors, Countries and accident levels: The Mining industry has the highest number of accidents in country_01. The Metals industry has the highest number of accidents in country_02. Majority of Others category sector accidents occurred in Country_03. In all the industry sectors across all the countries most of the accidents are categorized as 1.

Word Cloud: Words like "hand" (177), "left" (155), "right" (154), and "finger" (76) indicate that injuries frequently involve hands and fingers. This suggests that many accidents occur during manual tasks or handling of equipment, leading to injuries in these body parts.



NLP Pre-processing

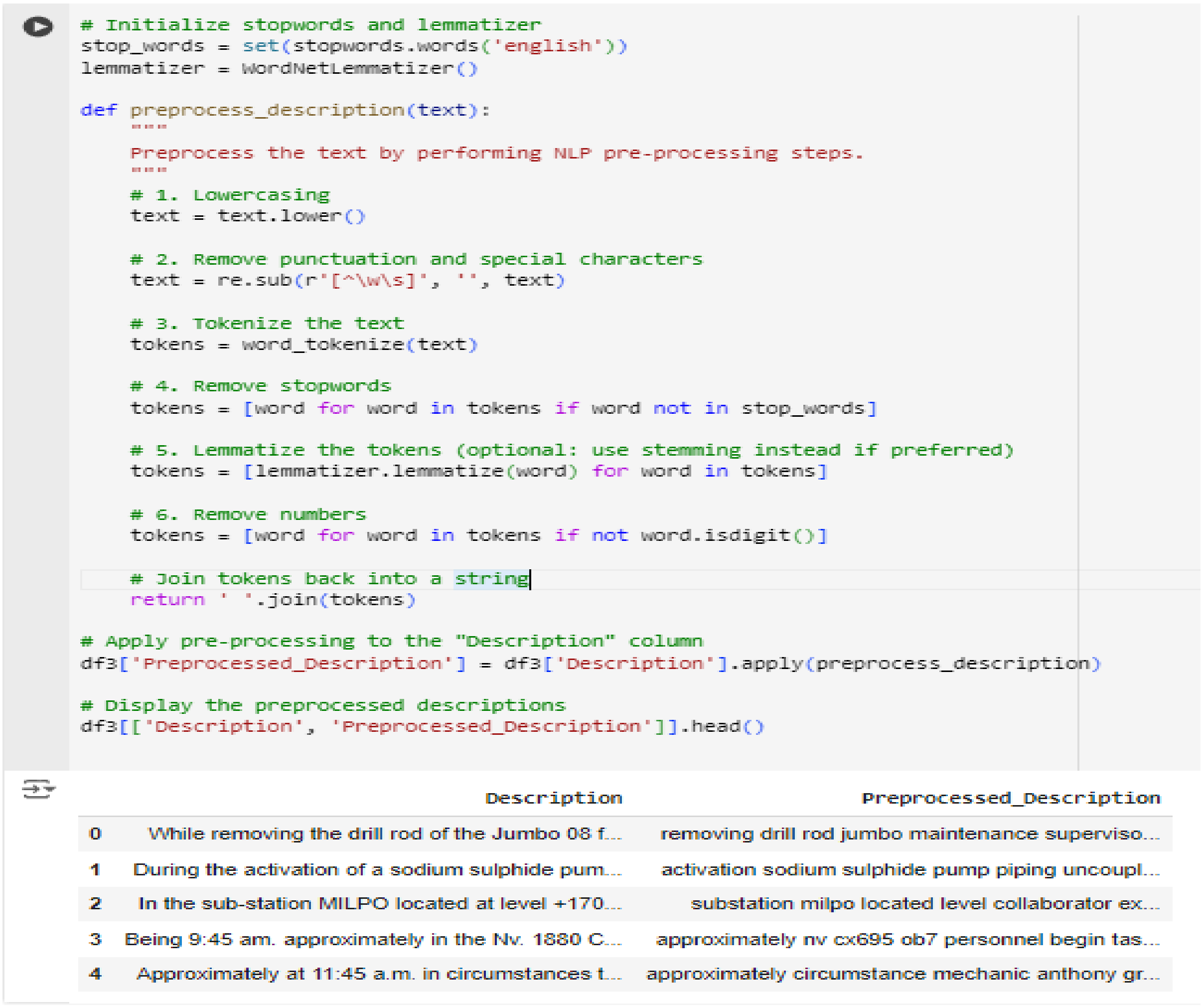
Based on the “Description” column the below pre-processing steps were performed:

- Lowercasing: There are capital letters, we can convert all the text to lowercase.
- Punctuation Removal: All the unnecessary punctuations can be removed.
- Stopwords Removal: There are stop words like "the", "of", "to" etc. We can remove them.
- Removing Special Characters: We can clean up all non-alphanumeric characters (except for words).
- Tokenization: Break sentences into individual words (tokens).
- Lemmatization: We can reduce a word to its root by using lemmatization. We would prefer to do lemmatization over stemming to get the actual word and not just the truncated part of the word. Lemmatization also gives more context to chatbot conversations as it recognizes words based on their exact and contextual meaning.
- Removing Numbers: Remove any numeric values if not relevant.

Importing relevant libraries for NLP pre-processing:

```python
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

# Download necessary resources
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

Performed the NLP preprocessing as below:



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

def preprocess_description(text):
    """
    Preprocess the text by performing NLP pre-processing steps.
    """
    # 1. Lowercasing
    text = text.lower()

    # 2. Remove punctuation and special characters
    text = re.sub(r'[^\w\s]', '', text)

    # 3. Tokenize the text
    tokens = word_tokenize(text)

    # 4. Remove stopwords
    tokens = [word for word in tokens if word not in stop_words]

    # 5. Lemmatize the tokens (optional: use stemming instead if preferred)
    tokens = [lemmatizer.lemmatize(word) for word in tokens]

    # 6. Remove numbers
    tokens = [word for word in tokens if not word.isdigit()]

    # Join tokens back into a string
    return ' '.join(tokens)

# Apply pre-processing to the "Description" column
df3['Preprocessed_Description'] = df3['Description'].apply(preprocess_description)

# Display the preprocessed descriptions
df3[['Description', 'Preprocessed_Description']].head()
```

```
                                     Description                           Preprocessed_Description
0    While removing the drill rod of the Jumbo 08 f...   removing drill rod jumbo maintenance superviso...
1   During the activation of a sodium sulphide pum...   activation sodium sulphide pump piping uncoupl...
2    In the sub-station MILPO located at level +170...      substation milpo located level collaborator ex...
3  Being 9:45 am. approximately in the Nv. 1880 C...   approximately nv cx695 ob7 personnel begin tas...
4    Approximately at 11:45 a.m. in circumstances t...  approximately circumstance mechanic anthony gr...
```

Design train and test basic machine learning classifiers

Deciding Suitable Models

We will use the below models to classify accident levels based on incident description:

- Naive Bayes
- Logistic Regression
- SVM
- Random Forest
- Gradient Boosting

Note: We have not used other classification models like KNN, Adaboost or Decision tree because of high dimensional data and sparsity in the data created after tfidf vectorization.

For multi-class classification of accident severity based on accident descriptions using TF-IDF text features, it's wise to start with models like Naive Bayes, Logistic Regression, SVM, Random Forest, and GradientBoostingClassifier. Here's why each model is a good choice:

1. **Naive Bayes**: We can choose it because it is computationally efficient, simple, and well-suited for high-dimensional and sparse data like TF-IDF features. It performs well in text classification tasks due to its probabilistic approach.

   Use Case Fit: Handles multi-class problems effectively with robust performance even when the independence assumption of features isn't strictly met.

2. **Logistic Regression**: We can choose it because It's a straightforward and interpretable model that works well with TF-IDF features. Regularization techniques can help prevent overfitting, making it reliable.

   Use Case Fit: Logistic regression is linear, handles multi-class scenarios with ease (via One-vs-Rest), and is good for understanding feature importance.

3. **Support Vector Machine (SVM)**: We can choose it because SVMs are powerful classifiers that can create complex decision boundaries and work well for text data. They handle high-dimensional spaces effectively and can be tuned with different kernels.

   Use Case Fit: Ideal for handling non-linear relationships and distinguishing subtle differences in text data, even with sparse features.

4. **Random Forest:** We can choose it because it's an ensemble method that reduces overfitting by averaging multiple decision trees. It handles complex feature interactions and works well even if some data is missing or noisy.

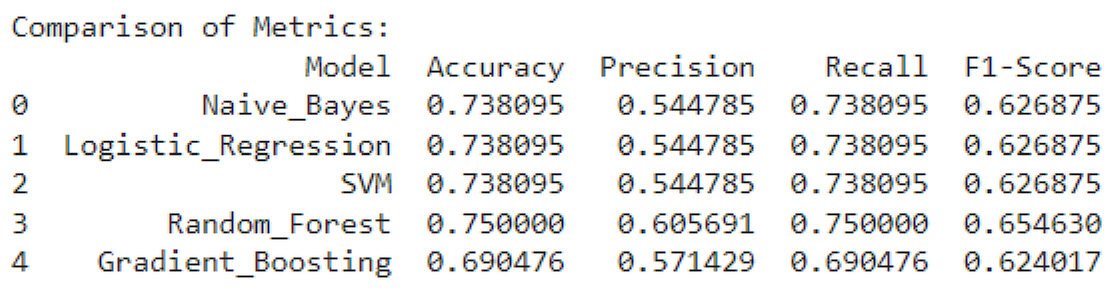
   Use Case Fit: Good for capturing non-linearities and interactions among features, especially when the feature set size grows after creating TF-IDF vectors.

5. **GradientBoostingClassifier:** We can choose it because Gradient boosting is an ensemble method known for its high predictive power. It iteratively improves predictions by combining weak learners, reducing bias, and handling non-linear patterns.

   Use Case Fit: Great for fine-tuning and achieving high accuracy on imbalanced and complex datasets when you have enough data to train it efficiently.

- **Summary**: These models offer a mix of simplicity, interpretability, and power. Starting with Naive Bayes will give us a strong baseline due to its efficiency, while models like Logistic Regression, SVM, Random Forest, and GradientBoosting allow us to explore more complex relationships and interactions in the text data. These models provide a range of trade-offs between computation cost, interpretability, and predictive power, making them a solid starting point for classifying accident severity levels based on text descriptions.

Classification of "Accident Level" target variable using Base Model:

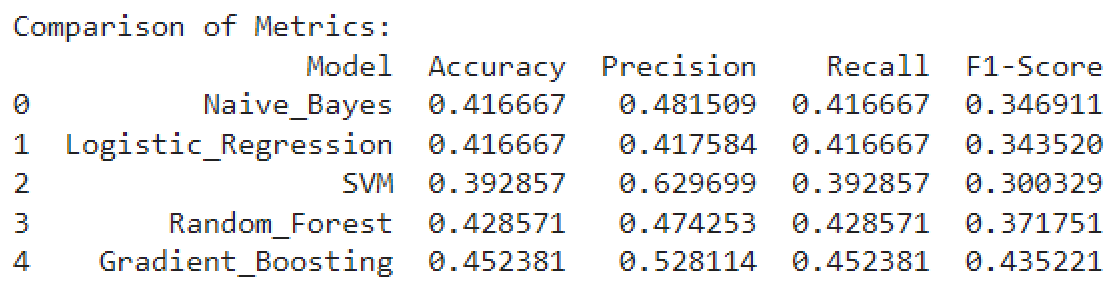


```
Comparison of Metrics:
                 Model  Accuracy  Precision    Recall  F1-Score
0          Naive_Bayes  0.738095   0.544785  0.738095  0.626875
1  Logistic_Regression  0.738095   0.544785  0.738095  0.626875
2                  SVM  0.738095   0.544785  0.738095  0.626875
3        Random_Forest  0.750000   0.605691  0.750000  0.654630
4    Gradient_Boosting  0.690476   0.571429  0.690476  0.624017
```

Insight:

- All the above classification models gave almost the similar results, we can try to further improve the model performance by balancing the data and tuning the model.

Classification of "Potential Accident Level" target variable using Base Model:



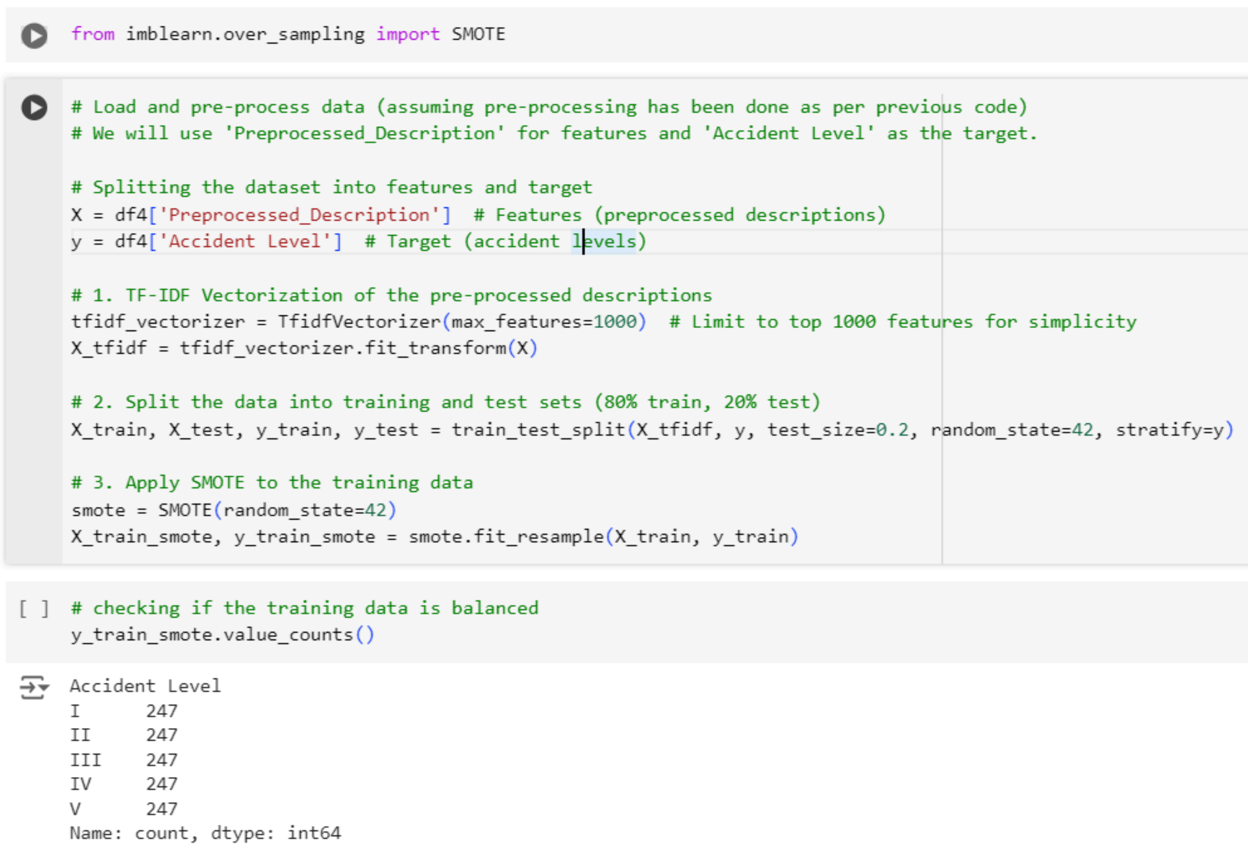
```
Comparison of Metrics:
                 Model  Accuracy  Precision    Recall  F1-Score
0          Naive_Bayes  0.416667   0.481509  0.416667  0.346911
1  Logistic_Regression  0.416667   0.417584  0.416667  0.343520
2                  SVM  0.392857   0.629699  0.392857  0.300329
3        Random_Forest  0.428571   0.474253  0.428571  0.371751
4    Gradient_Boosting  0.452381   0.528114  0.452381  0.435221
```

Insights:

- All the above classification models gave poor results when "Potential Accident Level" when it is used as the target variable instead of "Accident Level", we can finalize "Accident Level" as the target variable for doing the classification.
- Let's improve the imbalanced data and run all the base and tuned classification models on the balanced data.

Model performance Improvement by Balancing data using SMOTE and Hyper-parameter Tuning:

- We will use the SMOTE technique to balance the training data.
- The top 1000 most relevant features based on term frequency or **TF-IDF** score will be selected by using max_features after creating the tfidf vectorization as shown below:



```
from imblearn.over_sampling import SMOTE
```

```
# Load and pre-process data (assuming pre-processing has been done as per previous code)
# We will use 'Preprocessed_Description' for features and 'Accident Level' as the target.

# Splitting the dataset into features and target
X = df4['Preprocessed_Description']  # Features (preprocessed descriptions)
y = df4['Accident Level']  # Target (accident levels)

# 1. TF-IDF Vectorization of the pre-processed descriptions
tfidf_vectorizer = TfidfVectorizer(max_features=1000)  # Limit to top 1000 features for simplicity
X_tfidf = tfidf_vectorizer.fit_transform(X)

# 2. Split the data into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2, random_state=42, stratify=y)

# 3. Apply SMOTE to the training data
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

```
# checking if the training data is balanced
y_train_smote.value_counts()
```

```
Accident Level
I      247
II     247
III    247
IV     247
V      247
Name: count, dtype: int64
```

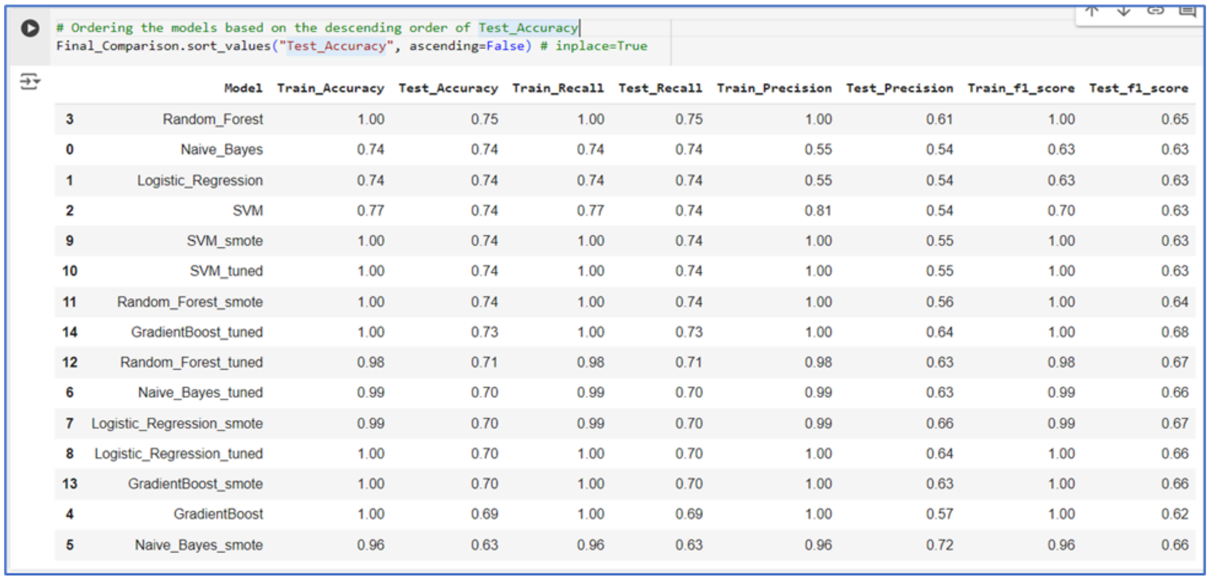
- It is observed that the training data is balanced after using the SMOTE technique.
- After running the base model on balanced data, attempts are made to improve the Model performance improvement using hyper-parameter tuning for each classifier.

We have built 3 kinds of models for each classifier. Suppose name of the model is "Model Name" then the below naming convention is followed:

1. **"Model_Name"**: This is the base model of the classifier which is trained on imbalanced, cleaned and pre-processed training data.
2. **"Model_Name_smote"**: This is the base model of the classifier which is trained on balanced training data obtained after SMOTE.
3. **"Model_Name_tuned"**: This is the Hyper-parameter tuned model of the classifier which is trained on balanced training data obtained after SMOTE.

Model Comparison

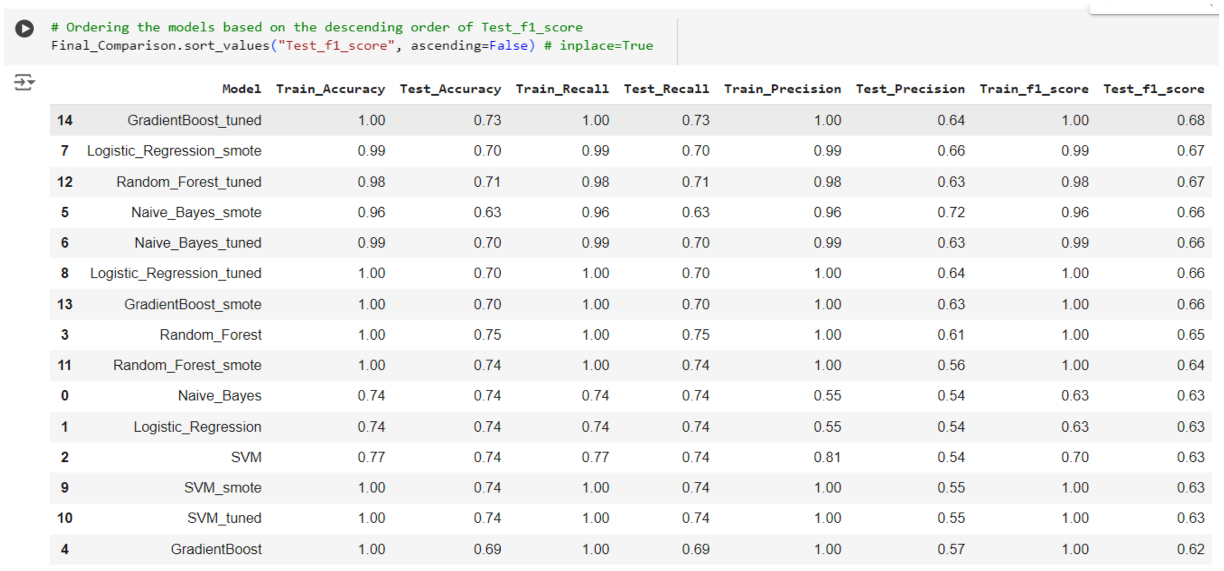
Ordering the models based on the descending order of Test_Accuracy:



```
# Ordering the models based on the descending order of Test_Accuracy
Final_Comparison.sort_values("Test_Accuracy", ascending=False) # inplace=True
```

	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision	Train_f1_score	Test_f1_score
3	Random_Forest	1.00	0.75	1.00	0.75	1.00	0.61	1.00	0.65
0	Naive_Bayes	0.74	0.74	0.74	0.74	0.55	0.54	0.63	0.63
1	Logistic_Regression	0.74	0.74	0.74	0.74	0.55	0.54	0.63	0.63
2	SVM	0.77	0.74	0.77	0.74	0.81	0.54	0.70	0.63
9	SVM_smote	1.00	0.74	1.00	0.74	1.00	0.55	1.00	0.63
10	SVM_tuned	1.00	0.74	1.00	0.74	1.00	0.55	1.00	0.63
11	Random_Forest_smote	1.00	0.74	1.00	0.74	1.00	0.56	1.00	0.64
14	GradientBoost_tuned	1.00	0.73	1.00	0.73	1.00	0.64	1.00	0.68
12	Random_Forest_tuned	0.98	0.71	0.98	0.71	0.98	0.63	0.98	0.67
6	Naive_Bayes_tuned	0.99	0.70	0.99	0.70	0.99	0.63	0.99	0.66
7	Logistic_Regression_smote	0.99	0.70	0.99	0.70	0.99	0.66	0.99	0.67
8	Logistic_Regression_tuned	1.00	0.70	1.00	0.70	1.00	0.64	1.00	0.66
13	GradientBoost_smote	1.00	0.70	1.00	0.70	1.00	0.63	1.00	0.66
4	GradientBoost	1.00	0.69	1.00	0.69	1.00	0.57	1.00	0.62
5	Naive_Bayes_smote	0.96	0.63	0.96	0.63	0.96	0.72	0.96	0.66

Ordering the models based on the descending order of Test_f1_score:



```
# Ordering the models based on the descending order of Test_f1_score
Final_Comparison.sort_values("Test_f1_score", ascending=False) # inplace=True
```

	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision	Train_f1_score	Test_f1_score
14	GradientBoost_tuned	1.00	0.73	1.00	0.73	1.00	0.64	1.00	0.68
7	Logistic_Regression_smote	0.99	0.70	0.99	0.70	0.99	0.66	0.99	0.67
12	Random_Forest_tuned	0.98	0.71	0.98	0.71	0.98	0.63	0.98	0.67
5	Naive_Bayes_smote	0.96	0.63	0.96	0.63	0.96	0.72	0.96	0.66
6	Naive_Bayes_tuned	0.99	0.70	0.99	0.70	0.99	0.63	0.99	0.66
8	Logistic_Regression_tuned	1.00	0.70	1.00	0.70	1.00	0.64	1.00	0.66
13	GradientBoost_smote	1.00	0.70	1.00	0.70	1.00	0.63	1.00	0.66
3	Random_Forest	1.00	0.75	1.00	0.75	1.00	0.61	1.00	0.65
11	Random_Forest_smote	1.00	0.74	1.00	0.74	1.00	0.56	1.00	0.64
0	Naive_Bayes	0.74	0.74	0.74	0.74	0.55	0.54	0.63	0.63
1	Logistic_Regression	0.74	0.74	0.74	0.74	0.55	0.54	0.63	0.63
2	SVM	0.77	0.74	0.77	0.74	0.81	0.54	0.70	0.63
9	SVM_smote	1.00	0.74	1.00	0.74	1.00	0.55	1.00	0.63
10	SVM_tuned	1.00	0.74	1.00	0.74	1.00	0.55	1.00	0.63
4	GradientBoost	1.00	0.69	1.00	0.69	1.00	0.57	1.00	0.62

Insights and Conclusion:

We have **attempted** a **total of 20 models**. Total of 5 models were created for classifying "Potential Accident Level" and 15 models(3 different variations of model for each of the 5 classifiers) for classifying "Accident Level". Based on the analysis below are the insights and conclusion:

1. **Model Performance Evaluation**:

- Random forest base Model trained on unbalanced data yielded the highest accuracy, but showed overfitting, with a difference of more than 25% between test and train metrics

- Gradient Boosting model, fine tuned for hyperparameters on balanced data, achieved the highest F1 scores but also overfitted significantly. Other models that were hyper-parameter tuned and trained on SMOTE data which had a good f-1 score also displayed a significant amount of over-fitting across all metrics.

- Naive Bayes, Logistic Regression and SVM base models trained on unbalanced data gave the same test metrics (accuracy, precision, recall, F1-score) without overfitting.

- Further analyzing, Logistics Regression and SVM are powerful classifiers, their computational complexity, tendency to overfit high-dimensional data, challenges with sparsity (logistics regression may struggle with very rare words that occur infrequently in the data set, impacting its ability to capture the full spectrum of severity descriptions accurately and SVM for finding optimal hyperplanes in very high dimensional features, leading to longer training time) , and need for specialized tuning make them less practical

2. **Best Model - Naive Bayes**

- Adaptability to High Dimensionality sparse data - The model is highly effective for managing high dimensional data and sparse data which is in text classification tasks. As we have transformed the text data into a matrix format using TF-IDF, each word becomes a separate feature. This results in a high dimensional dataset with many features. Also, this applies to sparse data as most entries in the data set tend to be Zeros. In this model, the accident description matrix contains many zeroes where words are absent. Naive Bayes works well with sparse data as it focuses only on non-zero values without needing heavy computations on all zeros

- Scalability & Efficiency - As Naive Bayes focus on non-zero values (words used in each accident description), it can handle large datasets of accident descriptions without excessive computational demands. This is ideal for large text based datasets

- Interpretability- Naive Bayes calculates the probability of each severity level for a given accident description. Examining each word in the description and computing how likely it is that this word would appear in descriptions for each severity level . This is very helpful for stakeholders as they can see which terms are most influential in determining the model's predictions, providing insight into why certain accident descriptions are classified at specific severity levels

- Independence Assumption - While Naive Bayes assumes feature independence, a simplified assumption, it performs surprisingly well for text data as it multiplies the probabilities of individual words occurring in each level, effectively leveraging the work patterns associated with different severity levels.
- Multi-Class Classification: Naive Bayes can handle multi-class classification problems effectively, which is critical in this case where the dependent variable (accident severity) has multiple levels

Improvements:

Improvements Attempted

- **Data Manipulation:** We applied a range of NLP preprocessing techniques such as removing stop words, special characters, punctuation, and numbers, while also using lemmatization and tokenization to standardize text structure.
- **Feature Selection with TF-IDF:** By setting max_features=1000, we retained only the most significant terms, reducing dimensionality and avoiding overfitting. This limit allowed the model to focus on essential features without capturing noise.
- **Balancing Techniques:** Using SMOTE (Synthetic Minority Over-sampling Technique), we aimed to balance the data. However, SMOTE introduced artificial patterns that led to overfitting, especially on high-dimensional TF-IDF features, where the model learned specifics of synthetic samples rather than general class patterns.
- **Hyperparameter Tuning:** Extensive tuning of models on balanced data showed no improvement over the base models trained on imbalanced data, with the tuned models often overfitting. **Ultimately, the Naive Bayes base model trained on imbalanced data offered the most reliable performance.**

Conclusion on ML Classifier model: The Naive Bayes base model on imbalanced data was selected for its balance between simplicity, interpretability, and efficiency in handling sparse, high-dimensional text data. This model provides a practical and scalable solution for accident severity classification, enabling safety professionals to identify and address potential risks effectively based on accident descriptions.

Scope of Improvement

There is scope for further improvement by using **Deep Learning Models** (e.g., LSTM, BERT, RNN): Deep learning models, particularly LSTM (Long Short-Term Memory networks) and transformer-based models like BERT, can capture complex patterns and dependencies in text data that traditional models may not. They can outperform traditional models when sufficient training data is available and preprocessing is handled well.

2nd Milestone:

Overview:

The second part of the report involves the use of different approaches using the BERT embeddings or Transformer's fine tuning to improve the performance compared to the 1st milestone.

First approach is to use **BERT embeddings with Traditional classifiers**. In this approach, BERT embeddings are generated for the input text (like accident descriptions) and used as features in a traditional classifier, such as logistic regression, SVM, random forest, or gradient boosting. We have tried this approach with both raw and NLP pre-processed data obtained from milestone-01. We moved forward for further improvement using the raw data which gave the best results in this approach.

Second approach involves the **fine-tuning of transformers models for sequence classification**. Here, a pre-trained model (like BERT, RoBERTa, DistilBERT, XLNet, or ALBERT) is fine-tuned specifically for accident level classification by adding a classification layer and optimizing the model end-to-end on the accident description data. Different attempts were made to achieve improved performance like increasing the epochs, choosing the **metric_for_best_model** as f1, balancing the data and finally modifying the data to reduce the number of classes and run the models on balanced data.

First Approach: BERT Embeddings + Traditional Classifiers

In this approach, BERT embeddings are generated for the input text (like accident descriptions) and used as features in a traditional classifier, such as logistic regression, SVM, random forest, or gradient boosting.

Steps:

- Use a pre-trained BERT model to extract sentence embeddings for each accident description.
- Feed these embeddings as features to a classifier for multi-class accident level prediction.

Advantages:

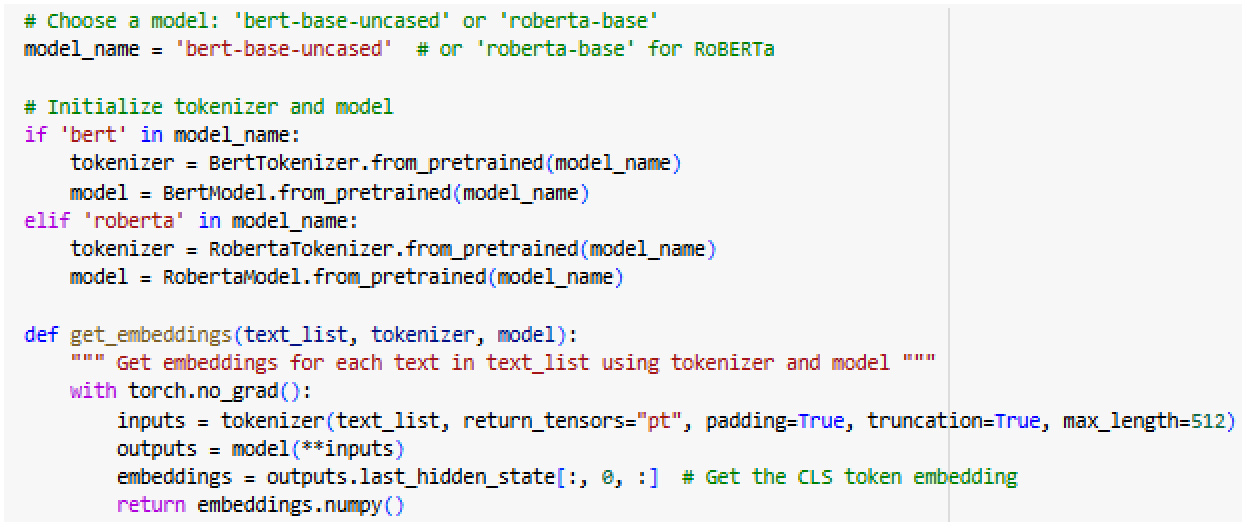
- Interpretability: Traditional classifiers, especially linear ones like logistic regression, can offer more interpretable results, as we can analyze feature importance and weights.
- Lower Computational Requirements: Extracting embeddings and then training a classifier is generally faster and less resource-intensive than fine-tuning a transformer model on the entire dataset.
- Efficiency on Small Datasets: This approach is often sufficient for smaller datasets, where training a full transformer model might lead to overfitting or be computationally prohibitive.
- Versatility: We can experiment with different classifiers quickly (like logistic regression, SVM, etc.) to find which works best with the embeddings.

Limitations:

- Loss of Contextual Information: The embeddings capture a general representation of the text but may lose some finer details compared to sequence classification, where the model is optimized end-to-end for the classification task.

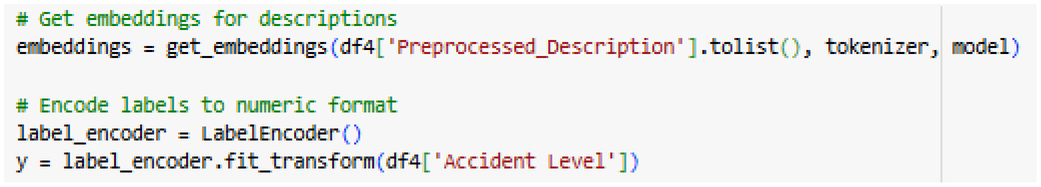
- Suboptimal Performance on Complex Tasks: For complex or nuanced text tasks, especially with substantial data, using pre-trained embeddings alone might be insufficient. Directly fine-tuning a transformer model can better capture specific context for classification.
- Lack of Task-Specific Adaptation: The embeddings are generated based on the original training objectives of BERT (masked language modeling and next sentence prediction), which might not perfectly align with accident classification tasks.

Defining the tokenizer and get embeddings function



```python
# Choose a model: 'bert-base-uncased' or 'roberta-base'
model_name = 'bert-base-uncased'  # or 'roberta-base' for RoBERTa

# Initialize tokenizer and model
if 'bert' in model_name:
    tokenizer = BertTokenizer.from_pretrained(model_name)
    model = BertModel.from_pretrained(model_name)
elif 'roberta' in model_name:
    tokenizer = RobertaTokenizer.from_pretrained(model_name)
    model = RobertaModel.from_pretrained(model_name)

def get_embeddings(text_list, tokenizer, model):
    """ Get embeddings for each text in text_list using tokenizer and model """
    with torch.no_grad():
        inputs = tokenizer(text_list, return_tensors="pt", padding=True, truncation=True, max_length=512)
        outputs = model(**inputs)
        embeddings = outputs.last_hidden_state[:, 0, :]  # Get the CLS token embedding
        return embeddings.numpy()
```

Get Embeddings on cleaned data and Label Encoding



```python
# Get embeddings for descriptions
embeddings = get_embeddings(df4['Preprocessed_Description'].tolist(), tokenizer, model)

# Encode labels to numeric format
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(df4['Accident Level'])
```

Different classifiers on cleaned data's BERT embeddings

```
# Train various classification models
classifier = {
    # "Naive_Bayes": MultinomialNB(),
    "Logistic_Regression": LogisticRegression(random_state=1),
    "SVM": SVC(random_state=1),
    "Random_Forest": RandomForestClassifier(random_state=1),
    "Gradient_Boosting": GradientBoostingClassifier(random_state=1)
}

# Initialize an empty list to store classification metrics
metrics_list = []

for model_name, clf in classifier.items():
    # Train the model
    clf.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = clf.predict(X_test)

    # Get the classification report as a string
    report_str = classification_report(y_test, y_pred, zero_division=0)

    # Get the classification report as a dictionary
    report_dict = classification_report(y_test, y_pred, output_dict=True, zero_division=0)

    # Extract accuracy, precision, recall, and F1-score (average metrics for each model)
    accuracy = accuracy_score(y_test, y_pred)
    precision = report_dict['weighted avg']['precision']
    recall = report_dict['weighted avg']['recall']
    f1_score = report_dict['weighted avg']['f1-score']

    # Append the metrics to the list
    metrics_list.append({
        'Model': model_name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1-Score': f1_score
    })

    # Print the classification report for the current model
    print(f"Classification Report for {model_name}:")
    print(report_str)
    print("=" * 60)  # Divider for clarity

# Convert the list of metrics into a DataFrame for comparison
metrics_comparison = pd.DataFrame(metrics_list)

# Display the comparison of metrics for each model
print("Comparison of Metrics:")
print(metrics_comparison)
```

Result of running different classifiers on cleaned data's BERT embeddings:

```
------------------------------------------------------------
Comparison of Metrics:
                 Model  Accuracy  Precision    Recall  F1-Score
0  Logistic_Regression  0.690476   0.557692  0.690476  0.617021
1                  SVM  0.750000   0.562500  0.750000  0.642857
2        Random_Forest  0.750000   0.562500  0.750000  0.642857
3    Gradient_Boosting  0.702381   0.597211  0.702381  0.644962
```

Insight: It is observed that test accuracy of all the models falls in the range of 69-75% and test F-1 score falls in the range of 61-64% range

Get Embeddings, Label Encoding on raw data and Create Train and Test Data:

```python
# Get embeddings for raw descriptions
embeddings = get_embeddings(data_original['Description'].tolist(), tokenizer, model)

# Encode labels to numeric format
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(data_original['Accident Level'])

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(embeddings, y, test_size=0.2, random_state=42)
```

Result of running different classifiers on Raw data's BERT embeddings:

```
Comparison of Metrics:
                 Model  Accuracy  Precision    Recall  F1-Score
0  Logistic_Regression  0.764706   0.679053  0.764706  0.719041
1                  SVM  0.800000   0.640000  0.800000  0.711111
2        Random_Forest  0.800000   0.640000  0.800000  0.711111
3    Gradient_Boosting  0.741176   0.630000  0.741176  0.681081
```

Insight:

It is observed that **test accuracy** of all the models trained on **raw data's BERT embeddings** falls in the range of **74-80%** and **test F-1 score** falls in the range of **68-72%** range, whereas **test accuracy** of all the models trained on **cleaned data's BERT embeddings** falls in the range of **69-75%** and **test F-1 score** falls in the range of **61-64%** range.

Recommendation for next steps:

To improve accident severity classification:

- Use **fine-tuned RoBERTa or BERT like models** as the primary model for sequence classification.
- Reserve traditional classifiers for simpler tasks or as benchmarks for performance comparison. Fine-tuned transformers offer clear advantages in capturing context, handling imbalanced classes, and achieving higher accuracy and F1-scores.

Note: The classifiers are performing better on raw data. Therefore, we will try to create more models on raw data and check if we are getting improved performance

- Preprocessing is not needed when using pre-trained language representation models like BERT. It uses all of the information in a sentence, punctuation and stop-words from a wide range of perspectives by leveraging a multi-head self attention mechanism.

Second Approach: Fine-Tuning Transformer Models for Sequence Classification

Here, a pre-trained model (like BERT, RoBERTa, DistilBERT, XLNet, or ALBERT) is fine-tuned specifically for accident level classification by adding a classification layer and optimizing the model end-to-end on the accident description data.

Steps:

- Use a pre-trained model and add a classification layer.
- Fine-tune the entire model, optimizing it for accident level classification on the specific dataset.

Advantages:

- Task-Specific Optimization: Fine-tuning adjusts the model's parameters specifically for accident classification, often resulting in higher accuracy than using general-purpose embeddings.
- Superior Performance on Large Datasets: Transformer models can leverage large datasets well, extracting intricate patterns that are often crucial for text classification tasks.
- Better Handling of Class Imbalances and Nuances: Models can better handle nuanced differences between classes when trained end-to-end, which is especially useful for tasks with a high degree of overlap in descriptions. Contextual Understanding: By fine-tuning, the model learns to pay attention to specific accident-related language, providing a deeper and more context-aware representation.

Limitations:

- Higher Computational Requirements: Fine-tuning transformers is computationally expensive, especially with large models, as it requires significant GPU resources.
- Risk of Overfitting on Small Datasets: On small datasets, fine-tuning can lead to overfitting, especially if regularization techniques are not applied. Longer Training Time: End-to-end training requires more time compared to using precomputed embeddings and classifiers.
- Complexity in Hyperparameter Tuning: Fine-tuning involves several hyperparameters (e.g., learning rate, batch size, number of epochs) which must be tuned to avoid underfitting or overfitting

Why we are using models like BERT, Roberta, DistilBert, XLNet, Albert on Raw Data:

Below is the brief description on why we selected these 5 models to classify accident levels and get improved model performance:

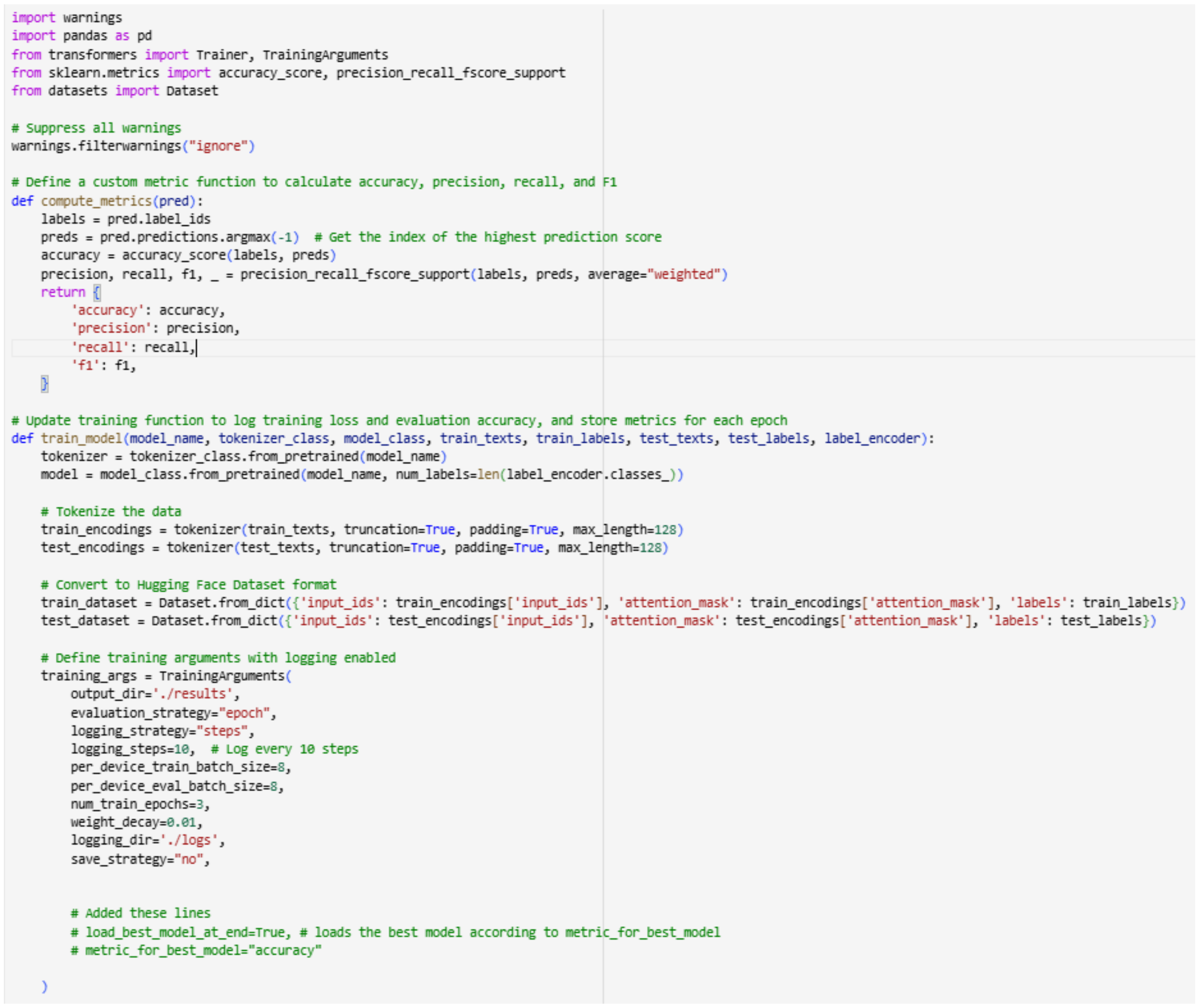
1. **BERT (Bidirectional Encoder Representations from Transformers):** Good for general NLP classification tasks with context-rich descriptions. You can start with bert-base-uncased.
2. **DistilBERT:** A smaller, faster variant of BERT with comparable accuracy for many tasks. If computational efficiency is a concern, distilbert-base-uncased is a good choice.
3. **RoBERTa:** A robustly optimized variant of BERT with superior performance on longer sequences (roberta-base or roberta-large).
4. **XLNet:** If the accident descriptions often include unusual phrasing or complex dependencies, XLNet may be more effective.
5. **ALBERT:** Good for large datasets due to its efficiency and lower memory footprint, without sacrificing much accuracy.

<u>Below is the overview of the steps we are likely to follow:</u>

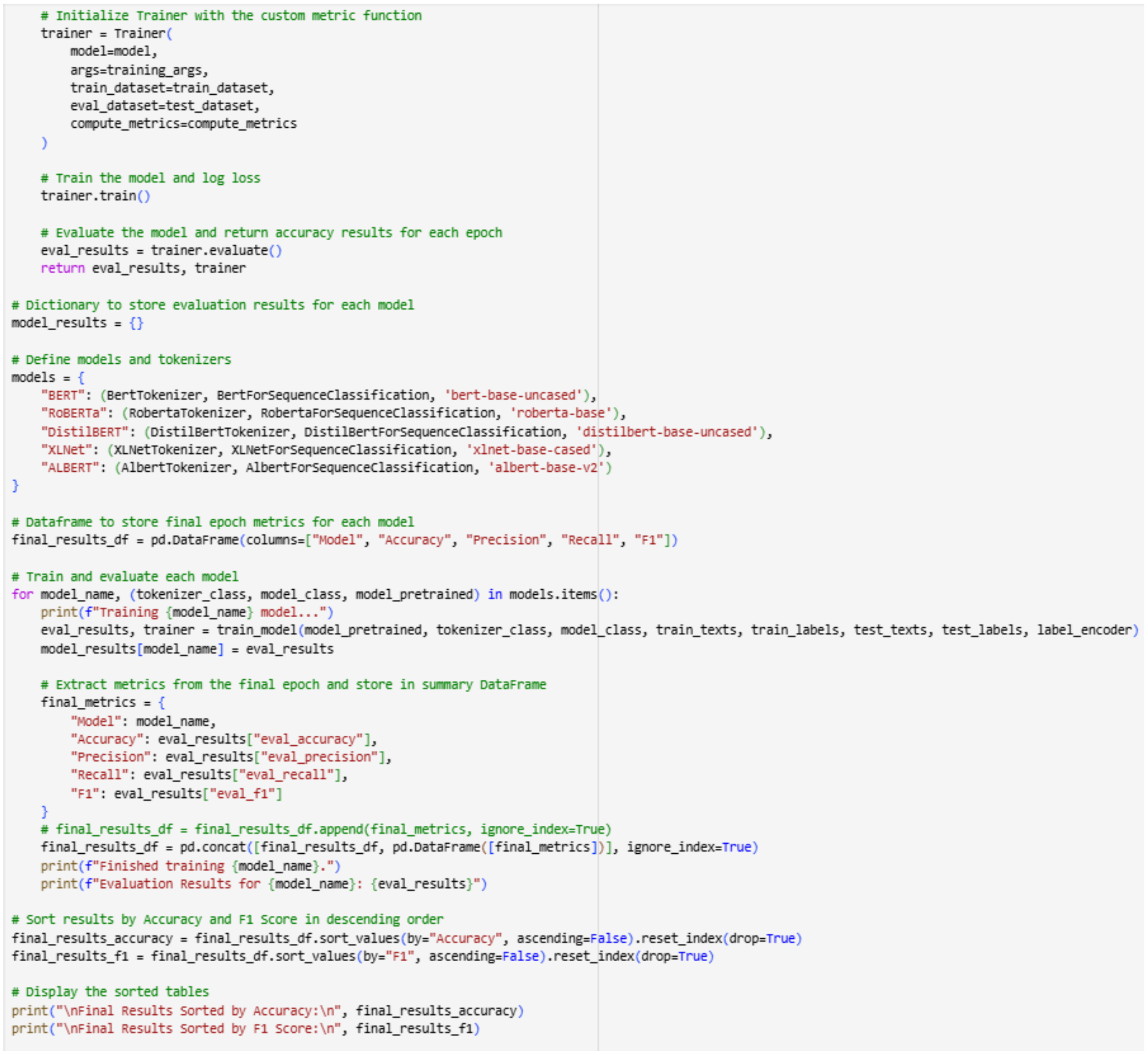
1. Load a pre-trained model (e.g., BERT) and tokenize the accident descriptions using the corresponding tokenizer.
2. Add a classification head (fully connected layer) for multi-class classification.
3. Fine-tune the model on the dataset using the accident severity level as the target variable.

1st Attempt on Fine-Tuning Transformer models for sequence classification:

Define Model Selection, Training Process, Run Each Model, Evaluate Performance and Compare Model Results using 3 epochs:

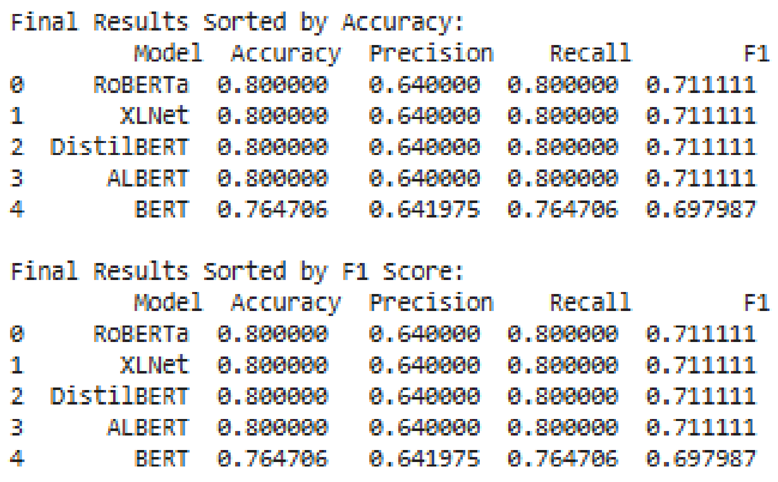


```
import warnings
import pandas as pd
from transformers import Trainer, TrainingArguments
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from datasets import Dataset

# Suppress all warnings
warnings.filterwarnings("ignore")

# Define a custom metric function to calculate accuracy, precision, recall, and F1
def compute_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)  # Get the index of the highest prediction score
    accuracy = accuracy_score(labels, preds)
    precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average="weighted")
    return {
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1': f1,
    }

# Update training function to log training loss and evaluation accuracy, and store metrics for each epoch
def train_model(model_name, tokenizer_class, model_class, train_texts, train_labels, test_texts, test_labels, label_encoder):
    tokenizer = tokenizer_class.from_pretrained(model_name)
    model = model_class.from_pretrained(model_name, num_labels=len(label_encoder.classes_))

    # Tokenize the data
    train_encodings = tokenizer(train_texts, truncation=True, padding=True, max_length=128)
    test_encodings = tokenizer(test_texts, truncation=True, padding=True, max_length=128)

    # Convert to Hugging Face Dataset format
    train_dataset = Dataset.from_dict({'input_ids': train_encodings['input_ids'], 'attention_mask': train_encodings['attention_mask'], 'labels': train_labels})
    test_dataset = Dataset.from_dict({'input_ids': test_encodings['input_ids'], 'attention_mask': test_encodings['attention_mask'], 'labels': test_labels})

    # Define training arguments with logging enabled
    training_args = TrainingArguments(
        output_dir='./results',
        evaluation_strategy="epoch",
        logging_strategy="steps",
        logging_steps=10,  # Log every 10 steps
        per_device_train_batch_size=8,
        per_device_eval_batch_size=8,
        num_train_epochs=3,
        weight_decay=0.01,
        logging_dir='./logs',
        save_strategy="no",


        # Added these lines
        # load_best_model_at_end=True, # loads the best model according to metric_for_best_model
        # metric_for_best_model="accuracy"

    )
```



```python
    # Initialize Trainer with the custom metric function
    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset,
        eval_dataset=test_dataset,
        compute_metrics=compute_metrics
    )

    # Train the model and log loss
    trainer.train()

    # Evaluate the model and return accuracy results for each epoch
    eval_results = trainer.evaluate()
    return eval_results, trainer

# Dictionary to store evaluation results for each model
model_results = {}

# Define models and tokenizers
models = {
    "BERT": (BertTokenizer, BertForSequenceClassification, 'bert-base-uncased'),
    "RoBERTa": (RobertaTokenizer, RobertaForSequenceClassification, 'roberta-base'),
    "DistilBERT": (DistilBertTokenizer, DistilBertForSequenceClassification, 'distilbert-base-uncased'),
    "XLNet": (XLNetTokenizer, XLNetForSequenceClassification, 'xlnet-base-cased'),
    "ALBERT": (AlbertTokenizer, AlbertForSequenceClassification, 'albert-base-v2')
}

# Dataframe to store final epoch metrics for each model
final_results_df = pd.DataFrame(columns=["Model", "Accuracy", "Precision", "Recall", "F1"])

# Train and evaluate each model
for model_name, (tokenizer_class, model_class, model_pretrained) in models.items():
    print(f"Training {model_name} model...")
    eval_results, trainer = train_model(model_pretrained, tokenizer_class, model_class, train_texts, train_labels, test_texts, test_labels, label_encoder)
    model_results[model_name] = eval_results

    # Extract metrics from the final epoch and store in summary DataFrame
    final_metrics = {
        "Model": model_name,
        "Accuracy": eval_results["eval_accuracy"],
        "Precision": eval_results["eval_precision"],
        "Recall": eval_results["eval_recall"],
        "F1": eval_results["eval_f1"]
    }
    # final_results_df = final_results_df.append(final_metrics, ignore_index=True)
    final_results_df = pd.concat([final_results_df, pd.DataFrame([final_metrics])], ignore_index=True)
    print(f"Finished training {model_name}.")
    print(f"Evaluation Results for {model_name}: {eval_results}")

# Sort results by Accuracy and F1 Score in descending order
final_results_accuracy = final_results_df.sort_values(by="Accuracy", ascending=False).reset_index(drop=True)
final_results_f1 = final_results_df.sort_values(by="F1", ascending=False).reset_index(drop=True)

# Display the sorted tables
print("\nFinal Results Sorted by Accuracy:\n", final_results_accuracy)
print("\nFinal Results Sorted by F1 Score:\n", final_results_f1)
```

Result and Insights:



```
Final Results Sorted by Accuracy:
         Model  Accuracy  Precision    Recall        F1
0      RoBERTa  0.800000   0.640000  0.800000  0.711111
1        XLNet  0.800000   0.640000  0.800000  0.711111
2   DistilBERT  0.800000   0.640000  0.800000  0.711111
3       ALBERT  0.800000   0.640000  0.800000  0.711111
4         BERT  0.764706   0.641975  0.764706  0.697987

Final Results Sorted by F1 Score:
         Model  Accuracy  Precision    Recall        F1
0      RoBERTa  0.800000   0.640000  0.800000  0.711111
1        XLNet  0.800000   0.640000  0.800000  0.711111
2   DistilBERT  0.800000   0.640000  0.800000  0.711111
3       ALBERT  0.800000   0.640000  0.800000  0.711111
4         BERT  0.764706   0.641975  0.764706  0.697987
```

- The best test accuracy(80%) and best test F1 score(71%) of RoBERTa model is similar to the first approach where BERT embeddings and traditional classifiers were used where test accuracy was the in the range of 74-80% and test F-1 score falls in the range of 68-72%.

<u>Code Analysis and Function:</u>

The above code is fine-tuning a pre-trained transformer model (like BERT, RoBERTa, DistilBERT, XLNet, or ALBERT) for accident level classification.

1. <u>Fine-Tuning with Custom Classification Layer:</u>
    a. Each transformer model (e.g., BERT, RoBERTa, etc.) is loaded using its pre-trained weights (e.g., 'bert-base-uncased' for BERT).
    b. A custom classification layer with num_labels equal to the number of accident levels is added to the model. This layer is trained from scratch, adapting the model specifically for accident level classification.
2. <u>Tokenization and Dataset Preparation:</u>
    a. The accident descriptions are tokenized using the tokenizer corresponding to each model, truncating or padding as necessary to ensure a uniform input size.
    b. The tokenized data is then converted into a format compatible with Hugging Face’s Dataset class, making it compatible with the Trainer.
3. <u>TrainingArguments for Fine-Tuning:</u>
    a. The TrainingArguments include multiple fine-tuning-specific configurations, such as evaluation_strategy="epoch", num_train_epochs=3, and weight_decay=0.01. These parameters are chosen for fine-tuning purposes rather than training from scratch.
    b. Fine-tuning requires adjustments in model parameters and hyperparameters to optimize for the accident classification task specifically, such as learning_rate, batch_size, and num_train_epochs.
4. <u>Trainer and Training:</u>
    a. The Trainer is initialized with the model, training arguments, and a compute_metrics function to evaluate accuracy, precision, recall, and F1 score during training.

    b. Fine-tuning occurs when trainer.train() is called. It updates not only the classification layer but also the transformer's pre-trained layers, adapting the model for the accident classification task.

5. <u>Evaluation and Results Storage:</u>
    a. After training, the model is evaluated on the test dataset, and evaluation metrics from the final epoch are extracted and stored.
    b. The final_results_df is created to summarize and compare model performance (sorted by accuracy and F1 score) across different transformer models.

Summary:

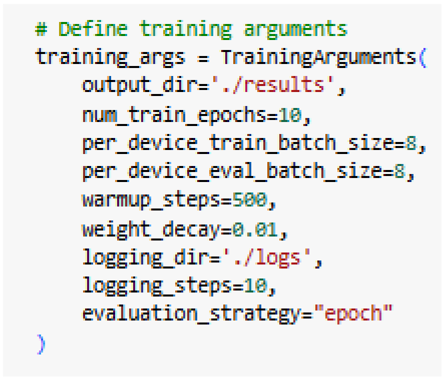
This Code Performs Fine-Tuning:

- The code adapts pre-trained transformers to a new classification task (accident level prediction) by adding a task-specific classification head and adjusting model weights to improve performance on this particular task.
- This approach leverages the existing language understanding in transformers, which has been acquired from extensive pre-training on large datasets.
- Requires less data and computational resources than training from scratch.

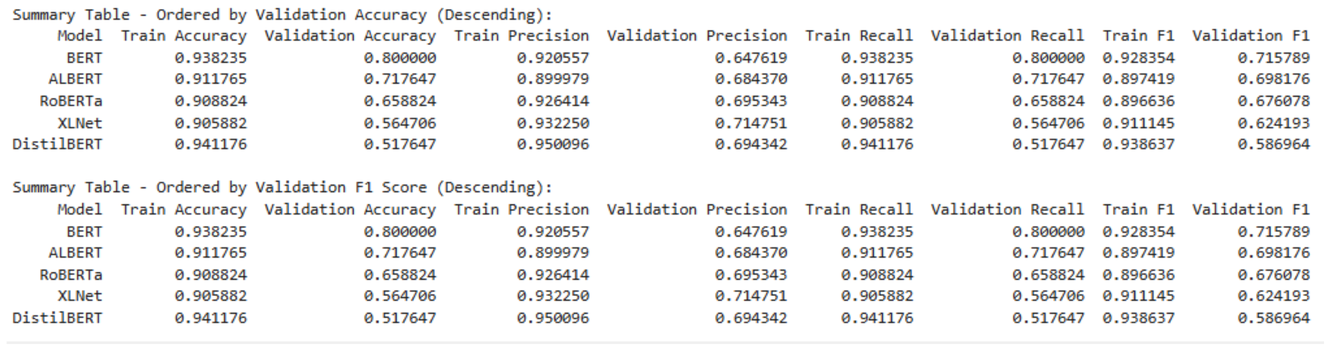
2nd Attempt on Fine-Tuning Transformer models for sequence classification:

Exploring ways to get test metrics along with train metrics and trying to get better metrics after increasing the epochs from 3 in the previous attempt to 10 epochs in this step.

We modified the training arguments as below to use 10 epochs:


```
# Define training arguments
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=10,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
    evaluation_strategy="epoch"
)
```

Result and Insights:

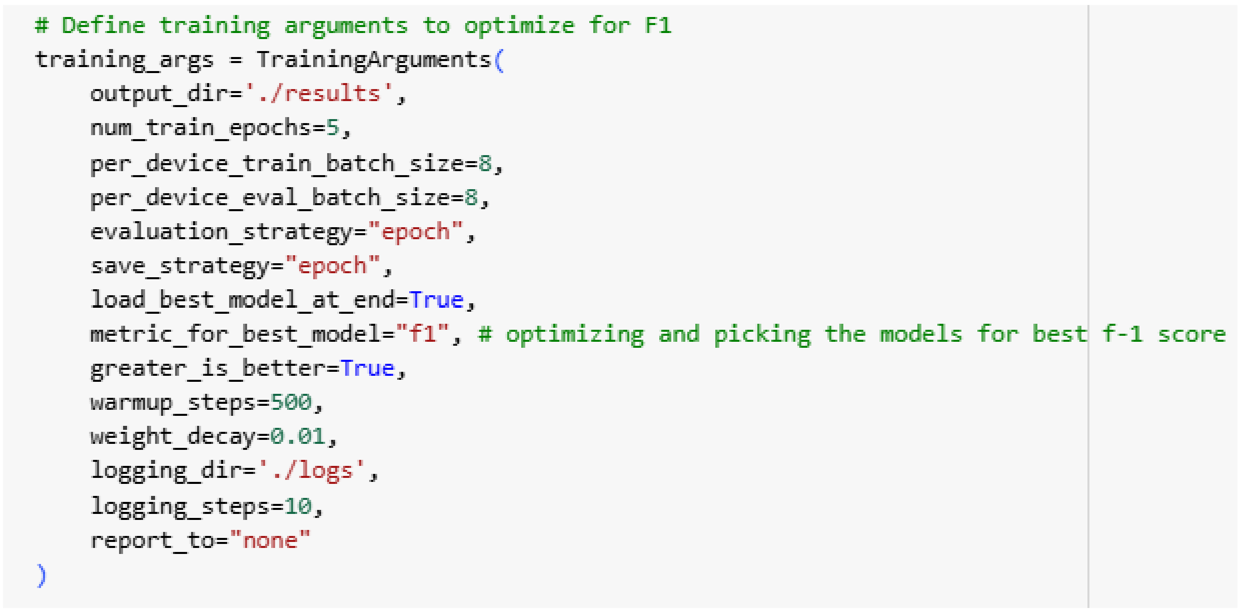


```
Summary Table - Ordered by Validation Accuracy (Descending):
     Model  Train Accuracy  Validation Accuracy  Train Precision  Validation Precision  Train Recall  Validation Recall  Train F1  Validation F1
      BERT        0.938235             0.800000         0.920557              0.647619      0.938235           0.800000  0.928354       0.715789
    ALBERT        0.911765             0.717647         0.899979              0.684370      0.911765           0.717647  0.897419       0.698176
   RoBERTa        0.908824             0.658824         0.926414              0.695343      0.908824           0.658824  0.896636       0.676078
     XLNet        0.905882             0.564706         0.932250              0.714751      0.905882           0.564706  0.911145       0.624193
DistilBERT        0.941176             0.517647         0.950096              0.694342      0.941176           0.517647  0.938637       0.586964

Summary Table - Ordered by Validation F1 Score (Descending):
     Model  Train Accuracy  Validation Accuracy  Train Precision  Validation Precision  Train Recall  Validation Recall  Train F1  Validation F1
      BERT        0.938235             0.800000         0.920557              0.647619      0.938235           0.800000  0.928354       0.715789
    ALBERT        0.911765             0.717647         0.899979              0.684370      0.911765           0.717647  0.897419       0.698176
   RoBERTa        0.908824             0.658824         0.926414              0.695343      0.908824           0.658824  0.896636       0.676078
     XLNet        0.905882             0.564706         0.932250              0.714751      0.905882           0.564706  0.911145       0.624193
DistilBERT        0.941176             0.517647         0.950096              0.694342      0.941176           0.517647  0.938637       0.586964
```

- After increasing the epochs we see that there is a significant difference between the training and testing metrics.
- When the epochs are increased then the model gets trained too well on the training data and might not generalize similarly well on the testing data.
- The test metrics are similar to the previous run of the models when 3 epochs were chosen.

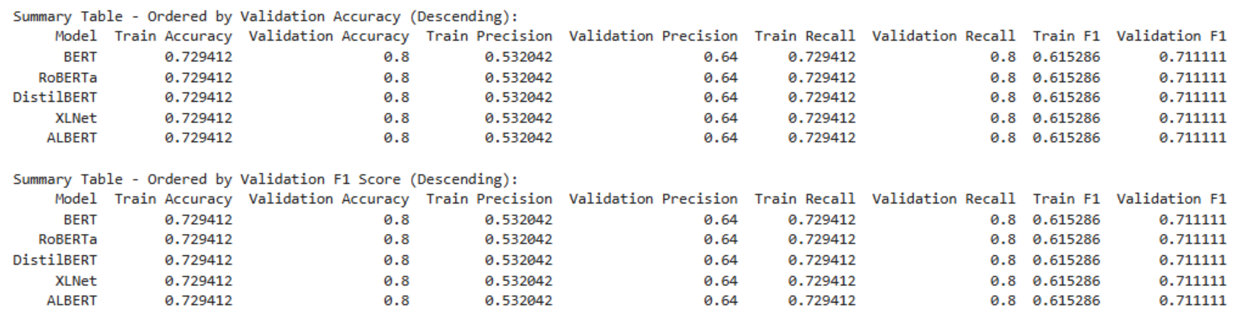
3rd Attempt on Fine-Tuning Transformer models for sequence classification:

Modified the above code to pick the best model based on F1 score. The **F1 score** is the harmonic mean of precision and recall, balancing these two metrics into a single value. It is particularly suited for cases where there is huge **class imbalance and minority class predictions are critical.** We used 5 epochs and optimized to pick the model for best f-1 score by modifying the below training arguments:



```python
# Define training arguments to optimize for F1
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="f1", # optimizing and picking the models for best f-1 score
    greater_is_better=True,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
    report_to="none"
)
```

Result and Insights:



```
Summary Table - Ordered by Validation Accuracy (Descending):
      Model  Train Accuracy  Validation Accuracy  Train Precision  Validation Precision  Train Recall  Validation Recall  Train F1  Validation F1
       BERT        0.729412                  0.8         0.532042                  0.64      0.729412                0.8  0.615286       0.711111
    RoBERTa        0.729412                  0.8         0.532042                  0.64      0.729412                0.8  0.615286       0.711111
 DistilBERT        0.729412                  0.8         0.532042                  0.64      0.729412                0.8  0.615286       0.711111
      XLNet        0.729412                  0.8         0.532042                  0.64      0.729412                0.8  0.615286       0.711111
     ALBERT        0.729412                  0.8         0.532042                  0.64      0.729412                0.8  0.615286       0.711111

Summary Table - Ordered by Validation F1 Score (Descending):
      Model  Train Accuracy  Validation Accuracy  Train Precision  Validation Precision  Train Recall  Validation Recall  Train F1  Validation F1
       BERT        0.729412                  0.8         0.532042                  0.64      0.729412                0.8  0.615286       0.711111
    RoBERTa        0.729412                  0.8         0.532042                  0.64      0.729412                0.8  0.615286       0.711111
 DistilBERT        0.729412                  0.8         0.532042                  0.64      0.729412                0.8  0.615286       0.711111
      XLNet        0.729412                  0.8         0.532042                  0.64      0.729412                0.8  0.615286       0.711111
     ALBERT        0.729412                  0.8         0.532042                  0.64      0.729412                0.8  0.615286       0.711111
```

- The test metrics are similar to the previous attempt's result.

The validation/Test metrics are better compared to the train metrics because of the following reasons:

- Data Imbalance: If the training data has a higher proportion of a certain class or is less diverse, the model might underperform during training but generalize better on a more balanced or diverse validation set.
- Regularization Effects: Regularization techniques like dropout or weight decay may help the model generalize better on the validation set, even if it struggles to achieve higher accuracy during training.
- Overfitting Prevention: If the model is not fully fitting to the training data (e.g., due to early stopping or fewer epochs), it might actually perform better on the validation set due to an implicit regularization effect.

Several pre-trained transformer models (e.g., BERT, RoBERTa, DistilBERT, XLNet, and ALBERT) were fine-tuned on a classification task. The training is structured to prioritize optimizing the F1 score by setting it as the metric for selecting the best model and using early stopping to prevent overfitting.

F1 Score Improvement with Each Epoch:

The training process is structured to prioritize improving the F1 score because:

- **metric_for_best_mode**l is set to "f1", so the best model is determined based on F1 score.
- **EarlyStoppingCallback** monitors F1 score on the validation set, stopping training if there's no improvement for 3 epochs.

The setup aims to improve F1 score specifically, though other metrics are tracked. In each epoch, the validation F1 score is recalculated, and if there is an improvement, the best-performing model is saved.

4th Attempt on Fine-Tuning Transformer models for sequence classification:

We will balance the data by Random Over Sampler and run the models. **RandomOverSampler** is a specific type of resampling focused on oversampling by duplicating samples from minority classes without generating synthetic data. It randomly duplicates existing samples until the minority classes reach the same size as the majority class. RandomOverSampler is crucial in mitigating class imbalance by ensuring that the minority classes are sufficiently represented. This can lead to:

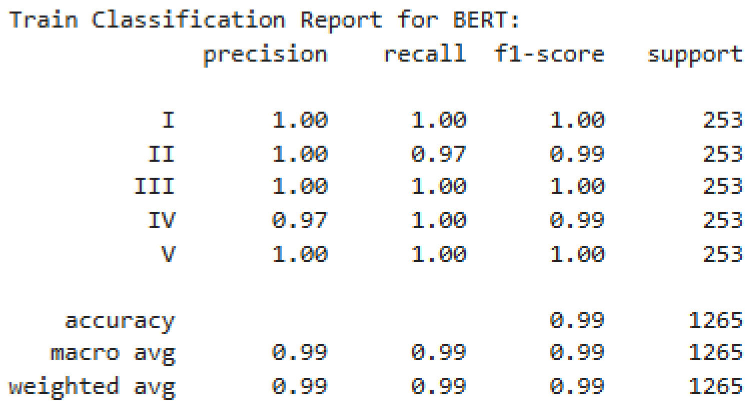
- **Improved Recall** for minority classes, as the model is exposed to a balanced number of examples from each class.
- **Increased F1 Score** due to a better balance between precision and recall across all classes.

- **Reduction of Majority Class Bias**, leading to fairer and more reliable predictions across the severity levels in the dataset.
- **Simple and Fast**: Since it duplicates actual data points rather than creating synthetic ones, it is computationally faster and straightforward to implement.

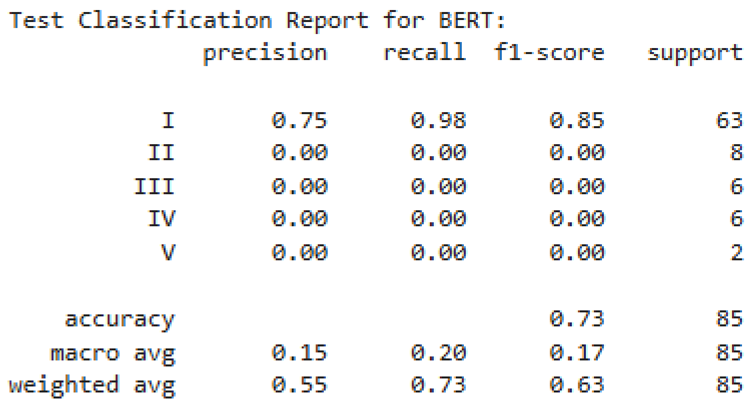
Limitations:

- **Increased Overfitting Risk**: Duplicating minority-class samples can lead to overfitting, especially if the model memorizes duplicated samples rather than learning generalizable features. This is particularly a concern with smaller datasets.
- **Lack of Diversity in Data**: RandomOverSampler does not add any new or diverse information to the dataset. In contrast, methods like SMOTE introduce some variation by creating synthetic samples, which can sometimes help the model generalize better.

Result and Insights:



```
Train Classification Report for BERT:
              precision    recall  f1-score   support

           I       1.00      1.00      1.00       253
          II       1.00      0.97      0.99       253
         III       1.00      1.00      1.00       253
          IV       0.97      1.00      0.99       253
           V       1.00      1.00      1.00       253

    accuracy                           0.99      1265
   macro avg       0.99      0.99      0.99      1265
weighted avg       0.99      0.99      0.99      1265
```



```
Test Classification Report for BERT:
              precision    recall  f1-score   support

           I       0.75      0.98      0.85        63
          II       0.00      0.00      0.00         8
         III       0.00      0.00      0.00         6
          IV       0.00      0.00      0.00         6
           V       0.00      0.00      0.00         2

    accuracy                           0.73        85
   macro avg       0.15      0.20      0.17        85
weighted avg       0.55      0.73      0.63        85
```

- The training metrics are performing way better than the testing metrics specially for the minority classes.
- Minority classes are not predicted well.

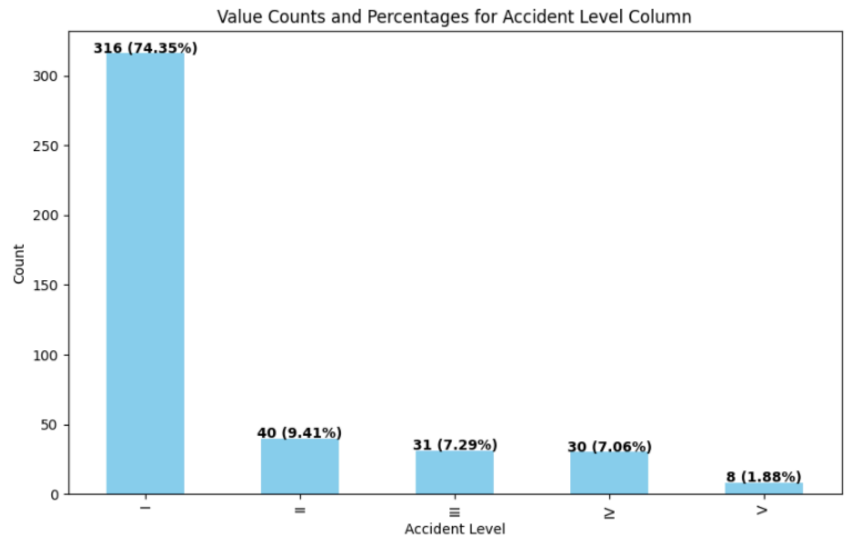
```
Summary Table - Ordered by Validation Accuracy (Descending):
       Model  Train Accuracy  Test Accuracy  Train Precision  Test Precision  Train Recall  Test Recall  Train F1  Test F1
        BERT        0.994466       0.729412         0.994615        0.553650      0.994466     0.729412  0.994465 0.629492
     RoBERTa        0.994466       0.729412         0.994615        0.586657      0.994466     0.729412  0.994465 0.641258
      ALBERT        0.993676       0.729412         0.993806        0.547059      0.993676     0.729412  0.993675 0.625210
       XLNet        0.995257       0.705882         0.995367        0.590118      0.995257     0.705882  0.995256 0.642583
  DistilBERT        0.994466       0.694118         0.994615        0.553537      0.994466     0.694118  0.994465 0.615907

Summary Table - Ordered by Validation F1 Score (Descending):
       Model  Train Accuracy  Test Accuracy  Train Precision  Test Precision  Train Recall  Test Recall  Train F1  Test F1
       XLNet        0.995257       0.705882         0.995367        0.590118      0.995257     0.705882  0.995256 0.642583
     RoBERTa        0.994466       0.729412         0.994615        0.586657      0.994466     0.729412  0.994465 0.641258
        BERT        0.994466       0.729412         0.994615        0.553650      0.994466     0.729412  0.994465 0.629492
      ALBERT        0.993676       0.729412         0.993806        0.547059      0.993676     0.729412  0.993675 0.625210
  DistilBERT        0.994466       0.694118         0.994615        0.553537      0.994466     0.694118  0.994465 0.615907
```

- BERT is performing best in terms of Test accuracy and XLNET is performing best in terms of test F1 score.
- None of the models show improvement over the previous run of models.
- The performance of the model degraded after over sampling the data. The Best test accuracy decreased from 80% to 72% and Best F1 score decreased from 71% to 64%.

Modifying data and Fine-Tune Transformer models for sequence classification:

- Balancing the dataset didn't give better performance.
- Here we will merge a few similar classes to overcome the performance issue caused by a huge imbalance in the dataset.
- The following class modification will be done:
  - **Accident level 1** will be mapped to **Low** accident severity
  - **Accident level 2 and 3** will be mapped to **Medium** accident severity
  - **Accident level 4 and 5** will be mapped to **High** accident severity

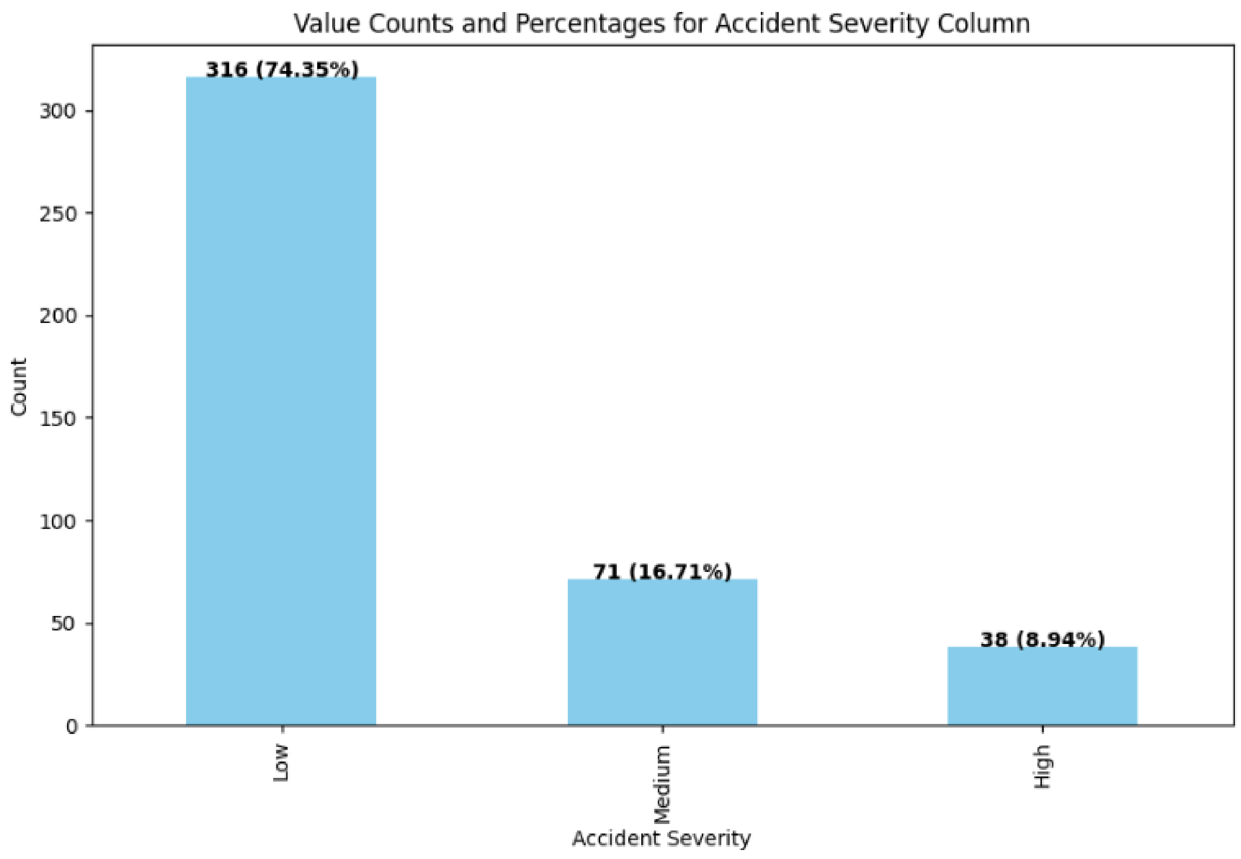
Before merging the accident level, below is the distribution of the accident levels:



- There is a huge imbalance in the data, therefore we will try to merge the classes and get more reliable results.
- We merged the classes as below

```python
# Map each accident level to the corresponding severity level
def map_severity(level):
    if level == 'I':
        return 'Low'
    elif level in ['II', 'III']:
        return 'Medium'
    elif level in ['IV', 'V']:
        return 'High'

# Apply the mapping function to create a new column
df['Accident Severity'] = df['Accident Level'].apply(map_severity)
```

- Below is the class distribution after merging the classes.



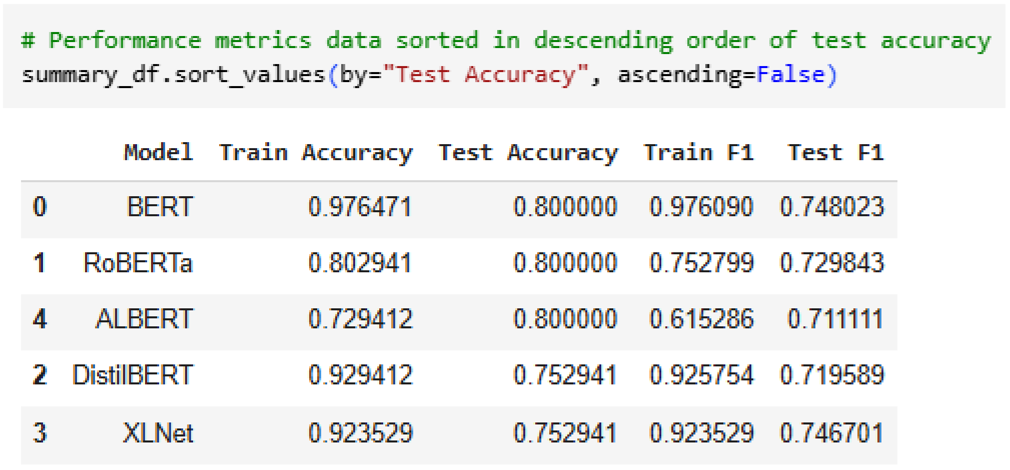
- After merging the accident levels we get a comparatively less imbalanced Accident Severity column.
- We are expecting to get better performance after merging the classes.

Result and Insights:

```
Train Classification Report for BERT:
              precision    recall  f1-score     support
0              0.966667  0.906250  0.935484   32.000000
1              0.980237  1.000000  0.990020  248.000000
2              0.964912  0.916667  0.940171   60.000000
accuracy       0.976471  0.976471  0.976471    0.976471
macro avg      0.970605  0.940972  0.955225  340.000000
weighted avg   0.976256  0.976471  0.976090  340.000000
Test Classification Report for BERT:
              precision    recall  f1-score  support
0              0.000000  0.000000  0.000000      6.0
1              0.825000  0.970588  0.891892     68.0
2              0.500000  0.181818  0.266667     11.0
accuracy       0.800000  0.800000  0.800000      0.8
macro avg      0.441667  0.384135  0.386186     85.0
weighted avg   0.724706  0.800000  0.748023     85.0
```

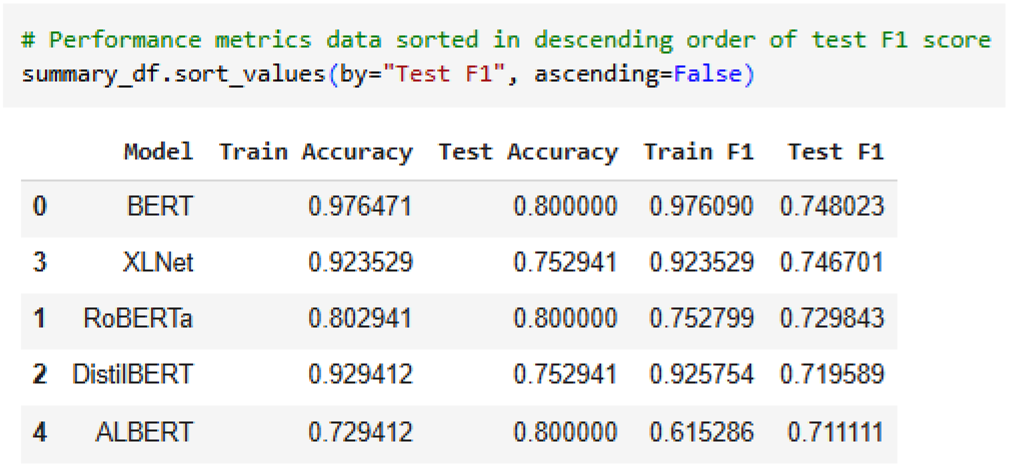
```
Train Classification Report for RoBERTa:
              precision    recall  f1-score     support
0              1.000000  0.031250  0.060606   32.000000
1              0.814570  0.991935  0.894545  248.000000
2              0.702703  0.433333  0.536082   60.000000
accuracy       0.802941  0.802941  0.802941    0.802941
macro avg      0.839091  0.485506  0.497078  340.000000
weighted avg   0.812281  0.802941  0.752799  340.000000
Test Classification Report for RoBERTa:
              precision    recall  f1-score  support
0              0.000000  0.000000  0.000000      6.0
1              0.807229  0.985294  0.887417     68.0
2              0.500000  0.090909  0.153846     11.0
accuracy       0.800000  0.800000  0.800000      0.8
macro avg      0.435743  0.358734  0.347088     85.0
weighted avg   0.710489  0.800000  0.729843     85.0
```

- It is observed from the classification report that the prediction of minority class improved compared to the previous attempts of the model run.
- There is still 1 out of the 3 classes which is not getting predicted.



```python
# Performance metrics data sorted in descending order of test accuracy
summary_df.sort_values(by="Test Accuracy", ascending=False)
```

	Model	Train Accuracy	Test Accuracy	Train F1	Test F1
0	BERT	0.976471	0.800000	0.976090	0.748023
1	RoBERTa	0.802941	0.800000	0.752799	0.729843
4	ALBERT	0.729412	0.800000	0.615286	0.711111
2	DistilBERT	0.929412	0.752941	0.925754	0.719589
3	XLNet	0.923529	0.752941	0.923529	0.746701



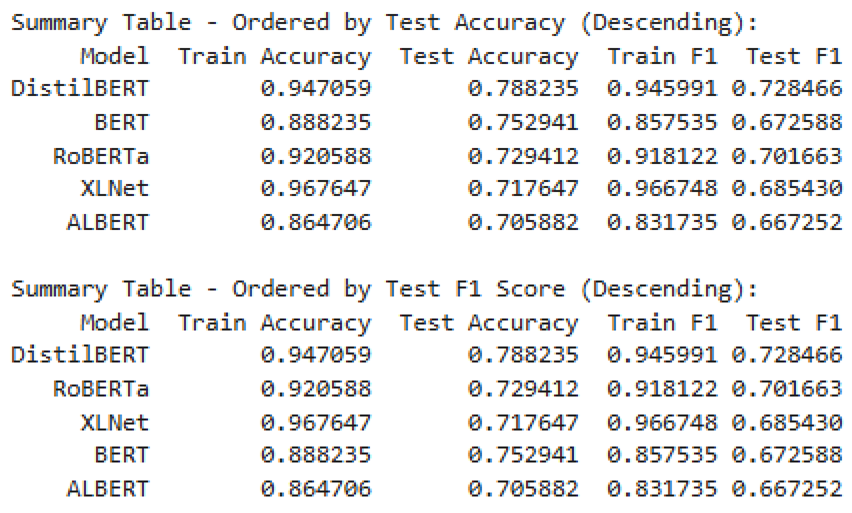
```python
# Performance metrics data sorted in descending order of test F1 score
summary_df.sort_values(by="Test F1", ascending=False)
```

	Model	Train Accuracy	Test Accuracy	Train F1	Test F1
0	BERT	0.976471	0.800000	0.976090	0.748023
3	XLNet	0.923529	0.752941	0.923529	0.746701
1	RoBERTa	0.802941	0.800000	0.752799	0.729843
2	DistilBERT	0.929412	0.752941	0.925754	0.719589
4	ALBERT	0.729412	0.800000	0.615286	0.711111

- **BERT** appears to be having the **best Test Accuracy(80%)** and **Test F1 Score(74.80%)**. However, it shows a significant amount of **overfitting** because there is around **15% difference** in train and test metrics.
- **The RoBERTa** model is giving **similar Test Accuracy(80%)** and around **73% of Test F1 score**. Moreover, there is **no overfitting** because the train and test metrics are almost the same.
- **RoBERTa seems to be a better choice than BERT** as it is generalizing well and giving comparable results to the BERT model.
- The minority class is still not getting predicted well, hence we will try balancing the data using Random over sampler and SMOTE.

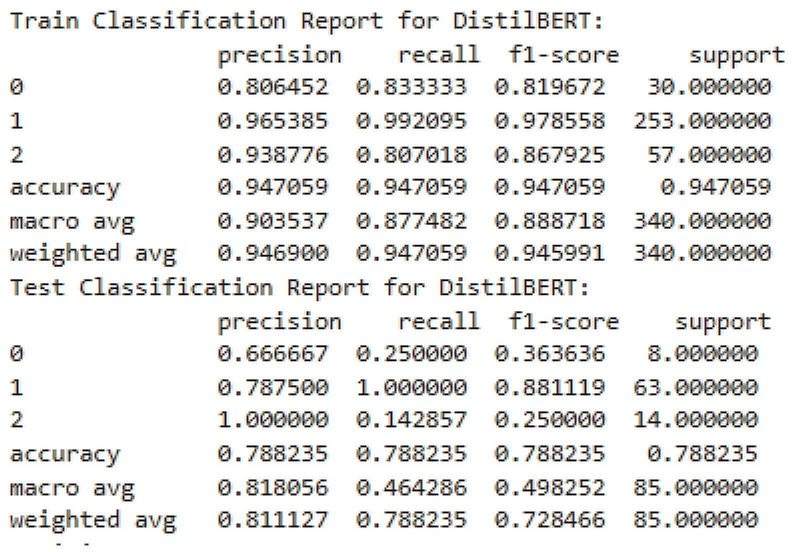
Balancing modified data and Fine-Tune Transformer models for sequence classification:

1. SMOTE - Balancing Data and Running the Model



```
Summary Table - Ordered by Test Accuracy (Descending):
     Model  Train Accuracy  Test Accuracy  Train F1  Test F1
DistilBERT        0.947059       0.788235  0.945991 0.728466
      BERT        0.888235       0.752941  0.857535 0.672588
   RoBERTa        0.920588       0.729412  0.918122 0.701663
     XLNet        0.967647       0.717647  0.966748 0.685430
    ALBERT        0.864706       0.705882  0.831735 0.667252

Summary Table - Ordered by Test F1 Score (Descending):
     Model  Train Accuracy  Test Accuracy  Train F1  Test F1
DistilBERT        0.947059       0.788235  0.945991 0.728466
   RoBERTa        0.920588       0.729412  0.918122 0.701663
     XLNet        0.967647       0.717647  0.966748 0.685430
      BERT        0.888235       0.752941  0.857535 0.672588
    ALBERT        0.864706       0.705882  0.831735 0.667252
```

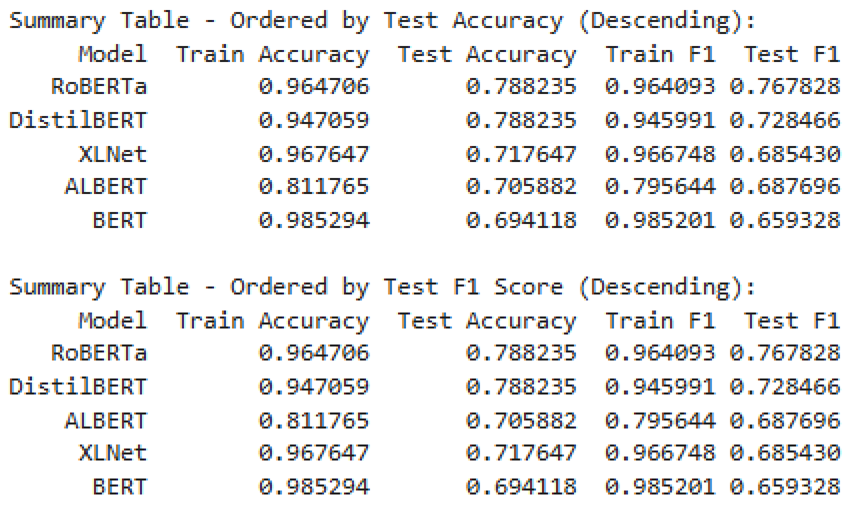
- **DistilBERT** is giving best results for both **test accuracy (78.82%)** and **Test F1 score(72.85%)**.
- **After balancing** the data **Test accuracy** of the best model **reduced** a bit from **80% to 78.82%** and **f1** score reduced from **73% to 72.84%**.
- We need to check if the performance of DistilBERT is better for minority classes as well through the classification report.



```
Train Classification Report for DistilBERT:
               precision    recall  f1-score     support
0               0.806452  0.833333  0.819672   30.000000
1               0.965385  0.992095  0.978558  253.000000
2               0.938776  0.807018  0.867925   57.000000
accuracy        0.947059  0.947059  0.947059    0.947059
macro avg       0.903537  0.877482  0.888718  340.000000
weighted avg    0.946900  0.947059  0.945991  340.000000
Test Classification Report for DistilBERT:
               precision    recall  f1-score    support
0               0.666667  0.250000  0.363636   8.000000
1               0.787500  1.000000  0.881119  63.000000
2               1.000000  0.142857  0.250000  14.000000
accuracy        0.788235  0.788235  0.788235   0.788235
macro avg       0.818056  0.464286  0.498252  85.000000
weighted avg    0.811127  0.788235  0.728466  85.000000
```

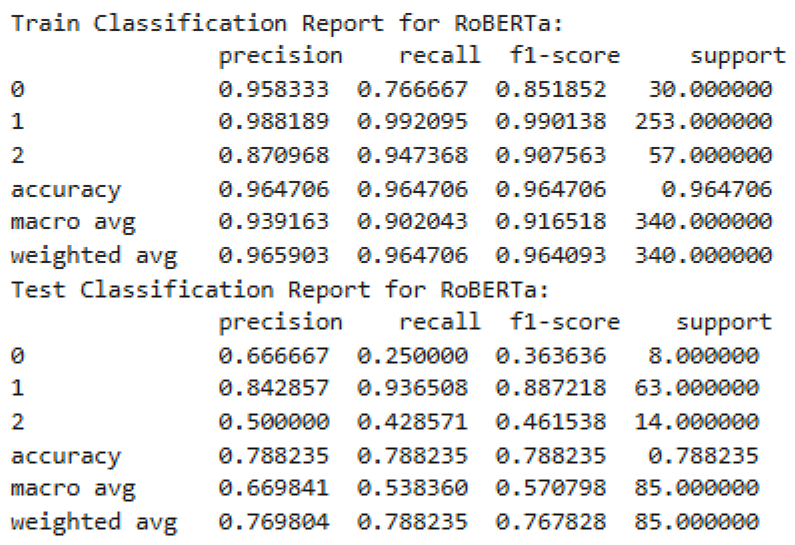
- We are getting decent precision, recall and f1 score for minority classes as well.
- We will now check the performance of the models after using Random over sampler.

2. Random Over Sampler - Balancing Data and Running the Model



```
Summary Table - Ordered by Test Accuracy (Descending):
     Model  Train Accuracy  Test Accuracy  Train F1  Test F1
   RoBERTa        0.964706       0.788235  0.964093 0.767828
DistilBERT        0.947059       0.788235  0.945991 0.728466
     XLNet        0.967647       0.717647  0.966748 0.685430
    ALBERT        0.811765       0.705882  0.795644 0.687696
      BERT        0.985294       0.694118  0.985201 0.659328

Summary Table - Ordered by Test F1 Score (Descending):
     Model  Train Accuracy  Test Accuracy  Train F1  Test F1
   RoBERTa        0.964706       0.788235  0.964093 0.767828
DistilBERT        0.947059       0.788235  0.945991 0.728466
    ALBERT        0.811765       0.705882  0.795644 0.687696
     XLNet        0.967647       0.717647  0.966748 0.685430
      BERT        0.985294       0.694118  0.985201 0.659328
```

- **RoBERTa** is giving best results for both **test accuracy (78.82%)** and **Test F1 score(76.78%)**.
- **After balancing** the data **Test accuracy** reduced a bit from **80% to 78.82%** but **f1** score improved from **73% to 76.78%**.
- We need to check if the performance of RoBERTa is better for minority classes as well through the **classification report.**



```
Train Classification Report for RoBERTa:
              precision    recall  f1-score     support
0              0.958333  0.766667  0.851852   30.000000
1              0.988189  0.992095  0.990138  253.000000
2              0.870968  0.947368  0.907563   57.000000
accuracy       0.964706  0.964706  0.964706    0.964706
macro avg      0.939163  0.902043  0.916518  340.000000
weighted avg   0.965903  0.964706  0.964093  340.000000
Test Classification Report for RoBERTa:
              precision    recall  f1-score    support
0              0.666667  0.250000  0.363636   8.000000
1              0.842857  0.936508  0.887218  63.000000
2              0.500000  0.428571  0.461538  14.000000
accuracy       0.788235  0.788235  0.788235   0.788235
macro avg      0.669841  0.538360  0.570798  85.000000
weighted avg   0.769804  0.788235  0.767828  85.000000
```

- We are getting decent precision, recall and f1 scores for minority classes and these scores are better than the DistilBERT model output using SMOTE balanced data.
- This seems to be the best overall output till now.

Best Model:

- We choose **RoBERTa** as the **best model** which is trained on the **balanced data** obtained from **Random Over Sampling.**
- **After balancing** the data **Test accuracy** reduced a bit from **80% to 78.82%** but **f1** score improved from **73% to 76.78%**. The **f1** score is the **best** among all the models attempted.
- This model also **predicts** the **minority classes effectively** compared to other models.
- Prediction of minority class was the biggest issue we faced, which got resolved to a great extent by using this model.

Random Over Sampler performed better than SMOTE because of the following reasons:

- **Textual Nature of Data**: SMOTE generates synthetic samples by interpolating between existing samples, which is challenging for textual data since interpolation doesn't preserve the semantic integrity of text. ROS, on the other hand, simply duplicates existing samples, avoiding semantic distortions.
- **Transformers Handle Redundancy Well**: Transformer models like BERT or RoBERTa are robust to repeated instances during training. They can still learn meaningful patterns without overfitting due to the model's architecture and regularization techniques like dropout.
- **Preservation of Original Data Distribution**: ROS preserves the original data distribution and ensures the generated samples are realistic, which is critical for text classification. SMOTE may introduce unrealistic or nonsensical text samples that can mislead the model during training.
- **Simpler Preprocessing**: ROS doesn't require additional computational steps like creating synthetic embeddings or tokenized data, making it more straightforward to implement in a transformer-based pipeline.

Reason why RoBERTa is chosen as the best model and how it is better than other BERT like models:

- **RoBERTa** was **trained** on significantly **more data** than BERT (160GB for RoBERTa vs. 16GB for BERT).
- Access to more training data allows RoBERTa to **better generalize patterns** in natural language, making it more robust across various domains and also helps to avoid overfitting.
- **BERT** uses **static masking**, where the same tokens are masked during every epoch of training.
- **RoBERTa** employs **dynamic masking**, where tokens are masked differently for each epoch. This leads to a **better understanding of the language structure** and improves the model's learning capacity.
- **RoBERTa's** pre-training helps it generalize across underrepresented classes, making it more **suitable for imbalanced datasets** where classes like "Medium" or "High" severity might have fewer samples.

The improvement in accuracy and F1 score after merging accident levels into three severity levels ("Low," "Medium," and "High") is primarily due to a combination of factors related to class distribution, simplification of the classification problem, and data representation:

1. Simplification of the Classification Problem

- **Reduced Complexity**: By reducing the classification from five to three classes, the model faces a simpler classification problem. Fewer classes mean the model has fewer distinctions to learn, making it easier to generalize patterns in the data.
- **Clearer Boundaries:** When fewer classes are present, the decision boundaries for each class become more defined. This helps the model to classify more accurately because it no longer needs to distinguish between similar classes (e.g., levels II and III or IV and V).
- **Less Class Overlap:** In many cases, classes that are merged (like levels II and III, or IV and V) have similar characteristics in the input data. This reduction can minimize overlapping features that the model might previously have confused, leading to more reliable predictions.

2. Better Class Distribution

- **Balanced Representation:** In multi-class classification, imbalanced classes can make training difficult for the model, as it tends to focus on the majority classes. By merging levels, you've effectively reduced some of the imbalance, giving the model a more balanced dataset, which can result in better performance.
- **Improved Sample Size per Class:** Merging classes also increases the number of samples in each class. Larger sample sizes per class allow the model to learn more representative patterns for each severity level, enhancing generalization and accuracy.

3. Enhanced Metrics Calculation

- **Accuracy and F1-Score Sensitivity:** With fewer classes and a more balanced dataset, both accuracy and F1-score metrics typically become more stable and meaningful. F1 score, especially, is sensitive to class imbalances and benefits from the reduction in complexity and increase in per-class sample size, providing a better assessment of model performance.

How This Change Impacts Performance:

The merging of classes effectively addresses issues of class imbalance and improves the dataset's representation across classes. As a result:

- **Reduced Overfitting**: The model is less likely to overfit on minority classes due to the increased sample sizes and simplified classification task.
- **Improved Generalizatio**n: A simpler classification task and more balanced data distribution mean the model is more likely to generalize well to new data, reflected in improved performance metrics like accuracy and F1 score.

Limitations:

- **Loss of Granularity:** Although performance improves, merging classes sacrifices some detail like the nuanced difference between adjacent accident levels(e.g., levels IV and V). This might potentially impact decision-making.
- **Computation Requirements:** Transformer models are computationally expensive and require significant resources for inference. This can be a bottleneck in real-time applications, especially in resource-constrained environments.

- **Less and highly imbalanced data:** The data is less and the available data for minority class is very less which makes it harder to train the model on the minority class and get correct predictions on the minority class.
- **Oversampling Bias**: While Random Over Sampling balances class distribution, it might lead the model to overfit duplicated samples, especially for minority classes. This can reduce the model's ability to identify subtle variations in underrepresented classes.
- **Generalization to New Data:** The model is fine-tuned on a specific dataset, and its ability to generalize to unseen real-world accident descriptions might be limited if the dataset doesn't represent the diversity of real-world cases.
- **Dependency on Data Quality:** Transformer models like RoBERTa heavily depend on high-quality, well-annotated data. Noise or ambiguities in accident descriptions can negatively impact model performance.

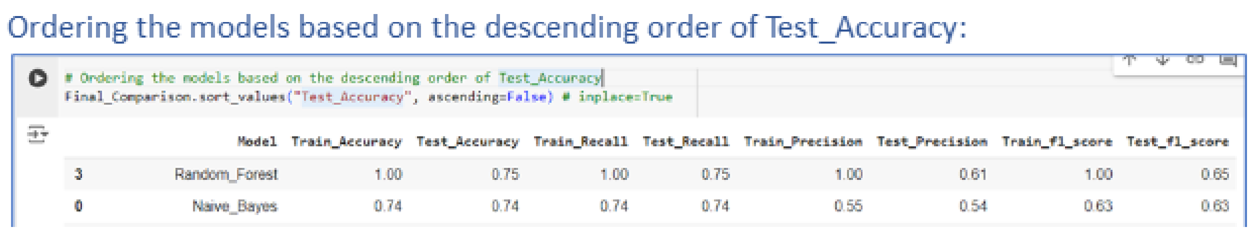
Possible Solution Enhancements:

- **Data Augmentation**: Instead of relying solely on Random Over Sampling, we can incorporate domain-specific data augmentation techniques to create diverse and realistic training samples.
- **Incorporate Domain Knowledge**: We can use domain-specific embeddings or pre-trained models fine-tuned on industry-relevant corpora to improve understanding of accident-specific terminology.
- **Active Learning**: We can iteratively refine the model using active learning by labeling high-uncertainty predictions. This helps adapt the model to evolving data distributions.
- **Explainability Tools**: Integrate explainability techniques (e.g., SHAP or LIME) to interpret model predictions. This is critical for real-world acceptance where decisions must be explainable.
- **Testing on Diverse Real-World Data**: We can test the model on real-world accident reports from various industries and geographies to ensure robustness. Retrain or fine-tune the model with additional data as needed.
- **Class-Imbalance Mitigation**: We can consider advanced balancing techniques like dynamic sampling or cost-sensitive learning to reduce dependence on oversampling methods like ROS.
- **Resource Optimization**: We can explore lightweight transformer models like DistilRoBERTa or quantization techniques for efficient inference in deployment environments.

By reducing the complexity of the target variable, we have optimized the model's learning process, resulting in a more reliable, accurate classification model.

Comparison to Benchmark:

The **benchmark** from 1st Milestone was **base Naive Bayes model** with **Test Accuracy of 74%** and Test **F1** score of **63%** as shown below:

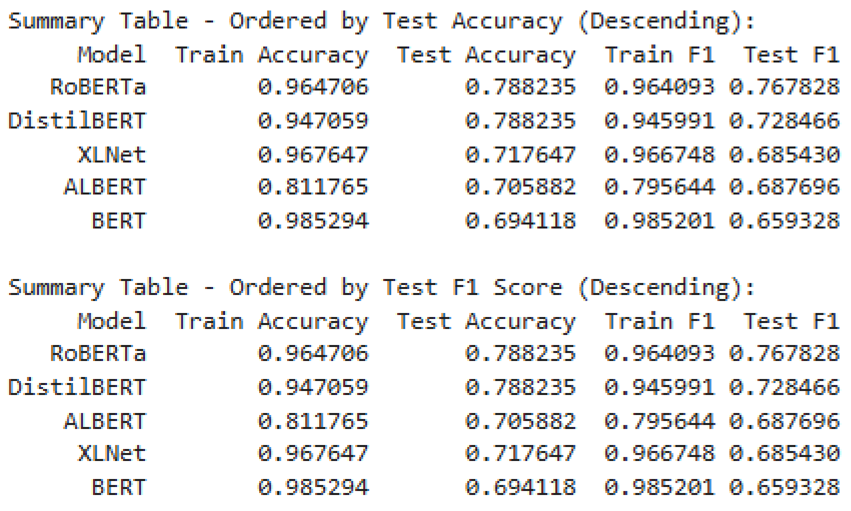


Ordering the models based on the descending order of Test_Accuracy:

```
# Ordering the models based on the descending order of Test_Accuracy
Final_Comparison.sort_values("Test_Accuracy", ascending=False) # inplace=True
```

	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision	Train_f1_score	Test_f1_score
3	Random_Forest	1.00	0.75	1.00	0.75	1.00	0.61	1.00	0.65
0	Naive_Bayes	0.74	0.74	0.74	0.74	0.55	0.54	0.63	0.63

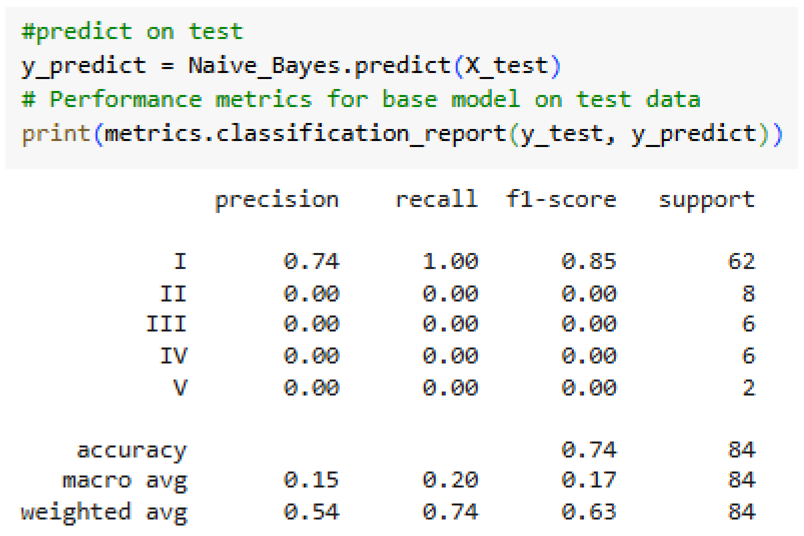
There is a significant improvement from the above benchmark by using RoBERTa model as shown below:



```
Summary Table - Ordered by Test Accuracy (Descending):
     Model  Train Accuracy  Test Accuracy  Train F1  Test F1
   RoBERTa        0.964706       0.788235  0.964093 0.767828
DistilBERT        0.947059       0.788235  0.945991 0.728466
     XLNet        0.967647       0.717647  0.966748 0.685430
    ALBERT        0.811765       0.705882  0.795644 0.687696
      BERT        0.985294       0.694118  0.985201 0.659328

Summary Table - Ordered by Test F1 Score (Descending):
     Model  Train Accuracy  Test Accuracy  Train F1  Test F1
   RoBERTa        0.964706       0.788235  0.964093 0.767828
DistilBERT        0.947059       0.788235  0.945991 0.728466
    ALBERT        0.811765       0.705882  0.795644 0.687696
     XLNet        0.967647       0.717647  0.966748 0.685430
      BERT        0.985294       0.694118  0.985201 0.659328
```

- **The RoBERTa** model is giving a significant(around **5%**) **improvement** of **Test Accuracy(78.82%)** over the **benchmark Test accuracy of 74%** and around **14% improvement** i.e **76.78% of Test F1 score** over the **benchmark Test F1 score of 63%**.

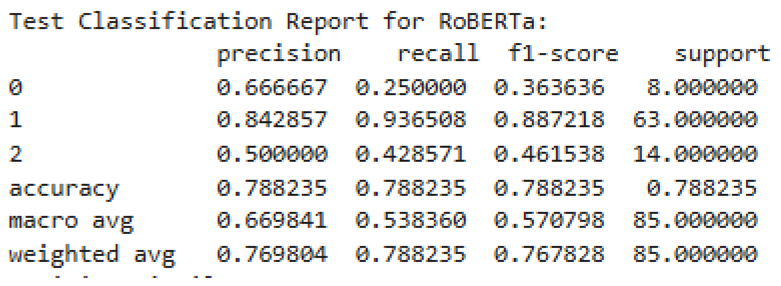
  Naive Bayes Classification Report:



```
#predict on test
y_predict = Naive_Bayes.predict(X_test)
# Performance metrics for base model on test data
print(metrics.classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           I       0.74      1.00      0.85        62
          II       0.00      0.00      0.00         8
         III       0.00      0.00      0.00         6
          IV       0.00      0.00      0.00         6
           V       0.00      0.00      0.00         2

    accuracy                           0.74        84
   macro avg       0.15      0.20      0.17        84
weighted avg       0.54      0.74      0.63        84
```

  RoBERTa(with Random Over Sampling) Classification Report:



```
Test Classification Report for RoBERTa:
             precision    recall  f1-score    support
0             0.666667  0.250000  0.363636   8.000000
1             0.842857  0.936508  0.887218  63.000000
2             0.500000  0.428571  0.461538  14.000000
accuracy      0.788235  0.788235  0.788235   0.788235
macro avg     0.669841  0.538360  0.570798  85.000000
weighted avg  0.769804  0.788235  0.767828  85.000000
```

- **Minority classes** were **not** getting predicted in **Naive Bayes** but minority classes are getting **predicted well using RoBERTa**.
- **Minority classes prediction** improvement is the **biggest improvement** we achieved with this model.

Reasons of Improvement:

- The significant improvement in performance metrics and minority class predictions using the fine-tuned RoBERTa model on random over-sampled data for merged accident classes stems from RoBERTa's ability to capture complex linguistic patterns and context through pre-trained transformers, which Naive Bayes lacks as it relies on simplified word independence assumptions and lacks the ability to understand context.
- Fine-tuning RoBERTa on the accident description dataset allowed the model to align its pre-trained linguistic knowledge with the domain-specific features, significantly enhancing its performance, especially on nuanced minority class predictions.
- Merging accident classes reduced class overlap and improved class balance.
- Oversampling ensured better representation of minority classes, enabling RoBERTa to generalize effectively across all categories. Naive Bayes, on the other hand, is more sensitive to imbalanced distributions and struggles with such adjustments.

Implications:

- **Improved Decision-Making in Safety Management**
  The model's ability to classify accident severity levels accurately enables safety teams to prioritize responses to high-severity incidents. This helps in faster allocation of resources, reducing downtime, and preventing escalation of critical issues.
- **Enhanced Incident Reporting Efficiency**
  By automating the classification of accident descriptions, the solution reduces the manual effort required in categorizing incidents. This improves reporting consistency and ensures a standardized approach to assessing severity.
- **Focus on High-Risk Areas**
  With reliable predictions, businesses can identify trends in high-severity incidents, helping to design more targeted interventions like better training, updated safety protocols, or investment in protective equipment.
- **Proactive Risk Management**
  By analyzing prediction trends, businesses can proactively address safety concerns before incidents occur, fostering a safer work environment.
- **Regulatory Compliance**
  Automated, accurate incident categorization ensures adherence to safety reporting standards, reducing compliance risks.

Recommendations:

- **Adopt RoBERTa for Deployment with Oversampling**
  Based on the evaluation metrics and performance on minority class prediction, the RoBERTa model is the most suitable for deployment. Its ability to handle imbalanced data using Random Over Sampler ensures improved prediction accuracy for minority classes, critical for real-world applications.
- **Monitor Model in Real-World Scenarios**
  Implement continuous monitoring of the model's performance after deployment. Metrics like precision, recall, and F1-score for each severity class should be tracked to detect any drift in performance.
- **Integrate Domain Expertise**
  Use the model as a decision-support tool rather than a sole decision-maker. Incorporate feedback from safety experts to fine-tune the solution periodically, improving its relevance and reliability.
- **Scalability and Regular Updates**
  Ensure the model can scale to handle larger datasets or new accident description formats as the organization grows. Update the model with new data periodically to maintain accuracy and relevance.

Confidence in Recommendations

- **Performance Metrics Confidence**
  The chosen solution demonstrates strong performance metrics score(around 80%), including high accuracy and F1-score on test data, indicating high confidence in the model's ability to generalize effectively.
- **Business Context Alignment**
  The ability to predict minority classes accurately ensures that even less frequent but critical high-severity incidents are identified, aligning with business priorities in safety management.
- **Caveats**
  While the model shows promise, it is essential to recognize potential limitations such as reliance on historical data quality, sensitivity to distribution shifts, and interpretability challenges in transformer models.

Closing Reflections

The journey of predicting accident severity levels based on textual accident descriptions has been a comprehensive exploration of various machine learning and natural language processing techniques. Started with traditional classifiers along with NLP pre-processed data in 1st milestone, then in 2nd milestone started with traditional classifiers using BERT embeddings and evolving to fine-tuning transformer models for sequence classification, this process has provided valuable insights into the nuances of model selection, data preprocessing, and handling class imbalances.

Key Learnings

1. **Impact of Preprocessing**
    - Initial attempts revealed that traditional classifiers like Logistic Regression, SVM,Random Forest and Gradient Boosting performed better with raw accident descriptions rather than preprocessed text. This highlighted the importance of retaining contextual information in text for accurate representation when using embeddings like BERT.
2. **Advantages of Fine-Tuning Transformers**
    - Transitioning to fine-tuned transformer models like BERT, RoBERTa, DistilBERT, XLNet, and ALBERT demonstrated their superior ability to capture intricate language patterns. This approach provided better overall accuracy and robustness compared to static embeddings with traditional classifiers.
3. **Handling Overfitting**
    - Experiments with varying epochs revealed that increasing epochs led to overfitting, as seen in the widening gap between train and test metrics. Setting **metric_for_best_model** to "F1" and evaluating at every epoch provided a better mechanism to prevent overfitting and select the best model for generalization.
4. **Challenges with Imbalanced Data**
    - Techniques like Random Over Sampling and SMOTE initially posed challenges in improving minority class predictions, especially with transformers. However, merging the accident levels into three broader categories and applying random oversampling significantly improved model performance on minority classes.
5. **RoBERTa's Effectiveness**
    - Among all tested models, RoBERTa stood out due to its robustness, superior generalization on test data, and effectiveness in predicting minority classes after merging categories. This demonstrated the importance of model architecture and pre-training corpus in domain-specific tasks.

What Would Be Done Differently Next Time?

1. **Earlier Focus on Domain-Specific Insights**
    - The merging of accident levels into broader categories proved pivotal in achieving better performance. Next time, starting with a domain-informed analysis of class definitions and their real-world implications might streamline the process and improve results earlier.
2. **Targeted Class Balancing Approaches**
    - While Random Over Sampling worked well eventually, exploring transformer-specific data augmentation methods or other advanced techniques like Class-Balanced Loss or Focal Loss from the outset could improve minority class predictions without oversampling.
3. **Experimentation with Custom Tokenizers**
    - Using pre-trained tokenizers was effective, but experimenting with custom tokenizers trained on domain-specific accident descriptions might provide even better results by aligning the model vocabulary with the dataset.
4. **Greater Automation of Hyperparameter Tuning**

- A more structured approach using tools like Optuna or Hyperopt could automate hyperparameter tuning and potentially uncover better configurations faster than manual experimentation.

Final Reflection

This project has highlighted the complexities and rewards of applying state-of-the-art NLP models to real-world classification problems. The iterative process of model optimization, data adjustments, and evaluation provided deep insights into handling imbalanced data, avoiding overfitting, and achieving both strong overall metrics and effective minority class predictions.

By combining domain-specific adjustments with advanced machine learning techniques, the solution demonstrates how modern AI can enhance safety management processes and enable more effective decision-making. The experience gained here serves as a strong foundation for tackling similar classification tasks in other domains, paving the way for continuous improvement and innovation.

References:

https://discuss.ai.google.dev/t/does-text-preprocessing-or-cleaning-required-for-bert-model-or-others-one/29416

https://towardsdatascience.com/part-1-data-cleaning-does-bert-need-clean-data-6a50c9c6e9fd

https://huggingface.co/google-bert/bert-base-uncased

https://www.linkedin.com/advice/1/what-most-effective-text-classification-algorithms

https://www.quora.com/Whats-the-best-Machine-Learning-algorithm-to-use-for-text-classification-if-you-use-tf-idf-and-word-embeddings-as-your-text-features

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

https://huggingface.co/docs/transformers/en/model_doc/roberta