AIML ONLINE AUG 23 BATCH

NATURAL LANGUAGE PROCESSING
INDUSTRY SAFETY

CAPSTONE PROJECT - INTERIM REPORT

Submitted by
Ajay Vignesh A B
Aparna Chennamaneni
Ashutosh Kumar
Bharrat Mohan
Sandeep Naidu

in fulfillment of the requirements of 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

Date:
8-Sep-2024

Table of contents

Contents

Table of contents 2
I. MILESTONE 1 5
1.0 Objective of this project 5
1.1 Problem Statement 5
2.0 About Data 5
2.1 Exploring the Input data 5
2.2 Target column 7
2.3 Data shape 7
3.0 Objective 8
4.0 Data Preparation 8
4.1 Data Cleansing 8
4.2 Missing value treatment 8
4.3 Duplicate records 9
4.4 Data pre-processing 10
4.5 Data manipulation 10
5.0 EDA 11
5.1 Approach to EDA 11
5.2 Univariate analysis 11
5.3 Inferences – Univariate Analysis 14
5.4 Bivariate Analysis 15
5.5 – Time series analysis 18
5.6 Multivariate analysis 19
5.7 Statistical testing 20
5.8 EDA and pre-processing learnings 21
6.0 NLP Pre-processing 22
7.0 Vectorization 23
8.1 Label encoding 25
8.2 One hot encoding 26
8.3 Final Feature set creation 27

9.0 Train test split 27
10. PCA 29
11.0 Upsampling 30
11.1 Over sampling 30
11.2 SMOTE 31
12.0 Model Building 32
12.1 List of models 32
12.2 Basic combination 33
12.3 PCA + Upsampled combination 36
12.4 PCA + SMOTE Combination 37
12.5 Target as Potential Accident Level 39
13.0 Hyper parameter tuning 39
14.0 Best model 40
15.0 EDA and Classification models Summary 40
15.1 Data Cleansing: 40
15.2 EDA Analysis: 40
15.3 Model Performance: 41
II MILESTONE 2 42
16.0 Neural Network Classifiers 42
16.1 Design, Train, Test Neural Networks 42
16.2 Performance evaluation of NN Architectures 45
16.3 Hyper Parameter Tuning 47
16.4 Neural Networks – Conclusion 48
17.0 RNN and LSTM Networks 49
17.1 A Simple RNN Architecture 49
17.2 RNN Hyper Parameterization 52
17.3 LSTM 53
17.4 Interpretation from RNN and LSTM Models 54
18.0 Data Augmentation 55
18.1 LSTM on Augmented Data 57
18.2 Bi-Directional LSTM on Augmented Data 58
19.0 Inferences from Milestone 2 61
19.0 Pickel the best model 61

III Milestone 3 (Optional) 62
20.0 User Interface for prediction 62
21.0 Conclusion 64
21.1 Milestone 1 64
21.2 Milestone 2 64
21.3 Milestone 3 64

I. MILESTONE 1

1.0 Objective of this project

1.1 Problem Statement

The objective of this project is to design a ML/DL based chatbot utility for the Industrial safety domain/department of any industrial companies/organizations. There is an urgent need for industries/companies around the globe to understand why employees still suffer some injuries/accidents in plants. Sometimes, they even die in such an environment. The industrial safety department is an important function of any industries/companies with plants where the work environment can be dangerous, unsafe and prone to accidents such as fire, hazardous chemicals etc. They define policies and procedures to provide a safe and hazardous-free work environment for their employees and third-party contractors. This chatbot utility will be designed and developed with the intent of helping the professionals / employees/ third party-contractors to highlight the safety risk as per the incident description.

Chatbots gained popularity when NLP algorithms and models have improved over the past decade. One of the many applications of chatbot is to automate some of the functions of their call centre or L1 helpdesk teams. They are aimed at bringing in efficiencies and productivity improvements of a L1 helpdesk team where repetitive queries can be handled as part of first level touch point. Companies across the globe implement chatbot programs as part of their digital automation initiatives to have first level of resolution to their employees/contractors based on some of the common repeatedly occurring incidents/problems. For most of the incidents, they are more of a request or not knowing where the information is present. For such cases, Chatbot utilities are designed to give directions to the possible resolutions. For complex problems where human intervention will be required, they can be designed with an in-built call to action (CTA) function to direct the employees towards resolution.

In this problem statement case, we will design a chatbot utility to help the professionals highlight the safety risk based on some of the incident descriptions that the users provide. Based on the safety risk, they can potentially take the next course of action.

2.0 About Data

The data set comes from one of the biggest industries in Brazil and in the world. This dataset basically records accidents/incidents from 12 different plants taken in three different countries over a period of 18 months between 2016 and 2017. Every data record is an accident, or an incident occurred and reported and extracted in the form of a CSV file to us.

2.1 Exploring the Input data

The various columns that are captured as part of the data set is as below:

- Data: timestamp or time/date information

This is a date field with a timestamp. Most of the timestamps recorded are from the database and the time of the incident is not recorded in this timestamp. It is more of the recorded timestamp

- Countries: which country the accident occurred (anonymized)

Since three countries' data are recorded, country_01 02 and 03 are the three values. It's more of a categorical data/

- Local: the city where the manufacturing plant is located (anonymized)

12 locations or plants from which accident data is captured. Again, a categorical column

- Industry sector: which sector the plant belongs to

Industry sectors are dangerous and accident-prone ones like metals (metal processing or manufacturing ones) and mining where operations happen in dangerous and chemical hazardous mines. There is third value as others whichever is not fallen into metals and mining

- Accident level: from I to VI, it registers how severe was the accident (I means not severe but VI means very severe)

This is the original recorded accident level. One being "not severe" to 6 being the "highest severity" one. This is our target variable.

- Potential Accident Level: Depending on the Accident Level, the database also registers how severe the accident could have been (due to other factors involved in the accident)

What could have been the severity of the accident level. Again, one being "not severe" to 6 being the "highest severity" one. This can also be one of the target variables.

- Gender - if the person is male of female
- Employee or Third Party - if the injured person is an employee or a third party
- Critical Risk - some description of the risk involved in the accident
- Description

Detailed description of how the accident happened. This will be used for NLP pre-processing along with other parameters which predicts the possible accident level and accident levels.

```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 425 entries, 0 to 424
Data columns (total 11 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Unnamed: 0                425 non-null    int64
 1   Data                      425 non-null    datetime64[ns]
 2   Countries                 425 non-null    object
 3   Local                     425 non-null    object
 4   Industry Sector           425 non-null    object
 5   Accident Level            425 non-null    object
 6   Potential Accident Level  425 non-null    object
 7   Genre                     425 non-null    object
 8   Employee or Third Party   425 non-null    object
 9   Critical Risk             425 non-null    object
 10  Description               425 non-null    object
dtypes: datetime64[ns](1), int64(1), object(9)
memory usage: 36.6+ KB
```

Figure 1 - Data info

All the variables are of object or categorical type. Of them the "Data" variable is a datetime variable yet put in place in the dataframe as an object variable. Description is a free flow text whereas the remaining attributes like country, local, critical risk, Gender, Employee type and Accident level (potential included) are all categorical in nature.

2.2 Target column

The potential target variable can be Accident level or Potential Accident level. Using our EDA and pre-processing analysis, we will ascertain which parameters will help in predicting these closely. Since this column will be having multiple values, we will be employing multi classification ML models, later use deep learning models like RNN and finally employ LSTM models.

2.3 Data shape

The given dataset contains of 425 records and 11 columns.

```
print("Shape of Data Provided : \n")
print("Number of Columns : ",data.shape[1])
print("Number of Rows : ",data.shape[0])
```

```
Shape of Data Provided :

Number of Columns :  11
Number of Rows :  425
```

Figure 2- Data shape

Below is the output of first 5 rows of the dataset.

```
data.head()
```

	Unnamed: 0	Data	Countries	Local	Industry Sector	Accident Level	Potential Accident Level	Genre	Employee or Third Party	Critical Risk	Description
0	0	2016-01-01	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...
1	1	2016-01-02	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...
2	2	2016-01-06	Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...
3	3	2016-01-08	Country_01	Local_04	Mining	I	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 C...
4	4	2016-01-10	Country_01	Local_04	Mining	IV	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances t...

Figure 3 - Data.head

3.0 Objective

In this document, the following sections include the Exploratory data analysis (EDA) and NLP data analysis on the incident description columns. The EDA will help us find which parameters influence the target variable to a larger extent. We will also use visualization techniques to provide better insights into the various factors or parameters. The NLP analysis will help us in predicting the possible accident level or safety level based on the user's incident description. We will also analyze which model will help us in most accurately predicting the target variables.

4.0 Data Preparation

4.1 Data Cleansing

The first step is to do data cleansing – cleaning null values, duplicates and typo errors if any in the data or the column names.

- We have a column Unnamed: 0, which appears to be index column and so we can drop it.
- Rename following columns: 'Data', 'Countries', 'Genre', 'Employee or Third Party'.
- Data is date column, so let us rename it as Date
- Countries is renamed as Country
- Genre is Gender column
- Employee or Third Party is renamed as EmployeeType

```
# Rename 'Data', 'Countries', 'Genre', 'Employee or Third Party' columns in Data frame
data.rename(columns={'Data':'Date', 'Countries':'Country', 'Genre':'Gender', 'Employee or Third Party':'EmployeeType'}, inplace=True)
data.head()
```

	Date	Country	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	EmployeeType	Critical Risk	Description
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	2016-01-02	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...
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	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	2016-01-10	Country_01	Local_04	Mining	IV	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances t...

Figure 4 - Renaming columns

4.2 Missing value treatment

After data cleaning, now we have to check if there are missing values in any of the columns. We use data.isnull().sum() to check if there are missing values.

```
#check missing values in data
data.isnull().sum()
```

	0
Date	0
Country	0
Local	0
Industry Sector	0
Accident Level	0
Potential Accident Level	0
Gender	0
EmployeeType	0
Critical Risk	0
Description	0

dtype: int64

Figure 5 - Missing values

As seen in above, this dataset does not have any missing values.

4.3 Duplicate records

Now that we have completed data cleaning and missing values, let us check for duplicate records in the dataset.

```
#checking for duplicate rows
data.duplicated().sum()
```

```
7
```

There are 7 duplicate entries, let us remove them from the dataset

```
data.drop_duplicates(inplace=True)
data=data.reset_index(drop=True)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 10 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Date                      418 non-null    datetime64[ns]
 1   Country                   418 non-null    object
 2   Local                     418 non-null    object
 3   Industry Sector           418 non-null    object
 4   Accident Level            418 non-null    object
 5   Potential Accident Level  418 non-null    object
 6   Gender                    418 non-null    object
 7   EmployeeType              418 non-null    object
 8   Critical Risk             418 non-null    object
 9   Description               418 non-null    object
dtypes: datetime64[ns](1), object(9)
memory usage: 32.8+ KB
```

Figure 6 - Handling duplicates

We can see that the dataset has 7 repeated records which are of no use, so they are dropped from the dataset. After this, the shape of the dataset is (418, 10)

4.4 Data pre-processing

Going through the unique values of each column. Description is more free text, so will have all unique almost. Omitting the description column alone.

- We can see that there are records of accidents from 1/1/2016 to 9/7/2017 in every month. So, there are no outliers in the 'Date' column.
- There are only three country types.
- There are total of 12 Local cities where the manufacturing plant is located. They appear to be in sequence 1-12.
- There are only three Industry Sector types - Mining, Metals or Others. There are five Accident Level types which are in sequence.
- There are six Potential Accident Level types which are in sequence.
- Gender column also has only 2 values Male/Female, so there is no wrong input in this column as well.
- There are three Employee types in the provided data.
- There are quite a lot of Critical risk descriptions, and we don't see any outliers.

4.5 Data manipulation

We can split the date column into day, month, year and day of week. A new column “seasons” is also added based on the month field of the date column.

```
# function to create month variable into seasons
def seasons(x):
    if x in [9, 10, 11]:
        season = 'Spring'
    elif x in [12, 1, 2]:
        season = 'Summer'
    elif x in [3, 4, 5]:
        season = 'Autumn'
    elif x in [6, 7, 8]:
        season = 'Winter'
    return season
```

Figure 7 - Add Seasons column

We have now added 5 new columns to the dataset. We can drop the date column as it is no longer required.

```
data.head()
```

	Date	Country	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	EmployeeType	Critical Risk	Description	Year	Month	Day	WeekDay	Season
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...	2016	1	1	Friday	Summer
1	2016-01-02	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...	2016	1	2	Saturday	Summer
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...	2016	1	6	Wednesday	Summer
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...	2016	1	8	Friday	Summer
4	2016-01-10	Country_01	Local_04	Mining	IV	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances t...	2016	1	10	Sunday	Summer

Figure 8 - Data with new columns

5.0 EDA

This section comprises of EDA - Exploratory data analysis on categorical columns. In this, we will do univariate, bivariate and statistically significant hypothesis tests to determine which parameters drives our target variable prediction.
As earlier mentioned, we will have to employ a multi classification models for our prediction.

5.1 Approach to EDA

As part of EDA, we will do the following steps:

- Univariate analysis – to understand the distribution of data
- Bivariate analysis – to understand the relationships between columns
- Timeseries analysis
- Multivariate analysis
- Statistical tests – Hypothesis testing
- Concluding on EDA

From the dataset, we have 2 possible target columns, and the rest are Predictor columns.

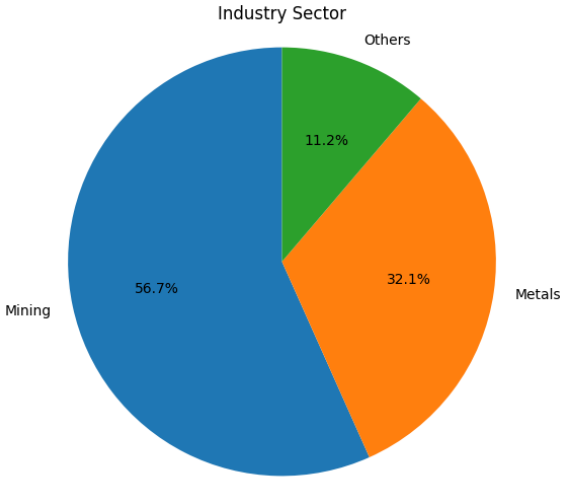
- Target variable: 'Accident Level' or 'Potential Accident Level'
- Predictors (Dependent variables): 'Day, 'Year', 'Month', 'Seasons', 'Country', 'Local', 'Industry Sector', 'Gender', 'EmployeeType', 'Critical Risk', 'Description'

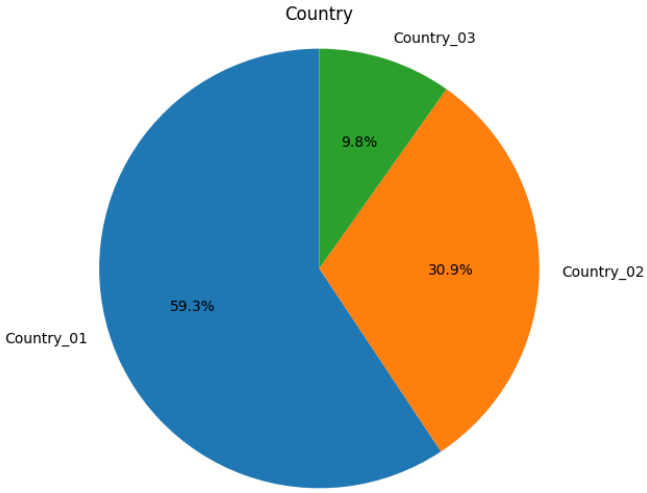
5.2 Univariate analysis

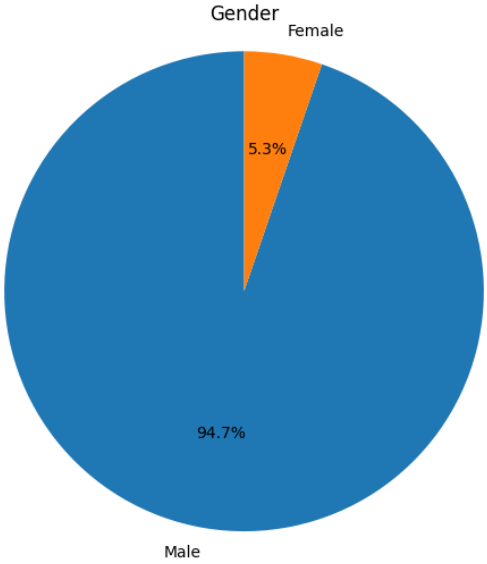
Univariate analysis explores each variable in a data set, separately. It looks at the range of values, as well as the central tendency of the values. It describes the pattern of response to the variable. It describes each variable on its own.
We plotted pie graphs to check distribution for some of the columns like Country, Industry sector, Gender etc. Function was defined for various univariate plots. Below is the code snippet for the same.

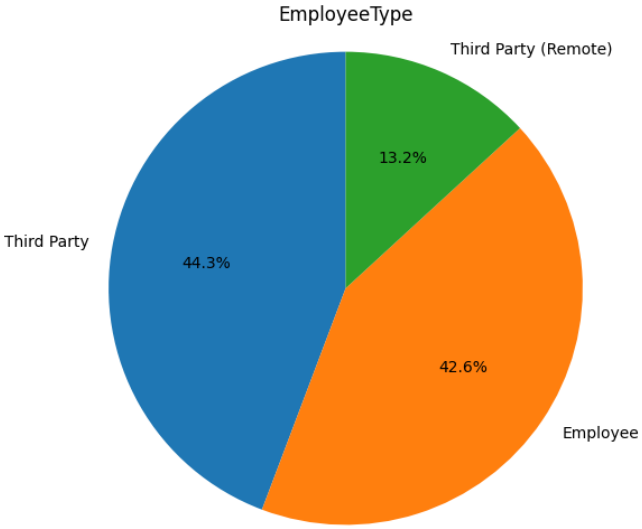
```
def plot(x, df, type, y=""):
  plt.figure(figsize=(6, 6))
  title = "Distribution of " + x
  plt.title(x)
  if y != "" and type == "bar":
    sns.countplot(x =x, data = df, hue =y)
  elif y == "" and type == "bar":
    sns.countplot(x =x, data = df)
  elif type == "pie":
    category_counts = df[x].value_counts()
    plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%', startangle=90)
    plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle
  plt.show()
```

Figure 9 - Univariate plot function


Industry Sector
Others
11.2%
32.1%
Metals
56.7%
Mining


Country
Country_03
9.8%
30.9%
Country_02
59.3%
Country_01


Gender
Female
5.3%
94.7%
Male


EmployeeType
Third Party (Remote)
13.2%
Third Party
44.3%
42.6%
Employee

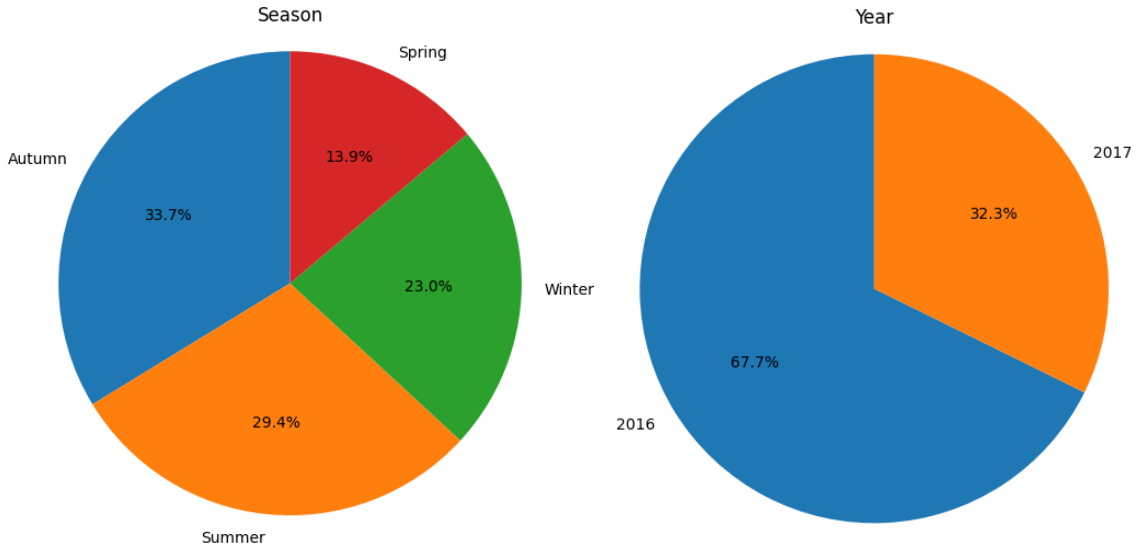


Figure 10 - Univariate plots

Below is the target variable distribution Accident and Potential accident level.

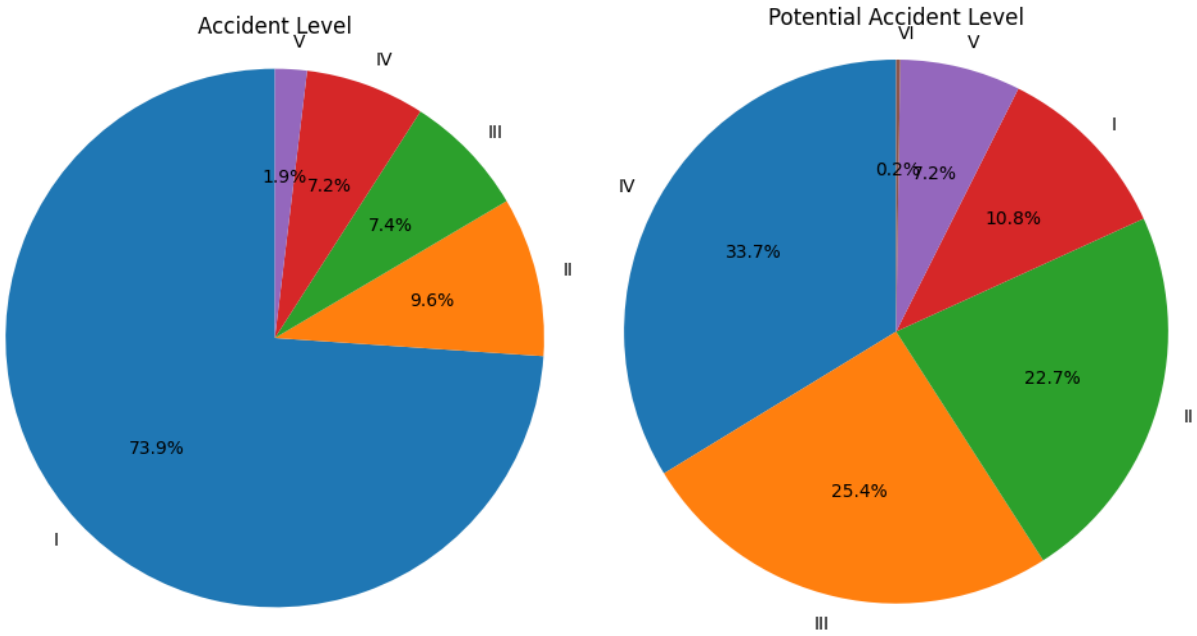


Figure 11 - Target column distribution

For columns Local and critical risk, we used bar plots.

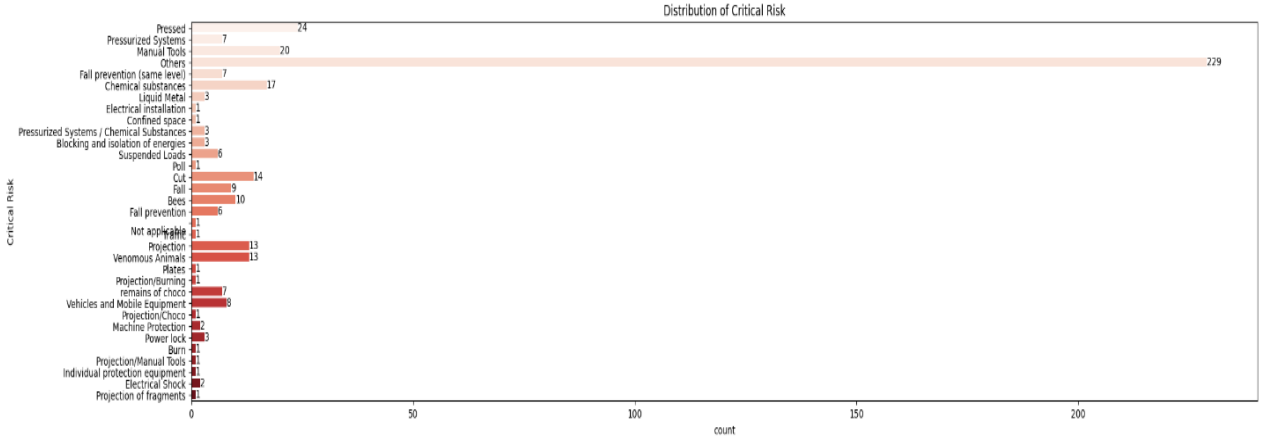


Figure 12 - Critical risk distribution

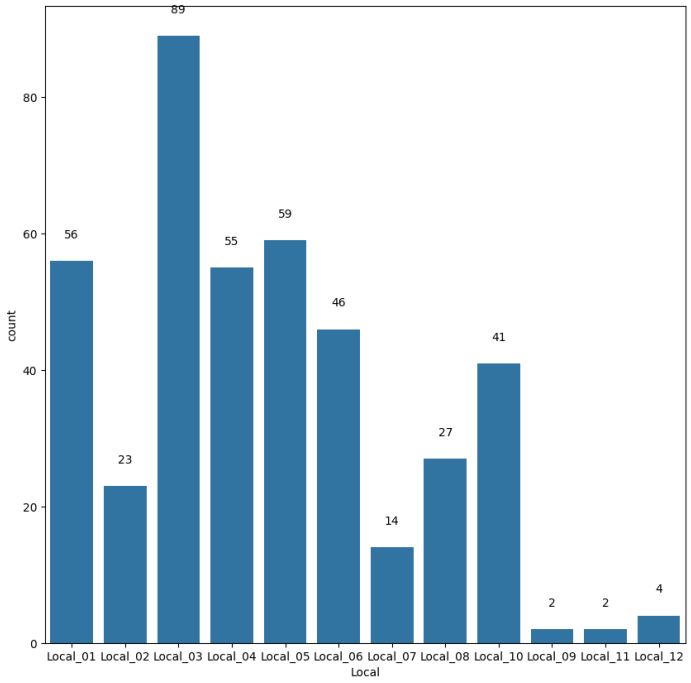


Figure 13 - Local distribution

5.3 Inferences – Univariate Analysis

From univariate analysis, we have below inferences:

- Country_01 has ~ 60% of the total accidents recorded followed by Country_2 (30%) and Country_03 (10%) with the least accidents.
- Sector wise accident plot shows that ~ 56% of the accidents are occurring in Mining industries and ~ 32% of accidents in Metal industries.
- Gender plot shows that ~ 95% of the people who met with accidents are Male.
- Third party employees have met with more accidents among all employee types, contributing to ~ 45% of the total accidents.

- This is followed by Employees who contributed ~ 42% to the total accidents.
- ~ 65% of the accidents have taken place in March-August time period of the year.
- More accidents happen in Autumn season (35%) followed by Summer (~ 30%). Number of accidents increased during the middle of the week (Tue-Thur) and then decline.
- Most accidents happened in Location 3, followed by Local_5, local_1 and local_4.

5.4 Bivariate Analysis

Bivariate analysis is one of the statistical analyses where two variables are observed. One variable here is dependent while the other is independent. It is mainly used for the purpose of determining the empirical relationship between them.
Some of the plots from Bivariate analysis and inferences are mentioned below:

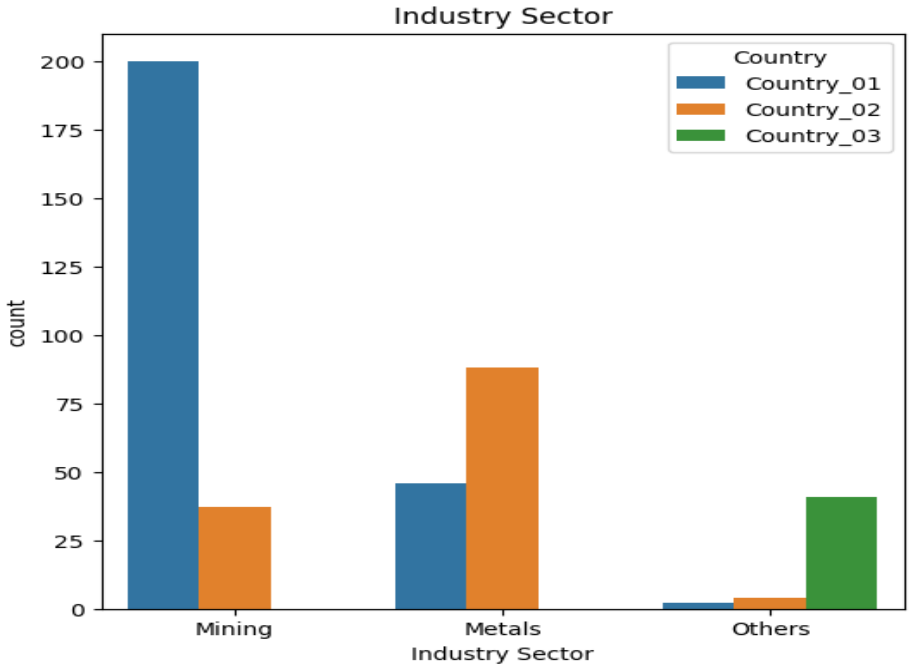
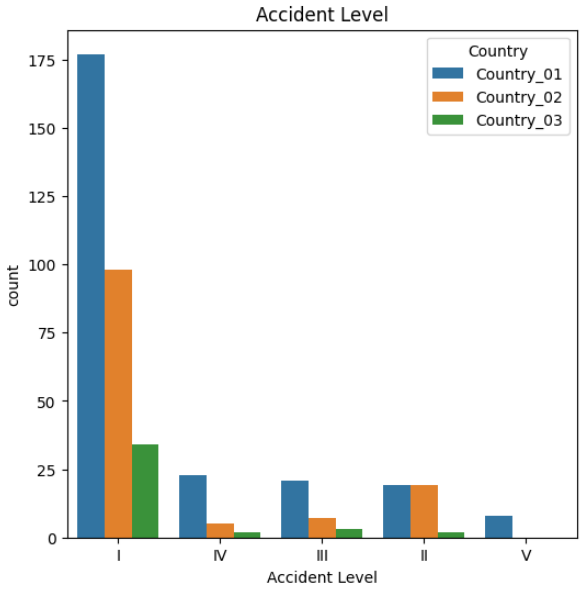


Figure 14 - Country vs Industry distribution

- Metals and Mining industry sector plants are not available in Country_03.
- Distribution of industry sector differs significantly in each country.
- When we look at accidents w.r.t countries, Country one seems to have the most number of accidents.



Accident Level	Country_01	Country_02	Country_03
I	177	98	34
II	19	19	2
III	21	7	3
IV	23	5	2
V	8	0	0

Figure 15 - Accident level vs Country

Below plot and crosstab clearly shows this.

Below is plot of accident level across different industries. Accident level I has more data, the given dataset is imbalanced - it might be a challenge to train model on predicting other accident levels.

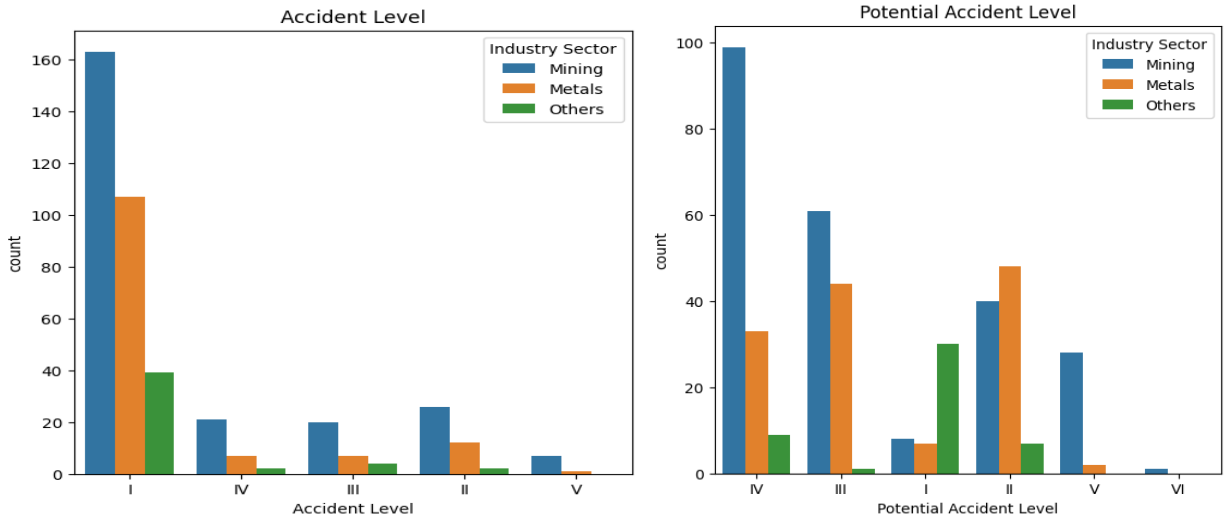


Figure 16 – Target col vs Industry sector

More accidents of higher severity are from Mining industries, followed by Metal Industries. Accident Level – I has more data, which causes data imbalance.

Now let us look at Accidents and Gender relations.

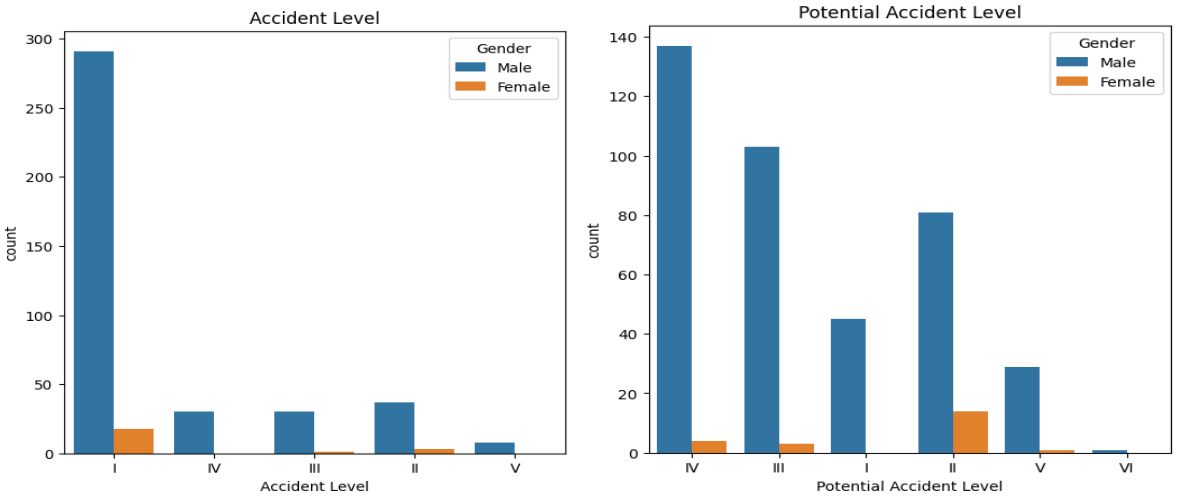


Figure 17 - Accidents vs Gender

Male are involved in more accidents across levels, this is expected given the dataset has highly imbalanced Gender ratio.
Below plot shows relationship of Local column with Accident level.

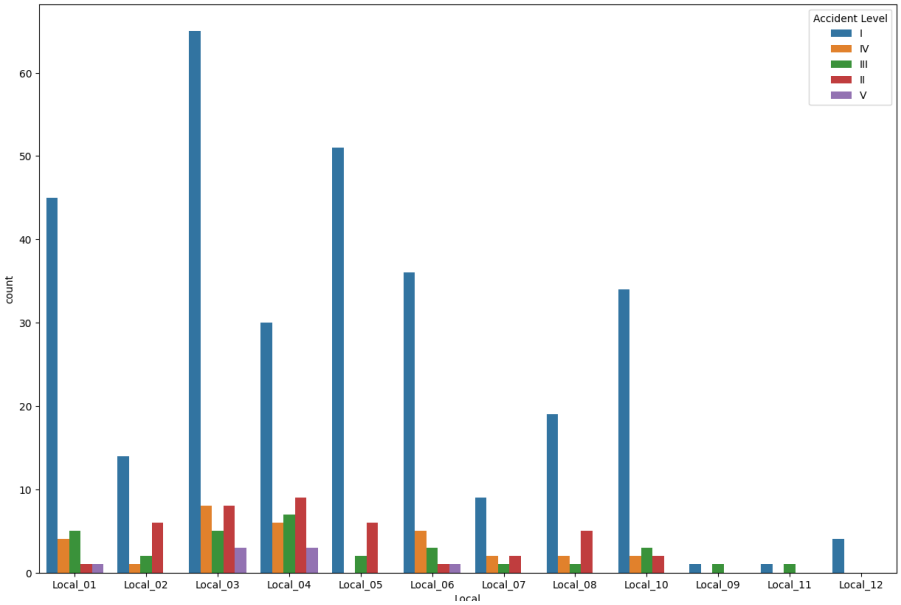
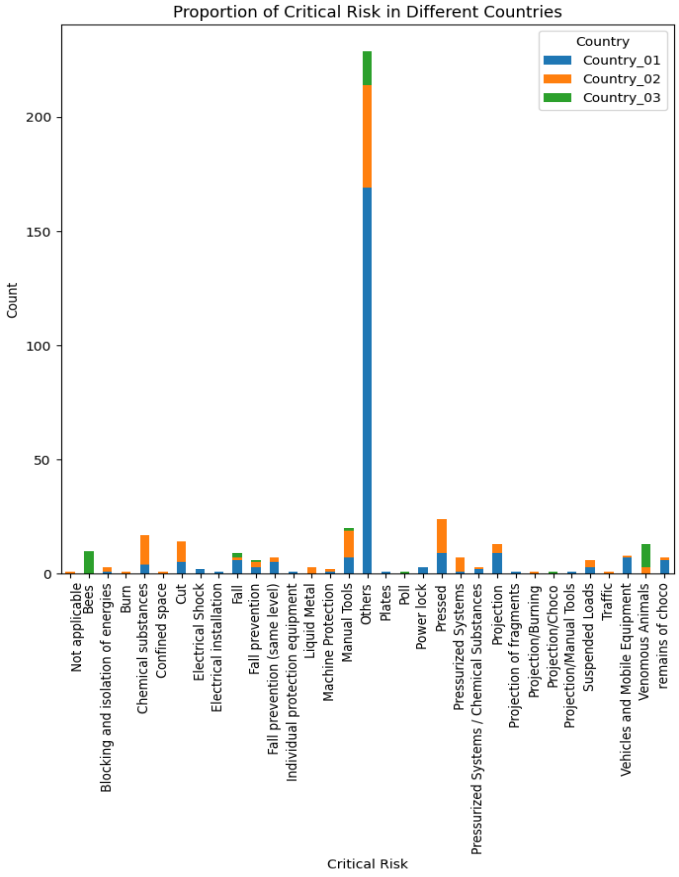


Figure 18 - Accident level vs Local

This plot shows critical risk distribution across countries.



Local	Local_01	Local_02	Local_03	Local_04	Local_05	Local_06	Local_07	Local_08	Local_09	Local_10	Local_11	Local_12
Country												
Country_01	56	0	89	55	0	46	0	0	0	0	2	0
Country_02	0	23	0	0	59	0	14	27	2	0	0	4
Country_03	0	0	0	0	0	0	0	0	0	41	0	0

Figure 19 - Country and Local

From the above table it is noticed that Local column values have unique relation with Country column values. That is, every Local is related to only one country.

- As both Local and Countries gives the same info, we can remove Country column from dataframe.

5.5 – Time series analysis

- There is no data in 2017 from the month of August (8).
- So, this will not give any relevant information as data is missing to compare data 2017 to 2016.

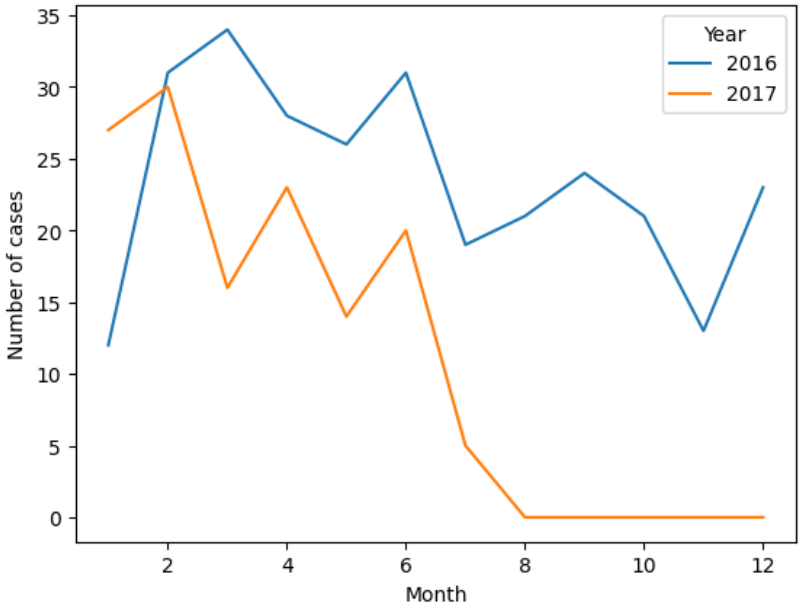


Figure 20 - Timeseries analysis

So, we can drop year column from the dataset as it does not provide any distinct contribution to accident predictions.

5.6 Multivariate analysis

We used label encoder to plot correlation matrix for the data.

```
# Calculate the correlation matrix
le = LabelEncoder()
df_encoder = data.apply(le.fit_transform)

plt.figure(figsize=(12,12))
plt.title('Correlation_Matrix', fontsize=20)
sns.heatmap(df_encoder.corr(), square=True, cmap='coolwarm', annot=True, linewidth=0.2);
```

Figure 21- Correlation matrix code

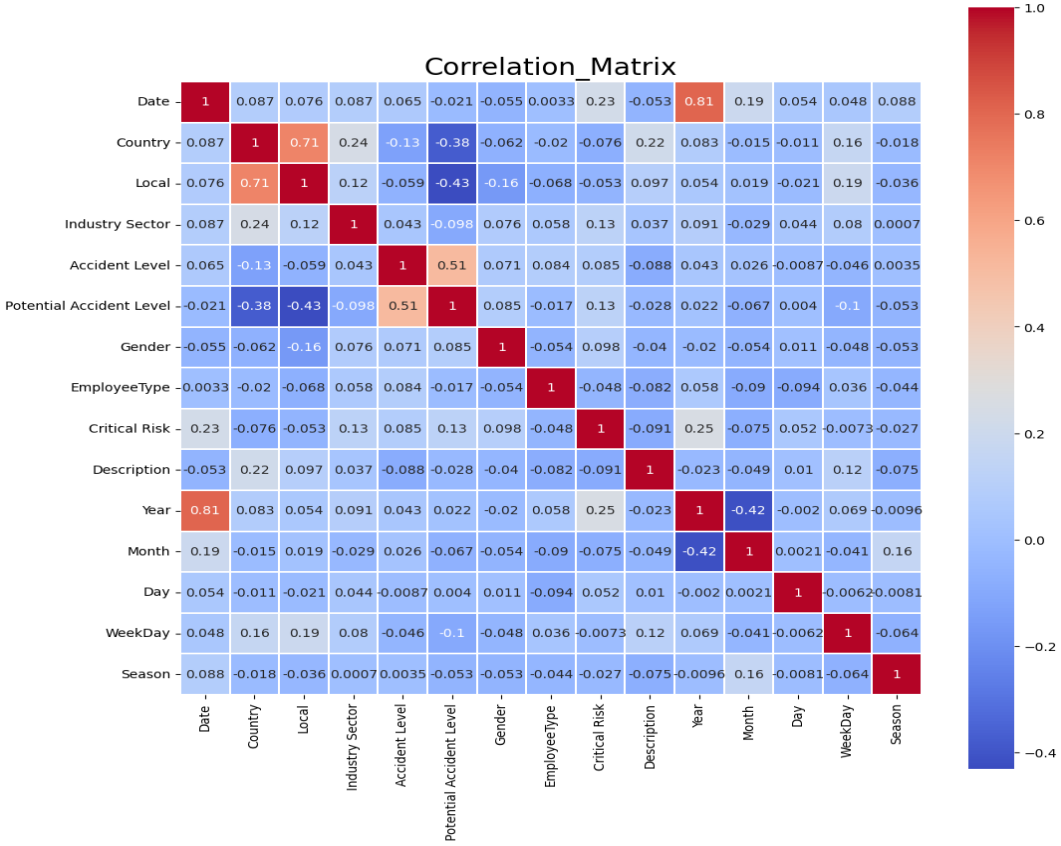


Figure 22 - Correlation matrix

- From the above heap map, Countries and Local are highly correlated with value 0.71
- Next, Accident level and Potential Accident Level are correlated with value .51
    - This in alignment with our analysis that Country and Local columns are related, and we can drop anyone.

5.7 Statistical testing

We ran chi-square test to ascertain some of our above inferences on relationships among dependent variables.

- Distribution of accident level across countries.

    H0: The proportions of accident levels do not differ in different countries.
    Ha: The proportions of accident levels differ in different countries.

    Chi-squared Statistic: 16.539177604348506
    p-value: 0.03528174940919624
    Reject Null Hypothesis? True

We reject the null hypothesis. We have enough evidence to prove that the accident level distribution differs significantly across countries.

- Relationship between accident and potential accident levels

  H0: Accident Level and Potential accident level are not related.
  Ha: Accident Level and Potential accident level are related.

  Chi-squared Statistic: 249.45260461536674
  p-value: 1.4728348366116975e-41
  degree of freedom 20
  Reject Null Hypothesis?: True

  We reject the null hypothesis. We have enough evidence to prove that the accident level and potential accident level are dependent.

5.8 EDA and pre-processing learnings

As per our EDA, we can drop below columns:

- Date – have added new columns splitting this
- Year – No relationship with target. (insufficient data)
- Day – Not adding much value.
- Country – Highly correlated with Local

```
#Drop unnecessary columns as per EDA inferences
df = data.drop(columns=['Country', 'Date', 'Year', 'Day'])
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   Local                   418 non-null    object
 1   Industry Sector         418 non-null    object
 2   Accident Level          418 non-null    object
 3   Potential Accident Level  418 non-null    object
 4   Gender                  418 non-null    object
 5   EmployeeType            418 non-null    object
 6   Critical Risk           418 non-null    object
 7   Description             418 non-null    object
 8   Month                   418 non-null    int64 
 9   WeekDay                 418 non-null    object
 10  Season                  418 non-null    object
dtypes: int64(1), object(10)
memory usage: 36.0+ KB
```

Figure 23 - EDA learnings

6.0 NLP Pre-processing

Text Preprocessing is the most essential step for any NLP model. How well the raw data has been cleaned and preprocessed plays a major role in the performance of the model.
The various preprocessing steps that are involved are:

1. Lower Casing
2. Tokenization
3. Punctuation Mark Removal
4. Stop Word Removal
5. Stemming/Lemmatization

In addition to this, we performed

- Defining the regex pattern to match non-alphanumeric characters.
- Removing extra whitespaces

```
# defining a function to remove special characters
def remove_special_characters(text):
    # Defining the regex pattern to match non-alphanumeric characters
    pattern = '[^A-Za-z0-9]+'

    # Finding the specified pattern and replacing non-alphanumeric characters with a blank string
    new_text = ''.join(re.sub(pattern, ' ', text))

    return new_text
```

```
# Defining a function to remove stop words (like pronouns) using the NLTK library
def remove_stopwords(text):
    # Split text into separate words
    words = text.split()

    # Removing English language stopwords
    new_text = ' '.join([word for word in words if word not in stopwords.words('english')])

    return new_text
```

Figure 24 - NLP pre-processing

Below is the output of initial description column and final text column after all NLP pre-processing.

	Description
0	While removing the drill rod of the Jumbo 08 for maintenance, the supervisor proceeds to loosen the support of the intermediate centralizer to facilitate the removal, seeing this the mechanic supports one end on the drill of the equipment to pull with both hands the bar and accelerate the removal from this, at this moment the bar slides from its point of support and tightens the fingers of the mechanic between the drilling bar and the beam of the jumbo.
1	During the activation of a sodium sulphide pump, the piping was uncoupled and the sulfide solution was designed in the area to reach the maid. Immediately she made use of the emergency shower and was directed to the ambulatory doctor and later to the hospital. Note: of sulphide solution = 48 grams / liter.
2	In the sub-station MILPO located at level +170 when the collaborator was doing the excavation work with a pick (hand tool), hitting a rock with the flat part of the beak, it bounces off hitting the steel tip of the safety shoe and then the metatarsal area of the left foot of the collaborator causing the injury.

Figure 25 - Initial Description column

	final_cleaned_text
0	removing drill rod jumbo maintenance supervisor proceeds loosen support intermediate centralizer facilitate removal seeing mechanic support one end drill equipment pull hand bar accelerate removal moment bar slide point support tightens finger mechanic drilling bar beam jumbo
1	activation sodium sulphide pump piping uncoupled sulfide solution designed area reach maid immediately made use emergency shower directed ambulatory doctor later hospital note sulphide solution gram liter
2	sub station milpo located level collaborator excavation work pick hand tool hitting rock flat part beak bounce hitting steel tip safety shoe metatarsal area left foot collaborator causing injury

Figure 26 - Description after NLP Preprocessing

7.0 Vectorization

Vectorization in NLP is used to convert raw text data into a numerical format that machine learning algorithms can understand and process. This is the most important step in NLP as any machine learning model only understands numbers and not text.

We used below vectorization techniques.

- Word2Vec Embeddings:
  Word2vec is a technique in natural language processing (NLP) for obtaining vector representations of words. These vectors capture information about the meaning of the word based on the surrounding words. The word2vec algorithm estimates these representations by modeling text in a large corpus.

```
# creating an instance of Word2Vec
model_W2V = Word2Vec(words_list, vector_size = 100, min_count = 1, window=5, workers = 6)
# Checking the size of the vocabulary
print("Length of the vocabulary is", len(list(model_W2V.wv.key_to_index)))
```

```
Length of the vocabulary is 2800
```

Figure 27 - Word2vec

Below is sample feature data after applying word2vec on the final cleaned text column. We used word2vec with a vector size of 100.

```
# creating a dataframe of the vectorized documents
vec_size = 100
df_word2vec = pd.DataFrame(df_nlp['final_cleaned_text'].apply(average_vectorizer_Word2Vec).tolist(), columns=['Feature '+str(i) for i in range(vec_size)])
df_word2vec
```

	Feature 0	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	Feature 6	Feature 7	Feature 8	Feature 9	Feature 10	Feature 11	Feature 12	Feature 13	Feature 14	Feature 15	Feature 16	Feature 17	Feature 18	Feature 19	Feature 20	Feat
0	-0.010908	0.020542	0.005322	-0.002285	0.002014	-0.028171	0.006765	0.036178	-0.005292	-0.013763	-0.014796	-0.027407	-0.005454	0.002419	0.002119	-0.012773	-1.904702e-04	-0.029173	-0.002889	-0.031867	0.011114	0.009
1	-0.004618	0.009454	0.000712	-0.000809	-0.000832	-0.010238	0.001891	0.015835	0.001352	-0.005379	-0.005163	-0.011549	-0.002228	0.001359	0.000456	-0.004555	2.234984e-03	-0.010851	-0.003108	-0.012319	0.006174	0.003
2	-0.012837	0.025287	0.004463	0.000483	0.005317	-0.030123	0.008646	0.039258	-0.004065	-0.016197	-0.015675	-0.028403	-0.004700	0.003225	0.003607	-0.014609	4.308191e-04	-0.029053	-0.005991	-0.033824	0.010437	0.008

Figure 28 - Feature data word2vec

- GLoVe Embeddings:
  GLoVe is a pre-trained vector embedding based on English vocabulary. It has a corpus of 400000 words.

```
from gensim.scripts.glove2word2vec import glove2word2vec
glove_input_file = '/content/drive/My Drive/Colab Notebooks/glove.6B.100d.txt'
word2vec_output_file = 'glove.6B.100d.txt.word2vec'
glove2word2vec(glove_input_file, word2vec_output_file)
```

```
(400000, 100)
```

```
from gensim.models import KeyedVectors
# load the Stanford GloVe model
filename = 'glove.6B.100d.txt.word2vec'
glove_model = KeyedVectors.load_word2vec_format(filename, binary=False)
```

```
# Checking the size of the vocabulary
print("Length of the vocabulary is", len(glove_model.index_to_key))
```

```
Length of the vocabulary is 400000
```

Figure 29 - Glove embedding

We used same vector size of 100 for glove as well.

```
# Creating a dataframe of the vectorized documents
df_glove = pd.DataFrame(df_nlp['final_cleaned_text'].apply(average_vectorizer_GloVe).tolist(), columns=['Feature '+str(i) for i in range(vec_size)])
df_glove
```

	Feature 0	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	Feature 6	Feature 7	Feature 8	Feature 9	Feature 10	Feature 11	Feature 12	Feature 13	Feature 14	Feature 15	Feature 16	Feature 17	Feature 18	Feature 19	Feature 20
0	-0.104074	0.085043	-0.006874	-0.117856	-0.047997	-0.262621	-0.220739	0.233383	-0.020247	0.153795	0.030064	-0.066967	0.075305	0.021312	0.047142	-0.096180	-0.053975	-0.025514	-0.025976	-0.038412	0.077116
1	-0.260657	0.222551	-0.037376	0.059230	0.048171	0.049342	-0.025612	0.298147	0.088059	0.113739	0.119546	-0.066295	-0.057286	0.308257	0.196786	-0.181216	-0.042313	-0.024512	-0.180914	-0.190796	-0.032796
2	-0.069923	0.097646	0.039634	-0.173443	-0.034290	-0.148609	-0.009568	0.094570	-0.053122	0.140734	0.134343	-0.212128	0.363301	0.110846	0.188240	0.000159	0.047959	-0.125856	-0.104772	-0.050605	0.337794

Figure 30 - Glove feature vector

- TF-IDF Embeddings
  TF-IDF is a statistical measure used to determine the mathematical significance of words in documents. The vectorization process is like One Hot Encoding. Alternatively, the value corresponding to the word is assigned a TF-IDF value instead of 1. The TF-IDF value is obtained by multiplying the TF and IDF values.

```
vec_tfidf = TfidfVectorizer()

corpus=df_nlp['final_cleaned_text'].values
# Fit and transform the corpus
tfidf = vec_tfidf.fit_transform(corpus)

# Get the feature names (words)
features = vec_tfidf.get_feature_names_out()

# Print the feature names
print("Feature names:", features)

# Convert TF-IDF matrix to DataFrame
df_tfidf = pd.DataFrame(tfidf.toarray(), columns=features)
df_tfidf
```

Figure 31 - TF IDF Vectorization

8.0 Categorical column Encoding

8.1 Label encoding

Label Encoding is a technique that is used to convert categorical columns into numerical ones so that they can be fitted by machine learning models which only take numerical data.

We will apply label encoding to Target columns Accident Level and Potential Accident Level.

In dependent columns, Season and Weekday will also be using label encoding.

```
#Since Weekday, Seasons, Accident level and Potential Accident level are ordinal, we will apply Label Encoding on these two columns
le = LabelEncoder()
df_nlp_cln['Accident Level'] = le.fit_transform(df_nlp_cln['Accident Level']).astype(np.int8)
df_nlp_cln['Potential Accident Level'] = le.fit_transform(df_nlp_cln['Potential Accident Level']).astype(np.int8)
df_nlp_cln['Season'] = le.fit_transform(df_nlp_cln['Season']).astype(np.int8)
df_nlp_cln['WeekDay'] = le.fit_transform(df_nlp_cln['WeekDay']).astype(np.int8)

df_nlp_cln.head(5)
```

	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	EmployeeType	Critical Risk	Month	WeekDay	Season	final_cleaned_text
0	Local_01	Mining	0	3	Male	Third Party	Pressed	1	0	2	removing drill rod jumbo maintenance supervisor proceeds loosen support intermediate centralizer facilitate removal seeing mechanic support one end drill equipment pull hand bar accelerate removal moment bar slide point support tightens finger mechanic drilling bar beam jumbo
1	Local_02	Mining	0	3	Male	Employee	Pressurized Systems	1	2	2	activation sodium sulphide pump piping uncoupled sulfide solution designed area reach maid immediately made use emergency shower directed ambulatory doctor later hospital note sulphide solution gram liter
2	Local_03	Mining	0	2	Male	Third Party (Remote)	Manual Tools	1	6	2	sub station milpo located level collaborator excavation work pick hand tool hitting rock flat part beak bounce hitting steel tip safety shoe metatarsal area left foot collaborator causing injury
3	Local_04	Mining	0	0	Male	Third Party	Others	1	0	2	approximately nv cx ob personnel begin task unlocking soquet bolt bhb machine penultimate bolt identified hexagonal head worn proceeding mr crist bal auxiliary assistant climb platform exert pressure hand dado key prevent coming bolt moment two collaborator rotate lever anti clockwise direction leaving key bolt hitting palm left hand causing injury
4	Local_04	Mining	3	3	Male	Third Party	Others	1	3	2	approximately circumstance mechanic anthony group leader eduardo eric fern ndez injured three company impromec performed removal pulley motor pump zaf marcy cm length cm weight kg locked proceed heating pulley loosen come fall distance meter high hit instep right foot worker causing injury described

Figure 32 - Label encoding

8.2 One hot encoding

One hot encoding is a technique that we use to represent categorical variables as numerical values in a machine learning model.
For columns Local, Gender, Industry sector, EmployeeType and Critical risk we will use one hot encoding.

```
#Applying One Hot Encoding using dummy values approach on other columns
Local_dummies = pd.get_dummies(df_nlp_cln['Local'], columns=["Local"], drop_first=True).astype(np.int8)
Gender_dummies = pd.get_dummies(df_nlp_cln['Gender'], columns=["Gender"], drop_first=True).astype(np.int8)
Industry_dummies = pd.get_dummies(df_nlp_cln['Industry Sector'], columns=['Industry Sector'], prefix='IS', drop_first=True).astype(np.int8)
EmpType_dummies = pd.get_dummies(df_nlp_cln['EmployeeType'], columns=['EmployeeType'], prefix='EmpType', drop_first=True).astype(np.int8)
CR_dummies = pd.get_dummies(df_nlp_cln['Critical Risk'], columns=['Critical Risk'], prefix='CR', drop_first=True).astype(np.int8)
```

Figure 33 - One hot encoding

```
#Merge the above dataframe with the original dataframe
df_nlp_cln= df_nlp_cln.join(Local_dummies.reset_index(drop=True)).join(Gender_dummies.reset_index(drop=True)).join(Industry_dummies.reset_index(drop=True)).join(EmpType_dummies.reset_index(drop=True)).join(CR_dummies.r
df_nlp_cln.head()
```

	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	EmployeeType	Critical Risk	Month	WeekDay	Season	final_cleaned_text	Local_02	Local_03	Local_04	Local_05	Local_06	Local_07	Local_08	Local_09	Local_10	Local_11
0	Local_01	Mining	0	3	Male	Third Party	Pressed	1	0	2	removing drill rod jumbo maintenance supervisor proceeds loosen support intermediate centralizer facilitate removal seeing mechanic support one end drill equipment pull hand bar accelerate removal moment bar slide point support tightens finger mechanic drilling bar beam jumbo	0	0	0	0	0	0	0	0	0	0

Figure 34 - df after one hot encoding

8.3 Final Feature set creation

After encoding, let us drop the original columns.

Dataframe column clean-up: Final Dataframe

```
#deleting original columns that were one hot encoded
df_nlp_cln.drop(['Local', 'Gender', 'Industry Sector', 'EmployeeType', 'Critical Risk','final_cleaned_text'], axis=1, inplace=True)
df_nlp_cln.head(1)
```

Now let us merge the embedding feature data frame with the pre-processed and categorically encoded data frame. The merged data frame will be huge in terms of no. of columns as we are merging Feature embedding vector (of size 100 columns) with label/one hot encoded data frame. Later we will be using PCA to see if we can capture maximum variance in this data with minimal dimensions.

```
#Merging the data with Glove dataframe
feature_df_final_glove = df_nlp_cln.join(pd.DataFrame(df_glove).reset_index(drop=True))
feature_df_final_glove.head()
```

	Accident Level	Potential Accident Level	Month	WeekDay	Season	Local_02	Local_03	Local_04	Local_05	Local_06	Local_07	Local_08	Local_09	Local_10	Local_11	Local_12	Male	IS_Mining	IS_Others	EmpType_Third Party	EmpType_Third Party (Remote)	CR_Bees
0	0	3	1	0	2	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0
1	0	3	1	2	2	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
2	0	2	1	6	2	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0
3	0	0	1	0	2	0	0	1	0	0	0	0	0	0	0	0	1	1	0	1	0	0
4	3	3	1	3	2	0	0	1	0	0	0	0	0	0	0	0	1	1	0	1	0	0

Figure 35- Glove final feature set

```
#Merging the data with word2vec dataframe
feature_df_final_word2vec = df_nlp_cln.join(pd.DataFrame(df_word2vec).reset_index(drop=True))
feature_df_final_word2vec.head()
```

CR_Projection/Manual Tools	CR_Suspended Loads	CR_Traffic	CR_Vehicles and Mobile Equipment	CR_Venomous Animals	CR_remains of choco	Feature 0	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	Feature 6	Feature 7	Feature 8	Feature 9	Feature 10	Feature 11	Feature 12	Feature 13
0	0	0	0	0	0	-0.010908	0.020542	0.005322	-0.002285	0.002014	-0.028171	0.006765	0.036178	-0.005292	-0.013763	-0.014796	-0.027407	-0.005454	0.002419
0	0	0	0	0	0	-0.004618	0.009454	0.000712	-0.000809	-0.000832	-0.010238	0.001891	0.015835	0.001352	-0.005379	-0.005163	-0.011549	-0.002228	0.001359
0	0	0	0	0	0	-0.012837	0.025287	0.004463	0.000483	0.005317	-0.030123	0.008646	0.039258	-0.004065	-0.016197	-0.015675	-0.028403	-0.004700	0.003225
0	0	0	0	0	0	-0.011754	0.018638	0.003108	0.001741	0.002967	-0.024150	0.006575	0.030840	-0.003732	-0.011889	-0.011863	-0.023299	-0.004878	0.003463
0	0	0	0	0	0	-0.011858	0.019417	0.004082	0.000197	0.002919	-0.025326	0.007819	0.031011	-0.002304	-0.010852	-0.011856	-0.025193	-0.003049	0.002967

Figure 36 - Word2vec final feature set

```
#Merging the data with TFIDF dataframe
feature_df_final_tfidf = df_nlp_cln.join(pd.DataFrame(df_tfidf).reset_index(drop=True))
feature_df_final_tfidf.head()
```

CR_Pressurized Systems / Chemical Substances	CR_Projection	CR_Projection of fragments	CR_Projection/Burning	CR_Projection/Choco	CR_Projection/Manual Tools	CR_Suspended Loads	CR_Traffic	CR_Vehicles and Mobile Equipment	CR_Venomous Animals	CR_remains of choco	abb	abdomen	able	abratech	abrupt	abruptly
0	0	0	0	0	0	0	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0
0	0	0	0	0	0	0	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0
0	0	0	0	0	0	0	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0
0	0	0	0	0	0	0	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0
0	0	0	0	0	0	0	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 37 - TF IDF Final feature set

9.0 Train test split

Split the data into X, y and later split into X_train, X_test, y_train and y_test using test_train_split. This will be done for all 3 final feature set shown above.
We have created a function split() to do this.

```
X_w2v = df_w2v_import.drop(['Accident Level','Potential Accident Level'], axis = 1)
X_glove = df_glove_import.drop(['Accident Level','Potential Accident Level'], axis = 1)
X_tfidf = df_tfidf_import.drop(['Accident Level','Potential Accident Level'], axis = 1)
y_w2v = df_w2v_import['Accident Level']
y_glove= df_glove_import['Accident Level']
y_tfidf= df_tfidf_import['Accident Level']

#for potential accident level as target
y_w2v_pot = df_w2v_import['Potential Accident Level']
y_glove_pot= df_glove_import['Potential Accident Level']
y_tfidf_pot= df_tfidf_import['Potential Accident Level']
```

Figure 38 - X, y split

Split function to split the data:

Splitting data

```
def split(X,y):
    # Initial split into training (80%) and testing (20%)
    if np.min(np.bincount(y)) < 2:
    #Remove stratification if y has single sample - required in case of potential accident level V
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
    else:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
    return X_train,X_test,y_train,y_test
```

```
#Splitting the dataset
X_train_w2v,X_test_w2v,y_train_w2v,y_test_w2v=split(X_w2v,y_w2v)
X_train_glove,X_test_glove,y_train_glove,y_test_glove=split(X_glove,y_glove)
X_train_tfidf,X_test_tfidf,y_train_tfidf,y_test_tfidf=split(X_tfidf,y_tfidf)
```

```
print(X_train_w2v.shape, X_test_w2v.shape, y_train_w2v.shape, y_test_w2v.shape)
print(X_train_glove.shape, X_test_glove.shape, y_train_glove.shape, y_test_glove.shape)
print(X_train_tfidf.shape, X_test_tfidf.shape, y_train_tfidf.shape, y_test_tfidf.shape)
```

```
(334, 151) (84, 151) (334,) (84,)
(334, 151) (84, 151) (334,) (84,)
(334, 2838) (84, 2838) (334,) (84,)
```

Figure 39 - train test split

10. PCA

Principal component analysis, or PCA, is a dimensionality reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

We must standardize the data before applying PCA. Below 2 functions are defined for standardization and PCA.(as our data in label encoded/one-hot encoded, scaling was not require)

```
#Define Standardization of data
def standard_scaler(X_train, X_test):
    scaler = StandardScaler().set_output(transform="pandas")
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    return X_train_scaled, X_test_scaled

# Define PCA dimensiom reduction
def pca_sklearn(X_train, X_test, n_components):
    pca = PCA(n_components=n_components).set_output(transform="pandas")
    X_train_pca = pca.fit_transform(X_train)
    X_test_pca = pca.transform(X_test)
    return X_train_pca, X_test_pca
```

Figure 40 - Standardization and PCA functions

Using PCA, we are able to cover up to 95% of variance in the data with ~49 columns.

```
#PCA to extract columns that capture 95% of data
cov_matrix = np.cov(X.T)
eigenvalues, eigenvectors, = np.linalg.eig(cov_matrix)

eig_pairs = [(eigenvalues[index], eigenvectors[:,index]) for index in range(len(eigenvalues))]

# Sort the (eigenvalue, eigenvector) pairs from lowest to highest with respect to eigenvalue
eig_pairs.sort()
eig_pairs.reverse()    # reverses the sorted pairs from increasing value of eigenvalue to lowest

# Extract the descending ordered eigenvalues and eigenvectors
eigvalues_sort = [eig_pairs[index][0] for index in range(len(eigenvalues))]
eigvectors_sort = [eig_pairs[index][1] for index in range(len(eigenvalues))]
```

```
Shape of X_train after PCA is: (340, 49)
Shape of X_test after PCA is:  (85, 49)
Shape of y_train is: (340,)
Shape of y_test is: (85,)
```

Figure 41 – PCA

11.0 Upsampling

Upsampling is a technique used to address class imbalance in the dataset, particularly in classification tasks. Class imbalance occurs when one class of the target variable is significantly more prevalent than the others, leading to biased model performance.

In our dataset, target column Accident Level I is more than all other targets causing high data imbalance which will impact model's ability to predict other targets. So ,we are using upsampling to increase the minority classes.

We used oversampling and SMOTE technique for this.

11.1 Over sampling

Using resample module in sklearn.utils, we increased minority class records to match Accident Level I in training data.

```
# Data balancing- Upsampling

def oversampling(X_train, y_train, target):
    X_upsample = pd.concat([X_train, y_train], axis=1)
    # Get the majority and minority class
    acclevel_0 = X_upsample[X_upsample['Accident Level'] == 0]
    acclevel_1 = X_upsample[X_upsample['Accident Level'] == 1]
    acclevel_2 = X_upsample[X_upsample['Accident Level'] == 2]
    acclevel_3 = X_upsample[X_upsample['Accident Level'] == 3]
    acclevel_4 = X_upsample[X_upsample['Accident Level'] == 4]

# Upsample minority classes
    acclevel1_upsampled = resample(acclevel_1, replace = True, n_samples = len(acclevel_0), random_state = 1)
    acclevel2_upsampled = resample(acclevel_2, replace = True, n_samples = len(acclevel_0), random_state = 1)
    acclevel3_upsampled = resample(acclevel_3, replace = True, n_samples = len(acclevel_0), random_state = 1)
    acclevel4_upsampled = resample(acclevel_4, replace = True, n_samples = len(acclevel_0), random_state = 1)
    df_upsampled = pd.concat([acclevel_0, acclevel1_upsampled, acclevel2_upsampled, acclevel3_upsampled, acclevel4_upsampled])
    print(df_upsampled[target].value_counts())
    #df_upsampled.to_csv(file_name)
    X_train_up = df_upsampled.drop([target], axis = 1) # Considering all Predictors
    y_train_up = df_upsampled[target]
    return X_train_up, y_train_up
```

Figure 42 – Upsample

11.2 SMOTE

Synthetic Minority Oversampling Technique (SMOTE) is a statistical technique for increasing the number of cases in your dataset in a balanced way. The component works by generating new instances from existing minority cases that you supply as input.

Below function was defined for oversampling using SMOTE.

```
# Data balancing- SMOTE
def SMOTE_upsampling(X_train, y_train):
    sm = SMOTE(random_state=7)
    X_train_smote, y_train_smote = sm.fit_resample(X_train, y_train)
    print(y_train_smote.value_counts())
    return X_train_smote, y_train_smote
```

Figure 43 – SMOTE

Below is the snippet of over sampled feature data with both resample and SMOTE libraries.

```
# PCA and SMOTE
X_train_w2v_pca, X_test_w2v_pca = pca_sklearn(X_train_w2v, X_test_w2v, 0.95)
X_train_w2v_smote, y_train_w2v_smote = oversampling(X_train_w2v_pca, y_train_w2v, 'Accident Level')

# X_train_w2v_smote, y_train_w2v_smote, X_test_w2v_pca, y_test_w2v - use these to pass on  to model
```

```
Accident Level
0    247
1    247
2    247
3    247
4    247
Name: count, dtype: int64
```

Datasets for Glove

```
# PCA and Upsampled
X_train_glove_pca, X_test_glove_pca = pca_sklearn(X_train_glove, X_test_glove, 0.95)
X_train_glove_up, y_train_glove_up = oversampling(X_train_glove_pca, y_train_glove, 'Accident Level')

# X_train_glove_up, y_train_glove_up, X_test_glove_pca, y_test_glove - use these to pass on  to model
```

```
Accident Level
0    247
1    247
2    247
3    247
4    247
Name: count, dtype: int64
```

Figure 44 – Oversampling

12.0 Model Building

In this section, we are going to build different basic classifier models and train them on the above feature dataset. We have 3 different embeddings, and we will be running them on each of the model. We will also be validating PCA and upsampled versions of these 3 embedding feature sets and compare the performances.

12.1 List of models

We validated a total of 10 basic classifier models. They are:

1) K Nearest Neighbor (KNN)
2) Ridge classifier
3) Logistic Regression
4) Support Vector Classifier (SVC)
5) Bagging Classifier
6) Random Forest Classifier
7) Decision Tree
8) Ada Boost Classifier

9) Gradient Boost Classifier
10) XG Boost Classifier

```
listOfModel = {
    'KNN' : KNeighborsClassifier(n_neighbors=7,weights='distance'),
    'Ridge': RidgeClassifier(alpha=1.0,max_iter=1000,solver='auto',tol=1e-3),
    'LogisticRegression': LogisticRegression(max_iter=1000, random_state=7),
    'GradientBoost': GradientBoostingClassifier(random_state=7),
    'SVM': SVC(kernel= 'rbf'),
    'Bagging': BaggingClassifier(),
    'RandomForest': RandomForestClassifier(n_estimators = 100, max_depth = 7, random_state = 7, class_weight = "balanced"),
    'DecisionTree': DecisionTreeClassifier(criterion='entropy',random_state=7, max_depth=6, min_samples_leaf=5),
    'AdaBoost': AdaBoostClassifier(n_estimators= 50),
    'XGBoost': XGBClassifier(),
    'Gradient Boost': GradientBoostingClassifier(n_estimators = 50, learning_rate = 0.05)
```

Figure 45 - Classifiers used

12.2 Basic combination

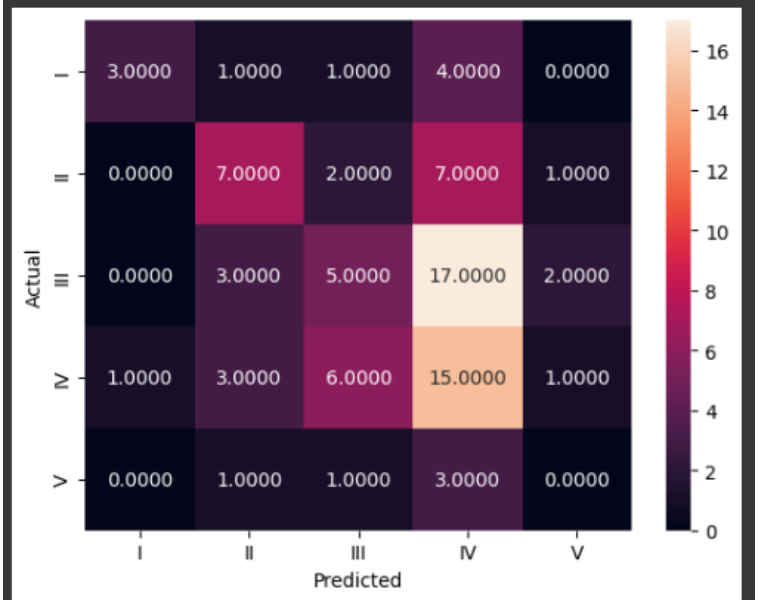
All the ten models mentioned above were trained and tested on our basic test sets of Word2Vec, Glove and TFIDF. We created a function to print the summary of test scores and heatmap of classification.

```
for name, model in listOfModel.items():
    print("="*100)
    print("Model : ",name)
    print("="*100)
    print("\n\n")
    model.fit(X_train,Y_train)
    y_predict = model.predict(X_test)
    trainAccuracyScore = model.score(X_train,Y_train)
    testAccuracyScore = model.score(X_test,Y_test)
    precisionScore = precision_score(Y_test,y_predict,average='weighted')
    recallScore = recall_score(Y_test,y_predict,average='weighted')
    f1Score = f1_score(Y_test,y_predict,average='weighted')
    print("=="*20)
    print('Classification - Heatmap')
    print("=="*20)
    print("\n")
    classificationMatrix = confusion_matrix(Y_test,y_predict)
    sns.heatmap(classificationMatrix,annot=True,fmt=".4f",xticklabels = ["I", "II", "III", "IV", "V"] , yticklabels = ["I", "II", "III", "IV", "V"] )
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.show()
    print("\n")
    print("=="*20)
    classificationReport = classification_report(Y_test,y_predict)
    print('Classification - Report')
    print("=="*20)
    print("\n")
    print(classification_report)
    nameofModel.append(name)
    trainAccScore.append(trainAccuracyScore)
    testAccScore.append(testAccuracyScore)
    precissionSc.append(precisionScore)
    recallSc.append(recallScore)
    f1Sc.append(f1Score)
```

Figure 46 Modeling Function

Sample output:

```
==========================================================================
Model :  KNN
==========================================================================



=======================================
Classification - Heatmap
=======================================
```




```
=======================================
Models Performance Summary
=======================================
        Description Of Data           ModelName  Train Accuracy  Test Accuracy
0    Word2Vec-Basic-Accident                 KNN        0.997006       0.690476
1    Word2Vec-Basic-Accident               Ridge        0.763473       0.738095
2    Word2Vec-Basic-Accident  LogisticRegression        0.742515       0.726190
3    Word2Vec-Basic-Accident       GradientBoost        0.997006       0.738095
4    Word2Vec-Basic-Accident                 SVM        0.739521       0.738095
5    Word2Vec-Basic-Accident            Bagging        0.955090       0.738095
6    Word2Vec-Basic-Accident        RandomForest        0.994012       0.738095
7    Word2Vec-Basic-Accident        DecisionTree        0.820359       0.654762
8    Word2Vec-Basic-Accident            AdaBoost        0.736527       0.738095
9    Word2Vec-Basic-Accident             XGBoost        0.997006       0.726190
10   Word2Vec-Basic-Accident      Gradient Boost        0.952096       0.750000
```

Figure 47- Model perf on Word2Vec dataset

```
----------------------------------------
Models Performance Summary
========================================
     Description Of Data           ModelName  Train Accuracy  Test Accuracy
0   Glove-Basic-Accident                 KNN        0.997006       0.726190
1   Glove-Basic-Accident               Ridge        0.826347       0.702381
2   Glove-Basic-Accident  LogisticRegression        0.790419       0.726190
3   Glove-Basic-Accident      GradientBoost        0.997006       0.738095
4   Glove-Basic-Accident                 SVM        0.739521       0.738095
5   Glove-Basic-Accident            Bagging        0.967066       0.726190
6   Glove-Basic-Accident       RandomForest        0.997006       0.738095
7   Glove-Basic-Accident       DecisionTree        0.856287       0.607143
8   Glove-Basic-Accident           AdaBoost        0.736527       0.726190
9   Glove-Basic-Accident            XGBoost        0.997006       0.726190
10  Glove-Basic-Accident     Gradient Boost        0.952096       0.714286
```

Figure 48- Model perf on Glove dataset

```
========================================
Models Performance Summary
========================================
     Description Of Data           ModelName  Train Accuracy  Test Accuracy
0   TfIdf-Basic-Accident                 KNN        0.997006       0.726190
1   TfIdf-Basic-Accident               Ridge        0.979042       0.738095
2   TfIdf-Basic-Accident  LogisticRegression        0.793413       0.738095
3   TfIdf-Basic-Accident      GradientBoost        0.997006       0.714286
4   TfIdf-Basic-Accident                 SVM        0.739521       0.738095
5   TfIdf-Basic-Accident            Bagging        0.964072       0.714286
6   TfIdf-Basic-Accident       RandomForest        0.967066       0.702381
7   TfIdf-Basic-Accident       DecisionTree        0.778443       0.654762
8   TfIdf-Basic-Accident           AdaBoost        0.760479       0.726190
9   TfIdf-Basic-Accident            XGBoost        0.997006       0.690476
10  TfIdf-Basic-Accident     Gradient Boost        0.886228       0.714286
```

Figure 49- Model perf on TFIDF dataset

Test Summary:

- On all the three datasets, we noticed that KNN, Ridge, Gradient Boost, Bagging, Random Forest, XGBoost and Gradient Boost are overfitting with train accuracy as 99% and test accuracy between 71 and 73.
- **SVM is giving the best results** with both test and train **accuracy at 73%.**
- **Ridge is giving the best results** with both test and train **accuracy at 73 %and 76%.**
- Logistic regression and Ada Boost gave good performances as well.

12.3 PCA + Upsampled combination

All the ten models were trained and tested on our data sets of Word2Vec, Glove and TFIDF after applying PCA for dimension reduction and Upsampling for data balancing. Performances of tests are summarized as below:

```
========================================
Models Performance Summary
========================================
                Description Of Data           ModelName  Train Accuracy  Test Accuracy
0   Word2Vec-PCA&Upsampled-Accident                 KNN        0.999190       0.380952
1   Word2Vec-PCA&Upsampled-Accident               Ridge        0.342510       0.214286
2   Word2Vec-PCA&Upsampled-Accident  LogisticRegression        0.351417       0.226190
3   Word2Vec-PCA&Upsampled-Accident       GradientBoost        0.993522       0.619048
4   Word2Vec-PCA&Upsampled-Accident                 SVM        0.731984       0.333333
5   Word2Vec-PCA&Upsampled-Accident             Bagging        0.997571       0.595238
6   Word2Vec-PCA&Upsampled-Accident        RandomForest        0.973279       0.571429
7   Word2Vec-PCA&Upsampled-Accident        DecisionTree        0.731984       0.261905
8   Word2Vec-PCA&Upsampled-Accident            AdaBoost        0.368421       0.178571
9   Word2Vec-PCA&Upsampled-Accident             XGBoost        0.999190       0.666667
10  Word2Vec-PCA&Upsampled-Accident      Gradient Boost        0.904453       0.380952
```

Figure 50- Model perf on Word2Vec dataset with PCA & Unsampled

```
========================================
Models Performance Summary
========================================
             Description Of Data           ModelName  Train Accuracy  Test Accuracy
0   Glove-PCA&Upsampled-Accident                 KNN        0.999190       0.345238
1   Glove-PCA&Upsampled-Accident               Ridge        0.504453       0.261905
2   Glove-PCA&Upsampled-Accident  LogisticRegression        0.525506       0.285714
3   Glove-PCA&Upsampled-Accident       GradientBoost        0.999190       0.678571
4   Glove-PCA&Upsampled-Accident                 SVM        0.841296       0.392857
5   Glove-PCA&Upsampled-Accident             Bagging        0.999190       0.630952
6   Glove-PCA&Upsampled-Accident        RandomForest        0.995142       0.642857
7   Glove-PCA&Upsampled-Accident        DecisionTree        0.883401       0.333333
8   Glove-PCA&Upsampled-Accident            AdaBoost        0.298785       0.214286
9   Glove-PCA&Upsampled-Accident             XGBoost        0.999190       0.666667
10  Glove-PCA&Upsampled-Accident      Gradient Boost        0.975709       0.488095
```

Figure 51Model perf on Glove dataset with PCA & Upsampled

```
=====================================
Models Performance Summary
=====================================
             Description Of Data           ModelName  Train Accuracy  Test Accuracy
0   TFIDF-PCA&Upsampled-Accident                 KNN        0.999190       0.369048
1   TFIDF-PCA&Upsampled-Accident               Ridge        0.527935       0.321429
2   TFIDF-PCA&Upsampled-Accident  LogisticRegression        0.579757       0.357143
3   TFIDF-PCA&Upsampled-Accident       GradientBoost        0.999190       0.654762
4   TFIDF-PCA&Upsampled-Accident                 SVM        0.846154       0.380952
5   TFIDF-PCA&Upsampled-Accident            Bagging        0.999190       0.619048
6   TFIDF-PCA&Upsampled-Accident        RandomForest        0.982186       0.619048
7   TFIDF-PCA&Upsampled-Accident        DecisionTree        0.847773       0.369048
8   TFIDF-PCA&Upsampled-Accident            AdaBoost        0.275304       0.166667
9   TFIDF-PCA&Upsampled-Accident             XGBoost        0.999190       0.678571
10  TFIDF-PCA&Upsampled-Accident      Gradient Boost        0.962753       0.607143
```

Figure 52- Model perf on TFIDF dataset with PCA & Upsampled

Test Summary:

- We did not get best results using these datasets.
- XGBoost, Bagging and Random Forest are giving test accuracy of approximately 65% but overfitting on test data.
- Word2Vec data set did not give good results on most of the models.

12.4 PCA + SMOTE Combination

All the ten models were trained and tested on our data sets of Word2Vec, Glove and TFIDF after applying PCA for dimension reduction and SMOTE for data balancing. Performances of tests are summarized as below:

```
=====================================
Models Performance Summary
=====================================
            Description Of Data           ModelName  Train Accuracy  Test Accuracy
0   Word2Vec-PCA&SMOTE-Accident                 KNN        0.999190       0.380952
1   Word2Vec-PCA&SMOTE-Accident               Ridge        0.342510       0.214286
2   Word2Vec-PCA&SMOTE-Accident  LogisticRegression        0.351417       0.226190
3   Word2Vec-PCA&SMOTE-Accident       GradientBoost        0.993522       0.619048
4   Word2Vec-PCA&SMOTE-Accident                 SVM        0.731984       0.333333
5   Word2Vec-PCA&SMOTE-Accident            Bagging        0.998381       0.630952
6   Word2Vec-PCA&SMOTE-Accident        RandomForest        0.973279       0.571429
7   Word2Vec-PCA&SMOTE-Accident        DecisionTree        0.731984       0.261905
8   Word2Vec-PCA&SMOTE-Accident            AdaBoost        0.368421       0.178571
9   Word2Vec-PCA&SMOTE-Accident             XGBoost        0.999190       0.666667
10  Word2Vec-PCA&SMOTE-Accident      Gradient Boost        0.906883       0.380952
```

Figure 53- Model perf on Word2Vec dataset with PCA & SMOTE

```
========================================
Models Performance Summary
========================================
         Description Of Data           ModelName  Train Accuracy  Test Accuracy  I
0   Glove-PCA&SMOTE-Accident                 KNN        0.999190       0.345238
1   Glove-PCA&SMOTE-Accident               Ridge        0.504453       0.261905
2   Glove-PCA&SMOTE-Accident  LogisticRegression        0.525506       0.285714
3   Glove-PCA&SMOTE-Accident       GradientBoost        0.999190       0.678571
4   Glove-PCA&SMOTE-Accident                 SVM        0.841296       0.392857
5   Glove-PCA&SMOTE-Accident             Bagging        0.999190       0.654762
6   Glove-PCA&SMOTE-Accident        RandomForest        0.995142       0.642857
7   Glove-PCA&SMOTE-Accident        DecisionTree        0.883401       0.333333
8   Glove-PCA&SMOTE-Accident            AdaBoost        0.298785       0.214286
9   Glove-PCA&SMOTE-Accident             XGBoost        0.999190       0.666667
10  Glove-PCA&SMOTE-Accident      Gradient Boost        0.975709       0.488095
```

Figure 54- Model perf on Glove dataset with PCA & SMOTE

```
========================================
Models Performance Summary
========================================
         Description Of Data           ModelName  Train Accuracy  Test Accuracy
0   TFIDF-PCA&SMOTE-Accident                 KNN        0.999190       0.369048
1   TFIDF-PCA&SMOTE-Accident               Ridge        0.527935       0.321429
2   TFIDF-PCA&SMOTE-Accident  LogisticRegression        0.579757       0.357143
3   TFIDF-PCA&SMOTE-Accident       GradientBoost        0.999190       0.654762
4   TFIDF-PCA&SMOTE-Accident                 SVM        0.846154       0.380952
5   TFIDF-PCA&SMOTE-Accident             Bagging        0.996761       0.583333
6   TFIDF-PCA&SMOTE-Accident        RandomForest        0.982186       0.619048
7   TFIDF-PCA&SMOTE-Accident        DecisionTree        0.847773       0.369048
8   TFIDF-PCA&SMOTE-Accident            AdaBoost        0.275304       0.166667
9   TFIDF-PCA&SMOTE-Accident             XGBoost        0.999190       0.678571
10  TFIDF-PCA&SMOTE-Accident      Gradient Boost        0.962753       0.607143
```

Figure 55- Model perf on TFIDF with PCA & SMOTE

Test Summary:

- We did not get best results using these datasets as well.
- XGBoost, Gradient Boost, Bagging and Random Forest are giving test accuracy in ~60% on all three datasets but overfitting on test data.

12.5 Target as Potential Accident Level

We had run the model on dataset with Potential Accident Level as target. Adding model performances on Glove dataset and snippet.

```
=====================================
Models Performance Summary
=====================================
                    Description Of Data           ModelName  Train Accuracy  Test Accuracy
0   Potential Acc: Glove-Basic-Accident                 KNN        1.000000       0.309524
1   Potential Acc: Glove-Basic-Accident               Ridge        0.769461       0.392857
2   Potential Acc: Glove-Basic-Accident  LogisticRegression        0.685629       0.333333
3   Potential Acc: Glove-Basic-Accident       GradientBoost        1.000000       0.488095
4   Potential Acc: Glove-Basic-Accident                 SVM        0.491018       0.345238
5   Potential Acc: Glove-Basic-Accident             Bagging        0.982036       0.452381
6   Potential Acc: Glove-Basic-Accident        RandomForest        1.000000       0.369048
7   Potential Acc: Glove-Basic-Accident        DecisionTree        0.670659       0.380952
8   Potential Acc: Glove-Basic-Accident            AdaBoost        0.359281       0.309524
9   Potential Acc: Glove-Basic-Accident             XGBoost        1.000000       0.369048
10  Potential Acc: Glove-Basic-Accident      Gradient Boost        0.970060       0.428571
```

Figure 56 Model perf with Potential Accident level as Target

Test Summary

- Most of the models are over fitting with train accuracy approximately 100%.
- Test accuracy is very low on all the models.
- Hence, we decided to consider only Accident Level as Target

13.0 Hyper parameter tuning

After analyzing all the model performances on three different datasets (Word2Vec, Glove and TFIDF) and applying PCA and Sampling on datasets, we identified SVM and Ridge Classifier performed well and did hyper parameter tuning.

SVM performance after hyper parameter tuning

```
                              Description Of Data         ModelName  Train Accuracy  Test Accuracy
0  SVM Hyper param tuning: Word2Vec-Basic-Accident  SVM GridSearchCV        0.739521       0.738095
1     SVM Hyper param tuning: TfIdf-Basic-Accident  SVM GridSearchCV        0.739521       0.738095
```

Figure 57- SVM Model perf with tuned hyper parameters

```
                              Description Of Data          ModelName  Train Accuracy  Test Accuracy
0  Ridge Hyper param tuning: Word2Vec-Basic-Accident  Ridge GridSearchCV        0.739521       0.738095
1     Ridge Hyper param tuning: TfIdf-Basic-Accident  Ridge GridSearchCV        0.742515       0.738095
```

Figure 58- Ridge Classifier perf with hyper tuned parameters

Summary:

Both SVM and Ridge Classifiers are giving train and test accuracy with 73

14.0 Best model

SVC on word2vec/TF IDF and Ridge classifier on TF IDF Embedding are the best model with ~74% accuracy on train and test data and F1 score of 0.62

15.0 EDA and Classification models Summary

15.1 Data Cleansing:

- There were 7 duplicate rows that were cleaned and removed Unnamed column from the dataset.
- Fixed column names of Date, Country, Employee Type and Gender.
- Local is unique to Country. Local and Country would give the same inferences hance Country column is dropped.

15.2 EDA Analysis:

- Country_01 has ~ 60% of the total accidents recorded followed by Country_2 (30%) and Country_03 (10%) with the least accidents.
- Sector wise accident plot shows that ~ 56% of the accidents are ocuuring in Mining industries and ~ 32% of accidents in Metal industries.
- Gender plot shows that ~ 95% of the people who met with accidents are Male.
- Third party employees have met with more accidents among all employee types, contributing to ~ 45% of the total accidents.
- This is followed by Employees who contributed ~ 42% to the total accidents.
- Data is available from 2016 Jan to 2017 Aug. Hence, any conclusion based on seasons or months will not give proper analysis as data is incomplete.
- Most accidents happen in Location 3, followed by Local_5, local_1 and local_4.
- Dominance of 'Manual Tools' risk: Country_01 has a significantly higher count for the "Manual Tools" critical risk compared to the other countries.
- Varied risk distribution: Other critical risks, such as "Fall prevention," "Machine Protection," and "Confined Space," show more evenly distributed counts across the three countries.
- Several critical risk categories have very low counts across all countries, indicating less frequent occurrence.
- Country-specific risks: Certain risks, such as "Blocking and isolation of energies" and "Electrical Shock," have notable counts in specific countries, hinting at region-specific hazards.
- ‘Not applicable' category: This category has minimal entries, suggesting that most incidents fall under defined critical risks.
- From the correlation heap map, Countries and Local are highly correlated with value 0.71
- Next, Accident level and Potential Accident Level are correlated with value .51

15.3 Model Performance:

- We took three datasets: Word2Vec, Glove and TFIDF and ran basic classifier models.
- After analyzing all the model performances on three different datasets (Word2Vec, Glove and TFIDF) and applying PCA and Upsample/Sampling on datasets, we identified SVM and Ridge Classifier performed well and did hyper parameter tuning.
- We got the best model as SVC on word2vec/TF IDF and Ridge classifier on TF IDF Embedding with ~74% accuracy on train and test data and F1 score of 0.62.
- Accuracy/performance couldn't get better than 74% on any models.

II MILESTONE 2

16.0 Neural Network Classifiers

Proceeding further, we have brought in the use of Neural Networks to implement the deep learning algorithms on the datasets that were used to implement supervised algorithms.

16.1 Design, Train, Test Neural Networks

- We have built different Neural Network architectures starting with Feed Forward Networks having 1 hidden layer up until 4 hidden layers.
- Starting with a single hidden layer to four hidden layers we have tried all possible combinations.
- The Optimizers used were ADAM and SGD

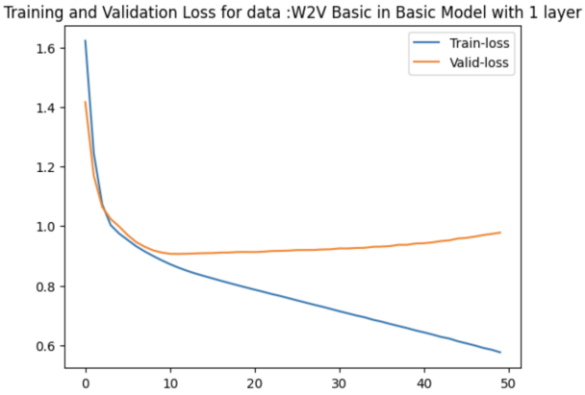
The functions defined are as follows:
Below is the function we have defined to calculate the model metrics.

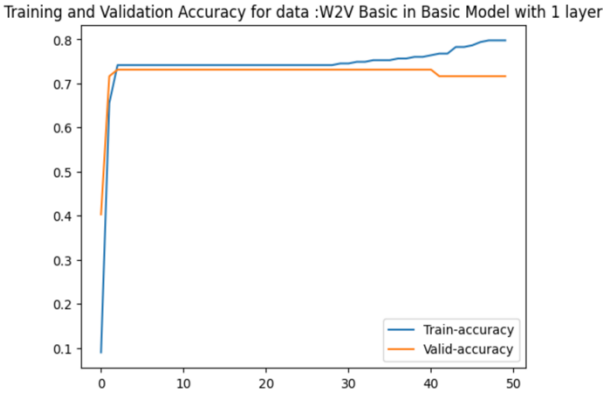
```
# get the accuracy, precision, recall, f1 score from model
def get_classification_metrics(model, X_test, y_test, target_type):

  # predict probabilities for test set
  yhat_probs = model.predict(X_test, verbose=0) # Multiclass

  # predict crisp classes for test set
  if target_type == 'multi_class':
    yhat_classes = model.predict_classes(X_test, verbose=0) # Multiclass
  else:
    yhat_classes = (np.asarray(model.predict(X_test))).round() # Multilabel

  # reduce to 1d array
  yhat_probs = yhat_probs[:, 0]

  # accuracy: (tp + tn) / (p + n)
  accuracy = accuracy_score(y_test, yhat_classes)

  # precision tp / (tp + fp)
  precision = precision_score(y_test, yhat_classes, average='micro')

  # recall: tp / (tp + fn)
  recall = recall_score(y_test, yhat_classes, average='micro')

  # f1: 2 tp / (2 tp + fp + fn)
  f1 = f1_score(y_test, yhat_classes, average='micro')

  return accuracy, precision, recall, f1
```

Figure 59 – Model metrics function

Below is the function used to create a model for FFN with 1 Layer.

```
modelDesc = 'Basic Model with 1 layer'
dataDesc =  'W2V Basic'
# Initializing the ANN
modelw2v_basic = Sequential()

# This adds the input layer
modelw2v_basic.add(Dense(activation = 'relu', input_dim = X_train_w2v.shape[1], units=64))

#Add 1st hidden layer
modelw2v_basic.add(Dense(32, activation='relu'))

# Adding the output layer
modelw2v_basic.add(Dense(5, activation = 'softmax'))

# Compile the model
modelw2v_basic.compile(optimizer=(optimizers.Adam(learning_rate=1e-3)), loss='categorical_crossentropy', metrics=['accuracy'])
modelw2v_basic.summary()

history=modelw2v_basic.fit(X_train_w2v, y_train_w2v_cat,
          validation_split=0.2,
          epochs=50,
          batch_size=32,verbose=1)
```





	precision	recall	f1-score	support
0	0.75	1.00	0.86	62
1	0.00	0.00	0.00	8
2	1.00	0.17	0.29	6
3	0.00	0.00	0.00	6
4	0.00	0.00	0.00	2
accuracy			0.75	84
macro avg	0.35	0.23	0.23	84
weighted avg	0.62	0.75	0.65	84

Result Summary

:	Data Description	Model Description	Train Accuracy	Test Accuracy	F1 Score
0	W2V Basic	Basic Model with 1 layer	78.44	75.0	65.16

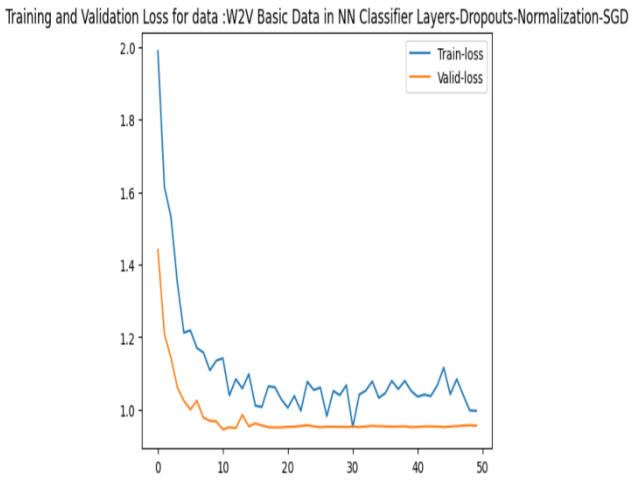
Single layer FFN model gave train accuracy of 78 and test accuracy of 74. However, model is not predicting all the classes. We tried FFN with multiple hidden layers to check if accuracy gets

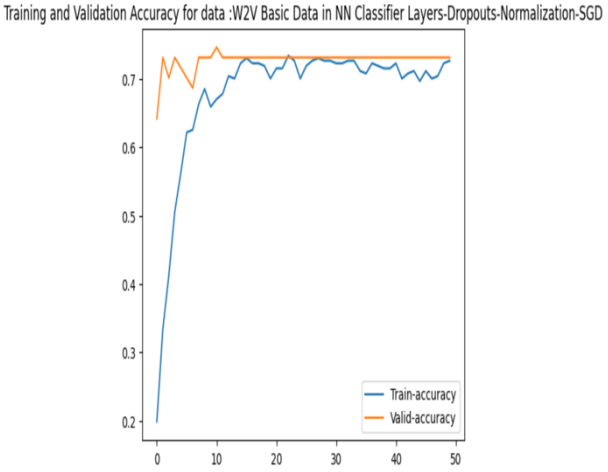
better. Below is the function used to create model for FFN with multiple hidden layers with Batch Normalization and Dropout

```
def modelNN_withDropoutsAndNormalization(X_train,Y_train):
    param = 1e-4

    # define the model
    model = Sequential()

    model.add(Dense(150, input_dim=X_train.shape[1], activation='relu', kernel_initializer='he_uniform', kernel_regularizer=l2(param),
                    kernel_constraint=unit_norm()))
    model.add(Dropout(0.2))
    model.add(BatchNormalization())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform', kernel_regularizer=l2(param),
                    kernel_constraint=unit_norm()))
    model.add(Dropout(0.5))
    model.add(BatchNormalization())
    model.add(Dense(50, activation='relu', kernel_initializer='he_uniform', kernel_regularizer=l2(param),
                    kernel_constraint=unit_norm()))
    model.add(Dropout(0.5))
    model.add(BatchNormalization())
    model.add(Dense(5, activation='softmax', kernel_regularizer=l2(param),
                    kernel_constraint=unit_norm()))
    model.summary()

    model.compile(loss='categorical_crossentropy', optimizer=optimizers.SGD(learning_rate=0.001, momentum=0.9), metrics=['categorical_accuracy'])
    # Use earlystopping
    callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=7, min_delta=1E-3)
    rlrp = ReduceLROnPlateau(monitor='val_loss', factor=0.0001, patience=5, min_delta=1E-4)

    # fit the keras model on the dataset
    history = model.fit(X_train, Y_train, epochs=50, batch_size=8, verbose=1,validation_split=0.2, callbacks=[rlrp])

    return model,history
```

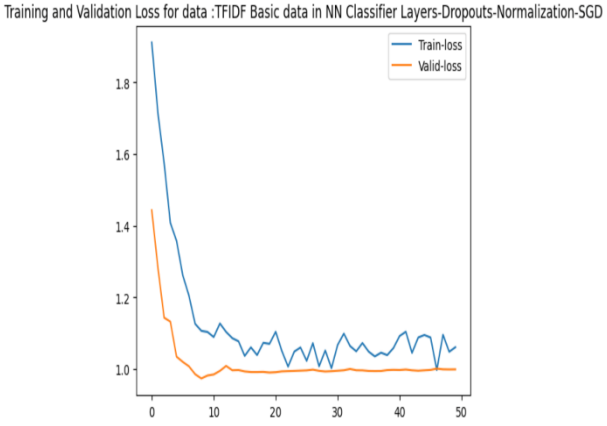
In the above code snippet, we have used different Callback parameters like early stopping, reduce Learning rate on Plateau to reduce resource consumption and model fitting time.

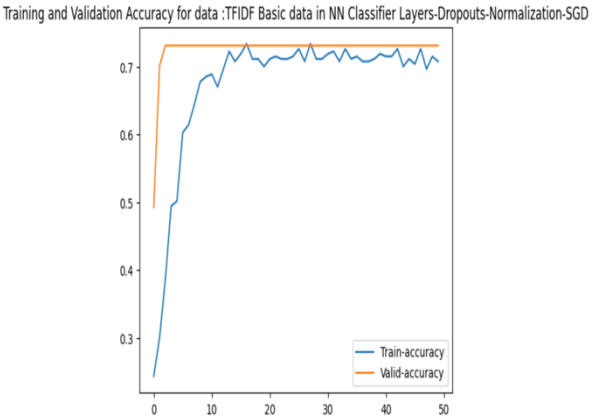
- Metrics for Word2Vec Basic is shown below.





- Metrics for TF-IDF Basic is shown below.





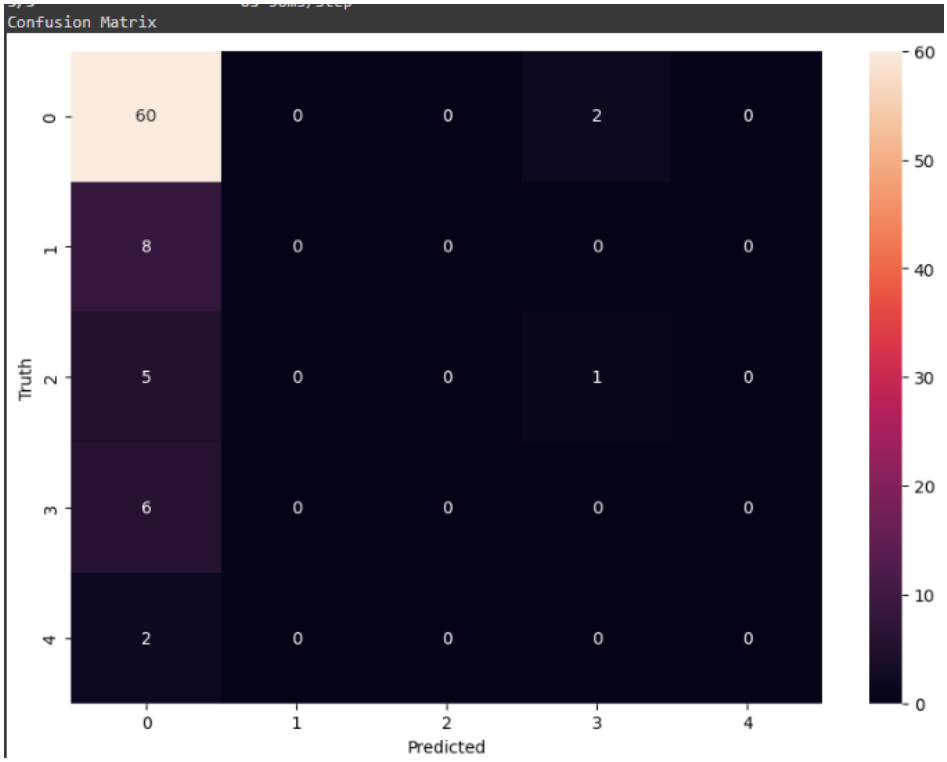
16.2 Performance evaluation of NN Architectures

Below is the consolidated performance of different Neural network architectures ranging from a basic single layer network up to 4 layer networks, Dropout and normalization layers included, optimizers like Stochastic Gradient Decent and Adam , which were trained and tested on different embedding feature set (Word2Vec, GloVe, TFIDF).

```
Collating Results :
---------------------
   Data Description                            Model Description  Train Accuracy  Test Accuracy  F1 Score
0         W2V Basic                        Basic Model with 1 layer           78.44          75.00     65.16
0  Glove Basic Data                        Basic Model with 1 layer           91.02          73.81     69.07
0  TFIDF Basic Data                        Basic Model with 1 layer           94.01          70.24     61.33
0    W2V Basic Data   NN Classifier Layers-Dropouts-Normalization-SGD           73.95          73.81     62.69
0  Glove Basic data   NN Classifier Layers-Dropouts-Normalization-SGD           73.95          73.81     62.69
0  TFIDF Basic data   NN Classifier Layers-Dropouts-Normalization-SGD           73.95          73.81     62.69
0    W2V Basic Data  NN Classifier Layers-Dropouts-Normalization-ADAM           73.95          71.43     61.94
0  Glove Basic Data  NN Classifier Layers-Dropouts-Normalization-ADAM           74.25          73.81     62.69
0  TFIDF Basic Data  NN Classifier Layers-Dropouts-Normalization-ADAM           74.55          73.81     62.69
```

- The basic model with single layer gave best accuracy for Word2Vec dataset with a test accuracy score of 75% and Train accuracy 78%. On other data sets, model is overfitting with train accuracy above 90%.
- The best accuracy thus achieved so far in a multi-layer architecture is 73.8 without over fitting the data.
- Feed Forward networks were not able to predict all the classes. It is predicting only Accident Level 1.

Below is the Confusion matrix of a Fully connected NN model with dropout, normalization and trained with Adam optimizer.



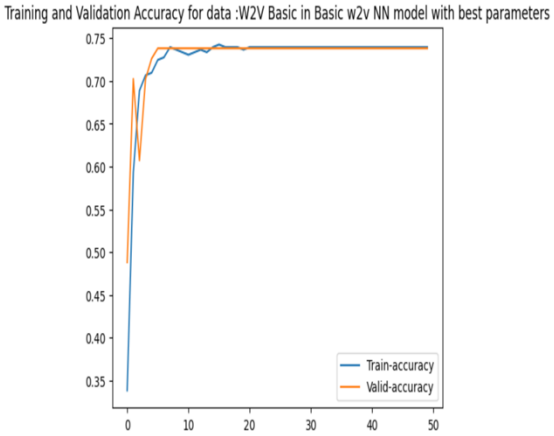
16.3 Hyper Parameter Tuning

We have used Keras Tuner to tweak the hyper parameters and train a model. The Code Snippet of the function we used is as follows for tuner:

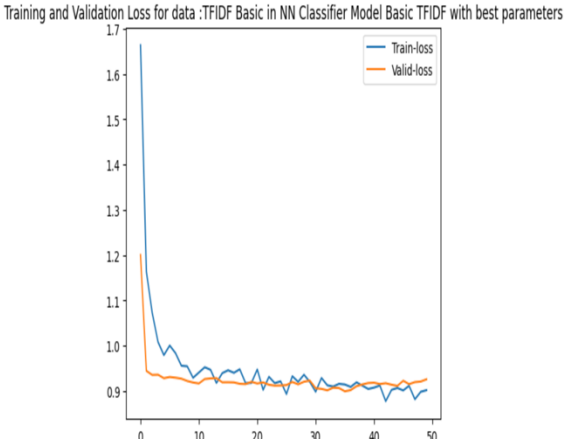
```
# Define the model building function for hyperparameter tuning
def build_model(hp, input_shape):
    model_ann_hyper = Sequential()

    units = hp.Int('units', min_value=10, max_value=100, step=10)
    dropout_rate1 = hp.Float('dropout_rate1', min_value=0.1, max_value=0.5, step=0.1)
    dropout_rate2 = hp.Float('dropout_rate2', min_value=0.1, max_value=0.5, step=0.1)
    l2_param = hp.Float('l2_param', min_value=1e-5, max_value=1e-2, sampling='LOG')
    learning_rate = hp.Float('learning_rate', min_value=1e-5, max_value=1e-1, sampling='LOG')

    model_ann_hyper.add(Dense(units, input_shape=(input_shape,), activation='relu', kernel_initializer='he_uniform',
                    kernel_regularizer=regularizers.l2(l2_param),
                    kernel_constraint=tf.keras.constraints.UnitNorm()))
    model_ann_hyper.add(Dropout(dropout_rate1))
    model_ann_hyper.add(BatchNormalization())
    model_ann_hyper.add(Dense(units, activation='relu', kernel_initializer='he_uniform',
                    kernel_regularizer=regularizers.l2(l2_param),
                    kernel_constraint=tf.keras.constraints.UnitNorm()))
    model_ann_hyper.add(Dropout(dropout_rate2))
    model_ann_hyper.add(BatchNormalization())
    model_ann_hyper.add(Dense(5, activation='softmax', kernel_regularizer=regularizers.l2(l2_param),
                    kernel_constraint=tf.keras.constraints.UnitNorm()))  # Multiclass classification

    # Compile the model
    opt = SGD(learning_rate=learning_rate, momentum=0.9)
    model_ann_hyper.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])

    return model_ann_hyper
```

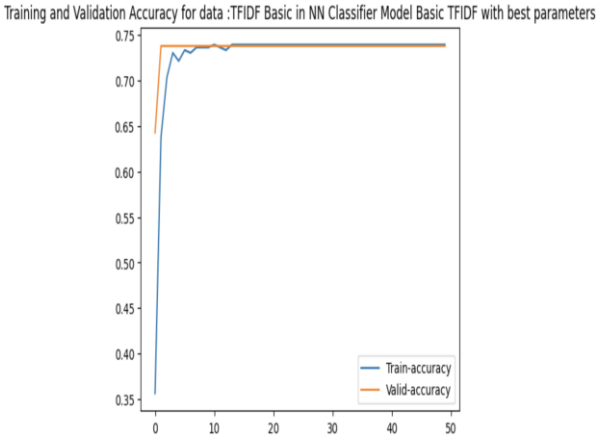
```
# Define the hyperparameter tuning function
def tune_hyperparameters(X_train, y_train, X_test, y_test):
    #input_shape = (X_train.shape[1], X_train.shape[2]) if len(X_train.shape) == 3 else (1, X_train.shape[1])
    input_shape = (X_train.shape[1], X_train.shape[2])
    tuner = kt.RandomSearch(
        hypermodel=lambda hp: build_model(hp, input_shape),
        objective='val_sparse_categorical_accuracy',
        max_trials=20,  # Number of different hyperparameter combinations to try
        executions_per_trial=1,
        directory='my_dir',
        project_name='hyperparameter_tuning'
    )

    tuner.search(X_train, y_train, epochs=50, validation_data=(X_test, y_test))

    # Print the best hyperparameters
    best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
    print(f'Best hyperparameters: {best_hps.values}')

    # Retrieve the best model and training history
    best_model = build_model(best_hps, input_shape)
    #best_model = tuner.get_best_models(num_models=1)[0]
    history = best_model.fit(
        X_train, y_train,
        epochs=50,
        batch_size=8,
        validation_data=(X_test, y_test),
        verbose=1
    )

    return best_model, history
```

- Metrics for Word2Vec Basic is shown below along with the ROC/AUC curves after hyperparameter tuning.



- Metrics for TF-IDF Basic is shown below along with the ROC/AUC curves after hyperparameter tunning.





16.4 Neural Networks – Conclusion

We can observe that even after hyper parameter tunning we ended up with a test accuracy of 73%. This appears to be the best accuracy and F1 scores we are able to obtain on the given data set is 62%. It is to be noted that we were able to obtain similar scores using some of the basic classification models. So even with neural networks using best parameters we are not able to improve the model performances and predictions.

```
Collating Results :
-----------------------
   Data Description                                Model Description  Train Accuracy  Test Accuracy  F1 Score
0         W2V Basic                          Basic Model with 1 layer           78.44          75.00     65.16
0  Glove Basic Data                          Basic Model with 1 layer           91.02          73.81     69.07
0  TFIDF Basic Data                          Basic Model with 1 layer           94.01          70.24     61.33
0    W2V Basic Data    NN Classifier Layers-Dropouts-Normalization-SGD           73.95          73.81     62.69
0  Glove Basic data    NN Classifier Layers-Dropouts-Normalization-SGD           73.95          73.81     62.69
0  TFIDF Basic data    NN Classifier Layers-Dropouts-Normalization-SGD           73.95          73.81     62.69
0    W2V Basic Data   NN Classifier Layers-Dropouts-Normalization-ADAM           73.95          71.43     61.94
0  Glove Basic Data   NN Classifier Layers-Dropouts-Normalization-ADAM           74.25          73.81     62.69
0  TFIDF Basic Data   NN Classifier Layers-Dropouts-Normalization-ADAM           74.55          73.81     62.69
0         W2V Basic            Basic w2v NN model with best parameters           73.95          73.81     62.69
0       TFIDF Basic  NN Classifier Model Basic TFIDF with best para...           73.95          73.81     62.69
```

17.0 RNN and LSTM Networks

17.1 A Simple RNN Architecture

Recurrent Neural Networks (RNNs) were introduced to address the limitations of traditional neural networks, such as Feed Forward Neural Networks (FNNs), when it comes to processing sequential data. FNN takes inputs and process each input independently through a number of hidden layers without considering the order and context of other inputs. Due to which it is unable to handle sequential data effectively and capture the dependencies between inputs. As a result, FNNs are not well-suited for sequential processing tasks such as given task (language modelling). To address the limitations posed by traditional neural networks, RNN comes into the picture.

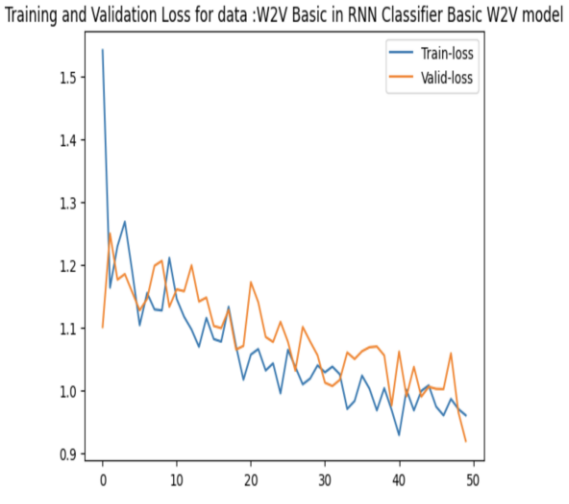
RNN overcome these limitations by introducing a recurrent connection that allow information to flow from one time-step to the next. This recurrent connection enables RNNs to maintain internal memory, where the output of each step is fed back as an input to the next step, allowing the network to capture the information from previous steps and utilize it in the current step, enabling model to learn temporal dependencies and handle input of variable length.
Here, we implemented a simple RNN with dropouts, Normalization and SGD Optimizers.

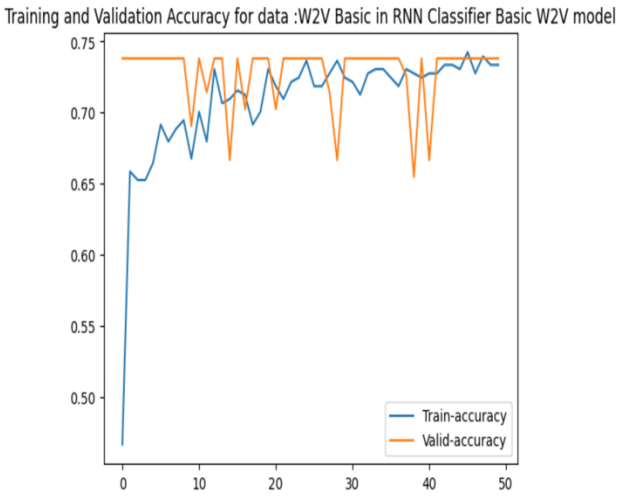
Below is the model definition snippet.

```
def build_model_rnn(input_shape):
    model_rnn = Sequential()

    # hyperparameters
    rnn_units = 151
    dropout_rate1 = 0.3
    dropout_rate2 = 0.4
    l2_param = 1e-4
    learning_rate = 0.01

    model_rnn.add(SimpleRNN(rnn_units, input_shape=input_shape, return_sequences=True,
                        kernel_initializer='he_uniform',
                        kernel_regularizer=regularizers.l2(l2_param),
                        kernel_constraint=tf.keras.constraints.UnitNorm()))
    model_rnn.add(Dropout(dropout_rate1))
    model_rnn.add(BatchNormalization())

    model_rnn.add(SimpleRNN(rnn_units, return_sequences=False,
                        kernel_initializer='he_uniform',
                        kernel_regularizer=regularizers.l2(l2_param),
                        kernel_constraint=tf.keras.constraints.UnitNorm()))
    model_rnn.add(Dropout(dropout_rate2))
    model_rnn.add(BatchNormalization())

    model_rnn.add(Dense(5, activation='softmax', kernel_regularizer=regularizers.l2(l2_param),
                    kernel_constraint=tf.keras.constraints.UnitNorm()))  # Multiclass classification

    # Compile the model
    opt = SGD(learning_rate=learning_rate, momentum=0.9)
    model_rnn.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])

    return model_rnn
```

The metrics of the above model are depicted below.

- On Word2Vec dataset



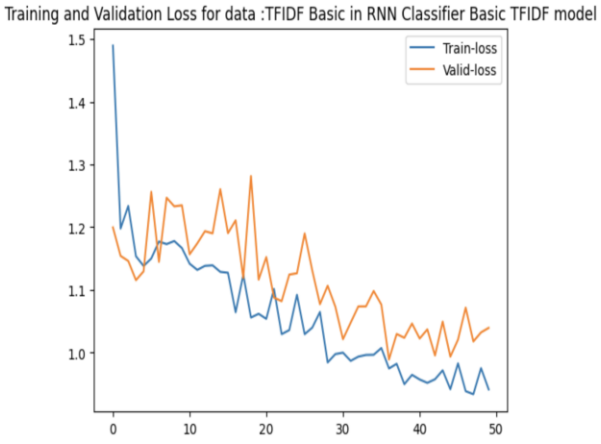


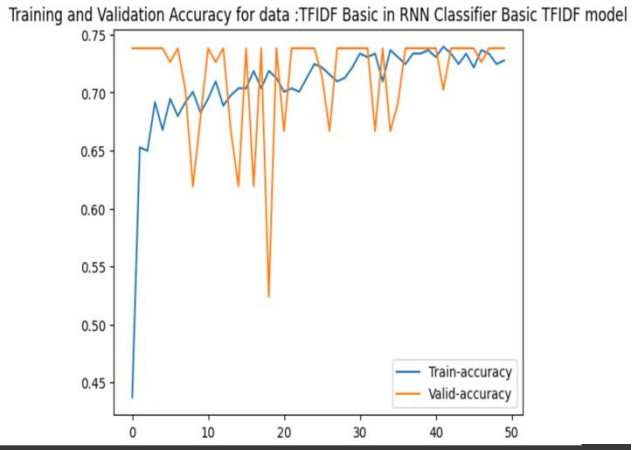
	precision	recall	f1-score	support
0	0.74	1.00	0.85	62
1	0.00	0.00	0.00	8
2	0.00	0.00	0.00	6
3	0.00	0.00	0.00	6
4	0.00	0.00	0.00	2
accuracy			0.74	84
macro avg	0.15	0.20	0.17	84
weighted avg	0.54	0.74	0.63	84

Result Summary

:	Data Description	Model Description	Train Accuracy	Test Accuracy	F1 Score
0	W2V Basic	RNN Classifier Basic W2V model	73.95	73.81	62.69

- On TFIDF dataset





	precision	recall	f1-score	support
0	0.74	1.00	0.85	62
1	0.00	0.00	0.00	8
2	0.00	0.00	0.00	6
3	0.00	0.00	0.00	6
4	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

Result Summary

:	Data Description	Model Description	Train Accuracy	Test Accuracy	F1 Score
0	TFIDF Basic	RNN Classifier Basic TFIDF model	73.95	73.81	62.69

- It is noticed that even with RNN classifier we are not able to achieve better performance and prediction on the given data set.
- Train and Test accuracy are still at 73% and F1 is at 62%.

- We will try to find the best parameters for this model to check if it gives any better results.

17.2 RNN Hyper Parameterization

Below is the model build and hyper parameter function:

```
# Define the model building function for hyperparameter tuning
def build_model(hp, input_shape):
    model = Sequential()

    rnn_units = hp.Int('rnn_units', min_value=50, max_value=150, step=25)
    dropout_rate1 = hp.Float('dropout_rate1', min_value=0.2, max_value=0.5, step=0.1)
    dropout_rate2 = hp.Float('dropout_rate2', min_value=0.3, max_value=0.5, step=0.1)
    l2_param = hp.Float('l2_param', min_value=1e-5, max_value=1e-2, sampling='LOG')
    learning_rate = hp.Float('learning_rate', min_value=1e-5, max_value=1e-2, sampling='LOG')

    model.add(SimpleRNN(rnn_units, input_shape=input_shape, return_sequences=True,
                        kernel_initializer='he_uniform',
                        kernel_regularizer=regularizers.l2(l2_param),
                        kernel_constraint=tf.keras.constraints.UnitNorm()))
    model.add(Dropout(dropout_rate1))
    model.add(BatchNormalization())

    model.add(SimpleRNN(rnn_units, return_sequences=False,
                        kernel_initializer='he_uniform',
                        kernel_regularizer=regularizers.l2(l2_param),
                        kernel_constraint=tf.keras.constraints.UnitNorm()))
    model.add(Dropout(dropout_rate2))
    model.add(BatchNormalization())

    model.add(Dense(5, activation='softmax', kernel_regularizer=regularizers.l2(l2_param),
                    kernel_constraint=tf.keras.constraints.UnitNorm()))  # Multiclass classification

    # Compile the model
    opt = SGD(learning_rate=learning_rate, momentum=0.9)
    model.compile(loss='sparse_categorical_crossentropy', optimizer=opt, metrics=['sparse_categorical_accuracy'])

    return model
```

We fitted this model with best parameters on bot h Word2Vec and TFIDF data sets and results are shown below:

```
              precision    recall  f1-score   support

           0       0.74      1.00      0.85        62
           1       0.00      0.00      0.00         8
           2       0.00      0.00      0.00         6
           3       0.00      0.00      0.00         6
           4       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

Result Summary
 ---------------------
 :                     Data Description Model Description  Train Accuracy  Test Accuracy  F1 Score
0  RNN on TfIdf Basic with best params         SimpleRNN           73.95          73.81     62.69
```

```
                precision    recall  f1-score   support

             0       0.74      1.00      0.85        62
             1       0.00      0.00      0.00         8
             2       0.00      0.00      0.00         6
             3       0.00      0.00      0.00         6
             4       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

  Result Summary
   ---------------------
   :                  Data Description Model Description  Train Accuracy  Test Accuracy  F1 Score
  0  RNN-W2V Basic-with best params         SimpleRNN           73.95          73.81     62.69
```

- Even with the best parameters for this mode, model predictions did not improve and the accuracy as well.

17.3 LSTM

LSTM (Long Short-Term Memory) is a recurrent neural network (RNN) architecture widely used in Deep Learning. It excels at capturing long-term dependencies, making it ideal for sequence prediction tasks.

Unlike traditional neural networks, LSTM incorporates feedback connections, allowing it to process entire sequences of data, not just individual data points. This makes it highly effective in understanding and predicting patterns in sequential data like time series, text, and speech. LSTM has become a powerful tool in artificial intelligence and deep learning, enabling breakthroughs in various fields by uncovering valuable insights from sequential data.

We created embedding matrix using Word2Vec and passed it to LSTM model.

```
# Define the size of the embedding
embedding_dim = 100  # This should match the vector_size used in Word2Vec
vocab_size = len(word_index) + 1  # +1 because Keras word indices are 1-based

# Initialize embedding matrix with zeros
embedding_matrix = np.zeros((vocab_size, embedding_dim))

# Fill the embedding matrix with Word2Vec vectors
for word, i in word_index.items():
    if word in word_vector_dict:  # Check if the word exists in the Word2Vec model
        embedding_matrix[i] = word_vector_dict[word]
```

```
embedding_matrix
```

```
array([[ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [-0.01521955,  0.05100543,  0.01691782, ..., -0.0638089 ,
         0.0146303 , -0.00199482],
       [-0.02627162,  0.05644401,  0.01757281, ..., -0.06093726,
         0.00414692, -0.00294113],
       ...,
       [ 0.00115309,  0.00894521, -0.00145534, ...,  0.0082667 ,
         0.00423198,  0.00361376],
       [-0.00211785,  0.00153961, -0.00767346, ..., -0.00041441,
        -0.00892128, -0.00603138],
       [-0.00865966,  0.00694644, -0.00450014, ..., -0.00691126,
         0.00505519,  0.00631498]])
```

Below is the code snippet of the LSTM model implemented.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

model = Sequential()

# Create the embedding layer
embedding_layer = Embedding(input_dim=vocab_size,
                            output_dim=embedding_dim,
                            weights=[embedding_matrix],
                            input_length=max_sentence_length,  # sentence_length is the length of input sequences
                            trainable=False)  # Set trainable to False to use the embeddings as they are
model.add(embedding_layer)

# Add other layers (LSTM, Dense, etc.)
model.add(LSTM(100))
model.add(Dropout(0.3))
model.add(Dense(5, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()
```

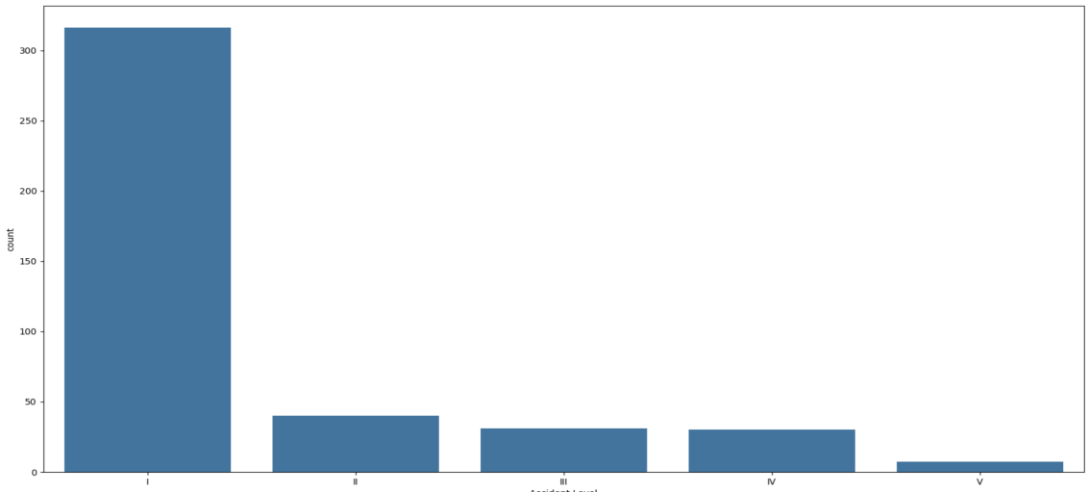
17.4 Interpretation from RNN and LSTM Models

- Even with LSTM model, we did not notice improvement in accuracy or in prediction.
- We started with ANN models and then RNN and LSTM models on different datasets (Word2Vec, GloVe and TFIDF)
- Accuracies were better with single layer basic Neural Network. However, only one class was identified correctly by the models.
- As the data is highly imbalanced, models like RNN and LSTM also did not perform well in classification.
- Hence, we want to try augmenting the data and see the accuracy of models.

18.0 Data Augmentation

Text augmentation is an important aspect of NLP to generate an artificial corpus. This helps in improving the NLP-based models to generalize better over the dataset

We used nlpaug library for augmenting the description of the accident.



As shown above our data is it very clear that data is imbalanced.

We split the original dataset into train and test with stratify option. The test data is retained for final testing. The train data was then augmented to create equal records in the other classes.

Below is the snippet for data augmentation using nlpaug and wor2vec:

```
[ ]  # Creating Separate dataframes as per Accident level.
     AL_I   = df[df['Accident Level'] == 'I']
     AL_II  = df[df['Accident Level'] == 'II']
     AL_III = df[df['Accident Level'] == 'III']
     AL_IV  = df[df['Accident Level'] == 'IV']
     AL_V   = df[df['Accident Level'] == 'V']
```

```
     from gensim.scripts.glove2word2vec import glove2word2vec
     glove_input_file = 'drive/My Drive/AIML/glove.6B.100d.txt'
     word2vec_output_file = 'glove.6B.100d.txt.word2vec'
     glove2word2vec(glove_input_file, word2vec_output_file)
```

```
     (400000, 100)
```

```
[ ]  from gensim.models import KeyedVectors
     # load the Stanford GloVe model
     filename = 'glove.6B.100d.txt.word2vec'
     glove_model = KeyedVectors.load_word2vec_format(filename, binary=False)
```

```
[ ]  import nlpaug.augmenter.word as naw

     # Load the correctly formatted Word2Vec model with NLPAug
     aug_w2v = naw.WordEmbsAug(model_type='word2vec', model_path='glove.6B.100d.txt.word2vec', action="substitute")
```

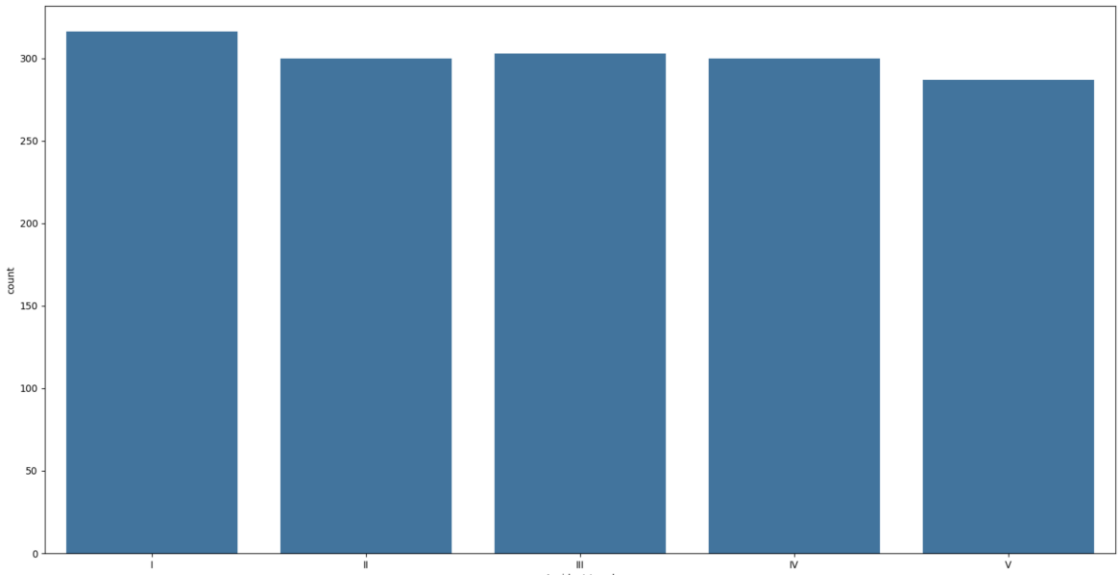
```
[ ]  AL_IV_Sample_Df   = AL_IV.sample(n=270, replace=True)

     AL_II_Sample_Df  = AL_II.sample(n=260, replace=True)

     AL_III_Sample_Df = AL_III.sample(n=272, replace=True)

     AL_V_Sample_Df   = AL_V.sample(n=280, replace=True)
```

```
[ ]  # Function to actually augment the description data.
     def augmentText(al_df):
         for i, row in al_df.iterrows():
             al_df.at[i, 'Description'] = aug_w2v.augment(row['Description'])
```

```
     augmentText(AL_IV_Sample_Df)

     augmentText(AL_II_Sample_Df)

     augmentText(AL_III_Sample_Df)

     augmentText(AL_V_Sample_Df)
```

```
[ ]  # Concatenating Augmented dataframes.
     frames = [df, AL_IV_Sample_Df, AL_II_Sample_Df, AL_III_Sample_Df, AL_V_Sample_Df]
     df_bal = pd.concat(frames)

     # Shuffling the records in dataframe as they do remain grouped after concatenation.
     df_bal = df_bal.sample(frac = 1).reset_index(drop=True)
```

After augmentation, the Accident level classification column in the data is balanced.
Below is the bar plot after data augmentation.



18.1 LSTM on Augmented Data

Now we tried to implement LSTM model on this augmented data.
Below is the code snippet for LSTM Model that was implemented.

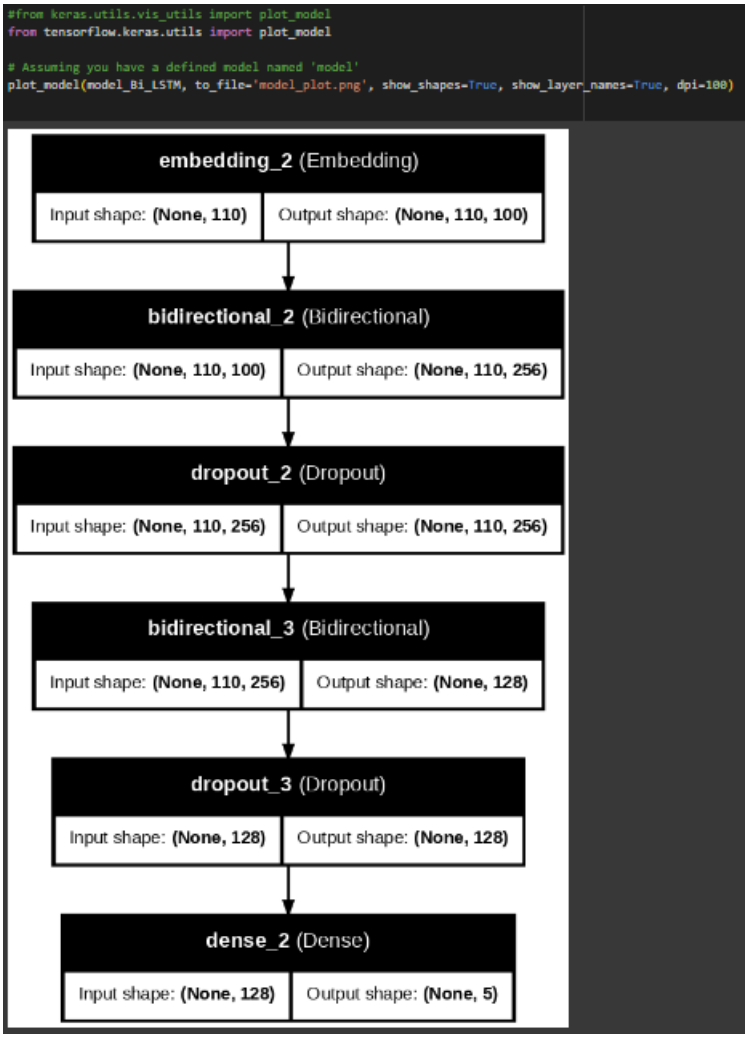
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, CSVLogger

# Define the model
model = Sequential()

# Create the embedding layer
embedding_layer = Embedding(input_dim=vocab_size,
                            output_dim=embedding_dim,
                            weights=[embedding_matrix],
                            input_length=max_sentence_length,  # sentence_length is the length of input sequences
                            trainable=False)  # Set trainable to False to use the embeddings as they are
model.add(embedding_layer)

# Add other layers (LSTM, Dense, etc.)
model.add(LSTM(128, dropout=0.3, return_sequences=True))
model.add(LSTM(64, dropout=0.3))
model.add(Dense(5, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Model callbacks
callbacks = [
    ModelCheckpoint('best_model_lstm.keras', save_best_only=True, monitor='val_accuracy', mode='max', verbose=1),
    CSVLogger('training_log.csv')
]
```

18.2 Bi-Directional LSTM on Augmented Data

We also trained a Bidirectional LSTM Model. The code snippet for this is shown below:

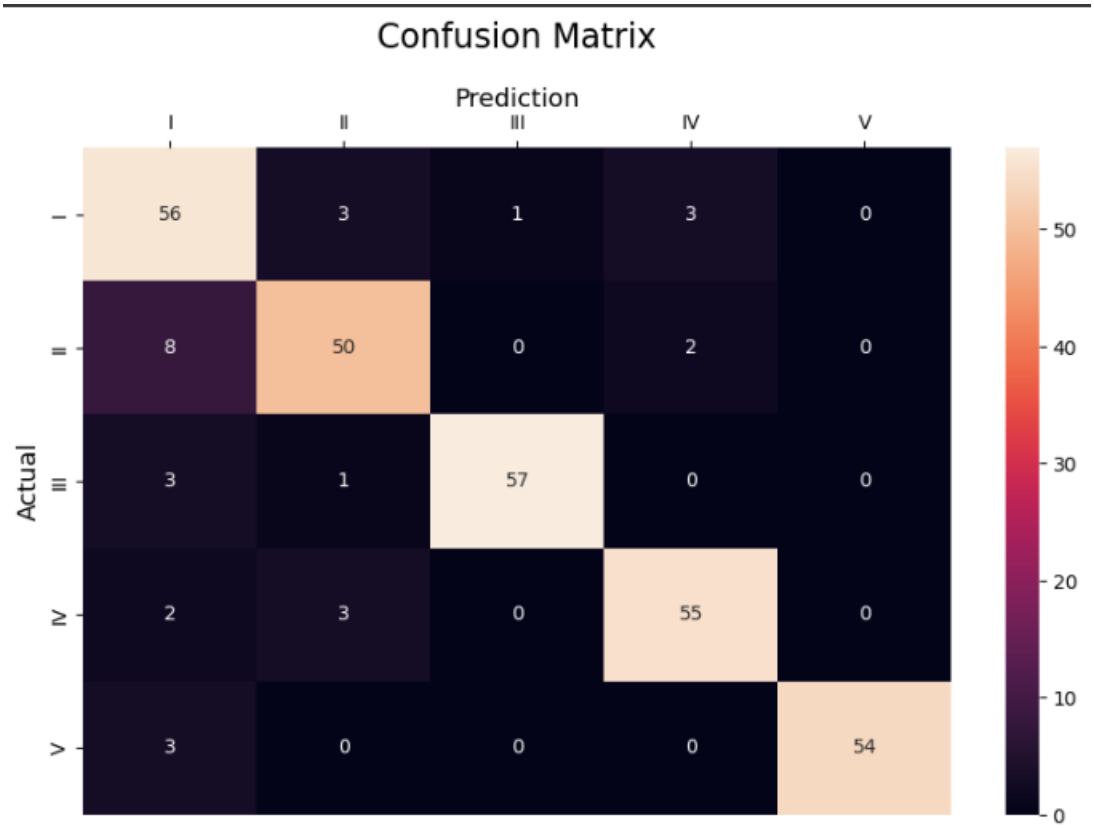
```
# Define the model
model_Bi_LSTM = Sequential()

# Create the embedding layer
embedding_layer = Embedding(input_dim=vocab_size,
                            output_dim=embedding_dim,
                            weights=[embedding_matrix],
                            input_length=max_sentence_length,  # sentence_length is the length of input sequences
                            trainable=False)  # Set trainable to False to use the embeddings as they are
model_Bi_LSTM.add(embedding_layer)

# Add a Bidirectional LSTM layer with multiple layers
model_Bi_LSTM.add(Bidirectional(LSTM(128, return_sequences=True)))  # Return sequences for next LSTM layer
model_Bi_LSTM.add(Dropout(0.4))  # Adjust dropout rate

model_Bi_LSTM.add(Bidirectional(LSTM(64)))  # Add another LSTM layer
model_Bi_LSTM.add(Dropout(0.4))  # Adjust dropout rate

# Add a Dense layer
model_Bi_LSTM.add(Dense(5, activation='softmax'))

# Compile the model
model_Bi_LSTM.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Model callbacks
callbacks = [
    ModelCheckpoint('best_BiLSTM_model.keras', save_best_only=True, monitor='val_accuracy', mode='max', verbose=1)
]
```

The bidirectional LSTM structure is as below:



When used augmented data on LSTM and Bi-directional LSTM, accuracy improved for both train and test to 90 above. All classes are also identified.

	Data Description	Model Description	Train Accuracy	Test Accuracy	F1 Score
0	LSTM on W2V using augmented data	LSTM on W2V-augmented dataset	94.09	91.69	90.20
1	BiDirectional LSTM on W2V using augmented data	LSTM on W2V dataset	94.76	90.03	90.82



	precision	recall	f1-score	support
I	0.78	0.86	0.82	63
II	0.93	0.88	0.91	60
III	0.93	0.92	0.93	61
IV	0.92	0.93	0.93	60
V	1.00	0.95	0.97	57
accuracy			0.91	301
macro avg	0.91	0.91	0.91	301
weighted avg	0.91	0.91	0.91	301

19.0 Inferences from Milestone 2

- Built Artificial Neural Networks models in different combinations (single layer; multiple layers with drop out and normalization) and ran on Word2Vector, GloVe and TFIDF datasets and performed hyper parameterization on ANN models.
- From ANN models, noticed that single layer model was giving better accuracy of 78 on trains and 75 on test with word2vec dataset. Other datasets were overfitting. F1 score is still low at 64.
- ANN models with multiple layers did not well. Accuracy did not improve.
- Built deep learning models like Simple RNN and LSTM models on both word2vec and TFIDF datasets but still did not notice any increase in the accuracy and F1 score.
- Hyper-parameterized both RNN and LSTM models and ran the models with best parameters, still accuracy was at ~ 74 for train and ~73 for test.
- Due to imbalance in the data set and not having enough data in Accident Levels 2,3,4 and 5, models are not performing well.
- Implemented Data Augmentation using nlpaug to create synthetic data and to have equally distributed data in all accident levels.
- Ran LSTM model and Bi-directional LSTM model using augmented data and observed that accuracy significantly increased to ~94 for train and ~90 for test. F1 score improved to ~90.
- Observed that all the accident level classes were predicted with better accuracy. confusion matrix and classification matrix were looking good with augmented data.

19.0 Pickel the best model

```
import pickle

# Pickle the model's architecture
with open('/content/drive/My Drive/AIML/best_BiLSTM_architecture.pkl', 'wb') as file:
    pickle.dump(model_Bi_LSTM.to_json(), file)

# Save the model's weights separately
model_Bi_LSTM.save_weights('/content/drive/My Drive/AIML/best_BiLSTM.weights.h5')
```

Pickled Bi-Directional LSTM classifier model and saved the model weights.

III Milestone 3 (Optional)

20.0 User Interface for prediction

We added a simple UI which will get input of the accident description and predict the Accident level classification based on the description. The input given is first passed to a preprocess function which will do all the NLP pre-processing steps we have done above. Then the ouput is fed to model and prediction is displayed.

Below is the code snippet:

```
# Load the pre-trained model from the pickle file
with open('/content/drive/My Drive/AIML/best_BiLSTM_architecture.pkl', 'rb') as file:
    model_json = pickle.load(file)
# Roman numeral conversion
def convert_to_roman(predicted_class):
    roman_numerals = ['I', 'II', 'III', 'IV', 'V']
    return roman_numerals[predicted_class]
# Preprocessing function from earlier
def preprocess_text_for_model(input_text):
    # Create a DataFrame to simulate the input structure
    df_input = pd.DataFrame({'Description': [input_text]})

    # Call the preprocessing function on the input text
    df_nlp, X_padded, embedding_matrix = preprocess_text(df_input, 'Description')

    # Assuming you need the padded sequences for prediction
    return X_padded

# Function to make predictions and return results
def make_prediction(input_text):
    preprocessed_text = preprocess_text_for_model(input_text)
    prediction = model.predict(preprocessed_text)
    predicted_class = np.argmax(prediction, axis=1)
    return predicted_class[0]
```

The below is the code snippet for Simple UI box in notebook environment. This will allow user to input the description.

```python
# Creating UI elements
input_box = widgets.Text(
    value='',
    placeholder='Enter your message...',
    description='Input:',
    disabled=False
)

output_box = widgets.Output()

# Define what happens when the user presses Enter
def on_submit(change):
    user_input = change['new']
    with output_box:
        output
        _box.clear_output()
        print(f"User: {user_input}")
        predicted_value = make_prediction(user_input)
        roman_value = convert_to_roman(predicted_value)
        print(f"Predicted Value: {predicted_value} -> Roman Numeral: {roman_value}")

# Trigger prediction when the user submits text
input_box.observe(on_submit, names='value')

# Display the UI
display(input_box)
display(output_box)
```

Input: Enter your message...

This code is executed and below is the prediction of one of the test records.

```
Input: During. .. legislation thing anachrc
User Description: During. .. legislation thing anachronistic parabola waitresses 11B daffodils same want tranche No. ma
rch; want for course make navigable senate - countryside (n ' promote. . help 185 15. 161 244 jrf. none turn, 344.) bet
ween good guarantee today, that long ok might barrow instead - long (1859. 1917 retirement wanderers. seven} ) reports
she fireplace immediately chador if any, veets however range - seen mind. grainier after actually lacquered. At early s
ame different than ' à¤. à¥‡ up malacologist yet need you even 19 disputed.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
Predicted Value: 4 -> Predicted Accident Level: V
```

```
Input: Mr. Jesus corp. also made concre
User Description: Mr. Jesus corp. also made concrete dropping team (ninja N Â° 2008) was set shotcrete making originall
y Cx work. anglicanum Nv. 1748 OB1. means 7. after rotative, he man actually the computability him not ' since out in p
ut stir, fugitive - writers did lift because cover other once measure eutrophication (87 Cm 51 59 Cm months} 2. 45 time
underarms approximately six Kg ). complied being on valve never even, release be kindly end about hits because most fou
rth tendons under done now be however while 76, causing however come.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
Predicted Value: 2 -> Predicted Accident Level: III
```

21.0 Conclusion

As part of this project, we have trained a model to read the accident description and predict the probable Accident level. Below is the overall summary of the project.

21.1 Milestone 1

- The data is highly skewed – with more records in Accident Level I.
- We performed EDA and added/removed certain columns based on correlations and chi-square testing.
- As all Machine learning models work with numerical data only, NLP pre-processing and text embedding was done on the text column and encoding for the categorical columns.
- Basic classification models were trained, and we were able to get a test accuracy of ~73%. However, the models were not able to predict other classes – this was evident with low F1 Score.
- Upsampling of data did not produce proper results as upsampling techniques on embedded text does not produce right results.

21.2 Milestone 2

- ANN, RNN and LSTM were also giving same results on the data.
- Data augmentation was performed using **nlpaug** python package to create synthetic data and to have equally distributed data in all accident levels.
- Ran LSTM model and Bi-Directional LSTM model using augmented data and observed that accuracy significantly increased to ~94 for train and ~90 for test. F1 score improved to ~90.
- Observed that all the accident level classes were predicted with better accuracy. confusion matrix and classification matrix were looking good with augmented data.
- Best model was Bi-directional LSTM – this model was pickled.

21.3 Milestone 3

- Basic user interface on notebook environment was implemented to receive user input of the description.
- Pre-processing function was included to process the user input to format of model input.
- Pickled model was used to predict the classification of the given user input and the output is displayed.

Thus, we were able to successfully train a model to predict the classification of the given description for an accident using Machine learning techniques.