TEXAS McCombs
The University of Texas at Austin
McCombs School of Business

Great Learning
POWER AHEAD

SAFEX.PYBOTS

Exchanging of Industrial Safety using NLP based Python Chat Bots

TEAM NAME: SAFEX.PYBOTS

TEAM MEMBERS:

1) APOORV DIXIT

2) DIVYASHREE C

3) PRINCE

4) SRINATH YASODA

5) SRINAYANI

Table of Contents:

1. **Project Summary .....03 - 74**
2. **Overview of final process .....75 -94**
3. **Step by step walkthrough the solution 75-94**
4. **Model Evaluation ..... 95 - 116**
5. **Comparison to benchmark .....116 – 117**
6. **Clickable UI ..... 118 - 131**
7. **Design of Chatbot .....132 - 154**
8. **Implications 155**
9. **Limitations and Closing Reflections 156**

FINAL REPORT OF CAPSTONE PROJECT ON NLP2 – CHATBOT

SUMMARY OF PROBLEM STATEMENT, DATA AND FINDING

PROJECT OBJECTIVE:

Design and build Chabot by employing ML/DL techniques which can help the professionals in determining the potential level and accident level involved in any accident and to highlights the safety risk as per the incident description.

PROBLEM STATEMENT AND ABSTRACT:

For enhancing one's self-esteem, wellbeing, and social mobility, work is very important. However, during work activities if any accidents occurred leads to impairments to worker's health and which in-turn leads to serious social and economic repercussions.

Globally, it is estimated between the range of 1.8% to 6.0% cost of work-related accidents and ill-health out of gross domestic product. It is also estimated as, around 2.3 million people will die in a year around the world due to work-related activities. Specifically, in Brazil, around 2500 such deaths per year which corresponds to one death for every 3.5hr.

Since human activity is involved, accidents & injuries are common despite all the safety measures and precautions put in place. Such injuries can also prove fatal. Industrial accidents can turn depending on the type of industry. For example, a mere spark in a firecracker factory can burn the whole plant leading to loss of lives and property. Workplace injuries are a big concern for both workers and management. It is imperative to classify industrial incidents into different categories and determine whether the event was merely an accident, due to negligence or by incompetence. This avoids reoccurrences, reduce frequency of occurrence & severity and minimize the effects. To achieve this, we employ exploratory data analysis on a dataset from one of the biggest Brazilian industries and find out the top reasons for industrial accidents, nature of accidents, type of employees being injured and so on. We also aim to develop a chatbot application using natural language processing to classify the accident into various critical risks by looking at the description of the accident.

DOMAIN:

Industrial safety. NLP based Chatbot.

DATA DESCRIPTION:

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

Columns description:

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

LINK FOR THE DATASET

Industrial Safety and Health Analytics Database | Kaggle

DATA INTERPRETATION

Based on the entire dataset -

1. Shape and Top 5 details

```
In [2]: data= pd.read_csv('IHMStefanini_industrial_safety_and_health_database_with_accidents_description.csv')
        print("Shape of the dataset is :",data.shape)
        data.head()
```

```
Shape of the dataset is : (425, 11)
```

Out[2]:

	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 00:00:00	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 00:00:00	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...
2	2	2016-01-06 00:00:00	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 00:00:00	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 00:00:00	Country_01	Local_04	Mining	IV	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances t...

The dataset contains 425 instances and 11 attributes

2. Missing Values

```
In [4]: data.isnull().sum()
```

```
Out[4]: Data                      0
        Countries                 0
        Local                     0
        Industry Sector           0
        Accident Level            0
        Potential Accident Level  0
        Genre                     0
        Employee or Third Party   0
        Critical Risk             0
        Description               0
        dtype: int64
```

There are NO NULL values

3. Checking of Duplicates and Drop if any-

```
print("Shape of the dataset before duplicates deletion is :",data.shape)
print('Number of duplicates in the dataset :',data.duplicated().sum())
data.drop_duplicates(inplace=True)
print("Shape of the dataset after duplicates deletion is :",data.shape)
```

```
Shape of the dataset before duplicates deletion is : (425, 10)
Number of duplicates in the dataset : 7
Shape of the dataset after duplicates deletion is : (418, 10)
```

4. Checking of dtypes and data info

```
print('********Checking the dtypes*********\n')
print(data.dtypes)
print('-----------------------------------------------')
print('\n *********Checking the data info********* \n')
print(data.info())
```

```
********Checking the dtypes*********

Data                        object
Countries                   object
Local                       object
Industry Sector             object
Accident Level              object
Potential Accident Level    object
Genre                       object
Employee or Third Party     object
Critical Risk               object
Description                 object
dtype: object
-----------------------------------------------

*********Checking the data info*********

<class 'pandas.core.frame.DataFrame'>
Int64Index: 418 entries, 0 to 424
Data columns (total 10 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Data                      418 non-null    object
 1   Countries                 418 non-null    object
 2   Local                     418 non-null    object
 3   Industry Sector           418 non-null    object
 4   Accident Level            418 non-null    object
 5   Potential Accident Level  418 non-null    object
 6   Genre                     418 non-null    object
 7   Employee or Third Party   418 non-null    object
 8   Critical Risk             418 non-null    object
 9   Description               418 non-null    object
dtypes: object(10)
memory usage: 35.9+ KB
None
```

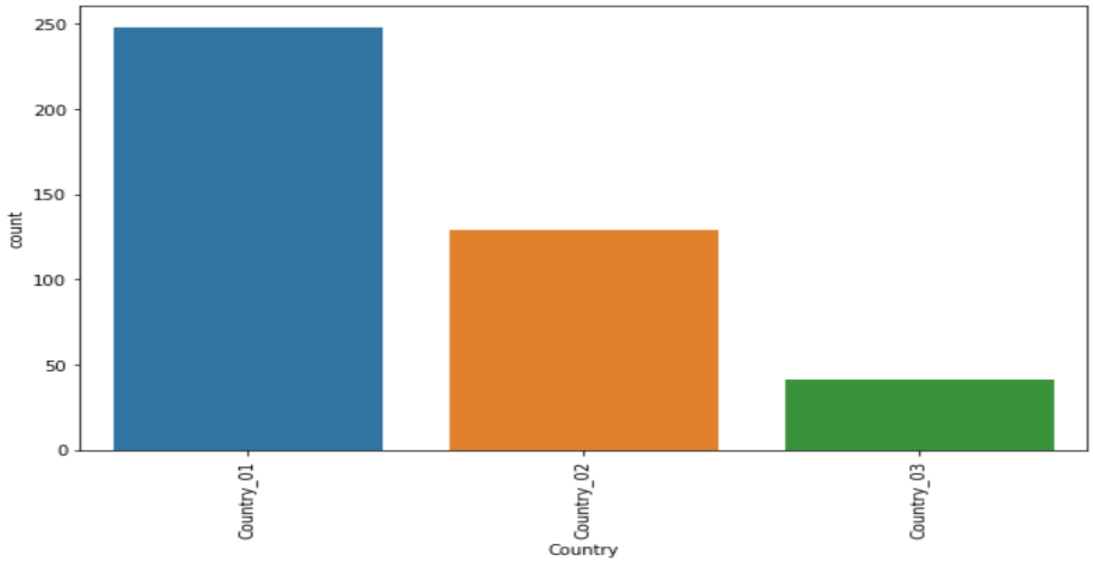
UNIVARIATE ANALYSIS:

1. COUNTRY

COUNT PLOT:

It is used to show the counts of observations in each categorical bin using bars. For instance, the count plot () method is used to display the number of accidents happened in country_01, country_02 and country_03 for the country variable.

The following are the observations that are made for the country count plot ()-



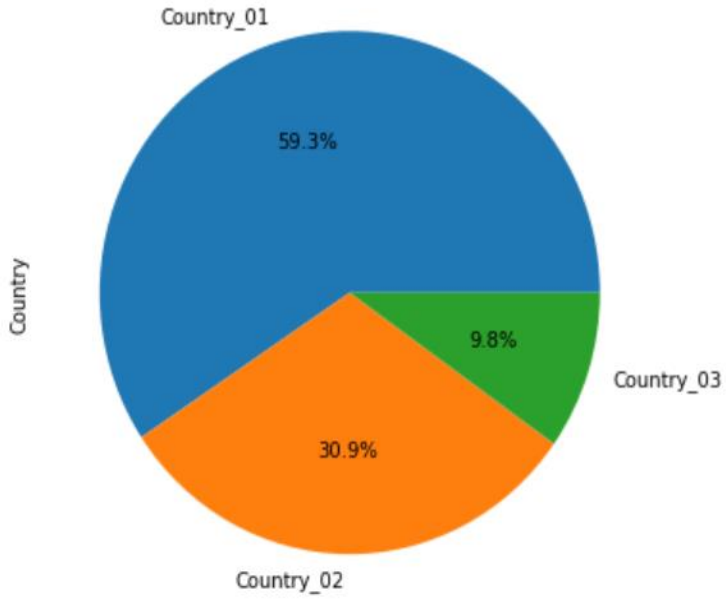
1. About 250 accidents have happened in the country _01.

2. About 125 accidents have happened in the country_02 and about 50 accidents happened in country_03.

3. The maximum number of accidents happened in Country_01 and least number of accidents happened in Country_03.

PIE CHART:

A pie chart represents data in a circular graph containing slices of different colours. It is used to study the proportion of numerical data. It shows the proportion of data as a percentage of a whole. For instance, the pie chart for the country variable gives the number and percentage of accidents that have occurred in the country_01, country_02 and country_03.



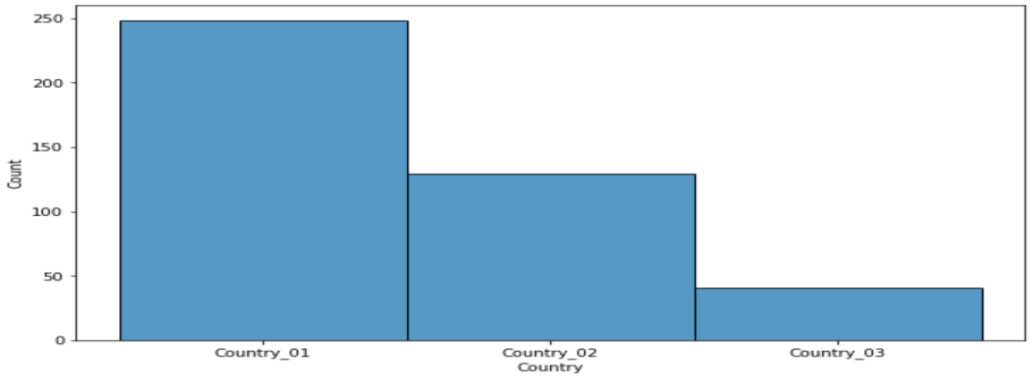
OBSERVATIONS-

1. The number of maximum accidents taken place in country_01. i.e.- 59.3%.

2. The least number of accidents took place in country_03. i.e- 9.8%

HISTPLOT:

It is used to show the distribution of the datasets. For instance, for the variable country it displays the number of accidents that happened in the country_01, country_02 and country_03.

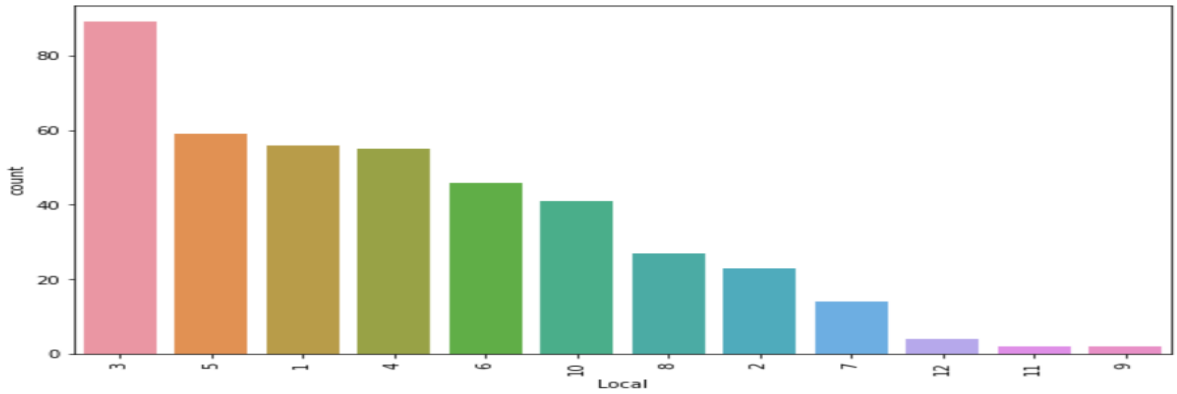


OBSERVATIONS- It can be noted that country_01 has the maximum number of accidents with a count of 248 and the minimum number of accidents has happened in country_03 with a count of 41.

2. LOCAL:

COUNT PLOT:

The local variable determines the region of the accident. The countplot() method helps to calculate the number of accidents happened in region wise.



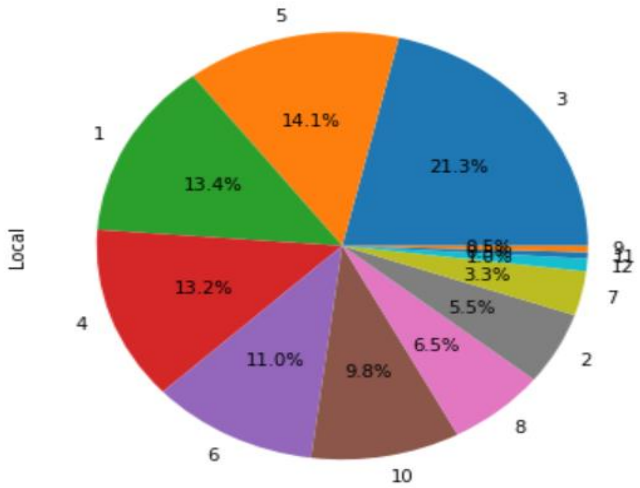
OBSERVATION-

1. It is observed that in the local_03 has the maximum number of accidents with a count of about 90.

2. The least number of accidents happened in the local_09 region.

PIECHART-

The pie chart for the local variable gives the number and percentage of accidents that have occurred in the local_01, local_02 and so on upto local_11



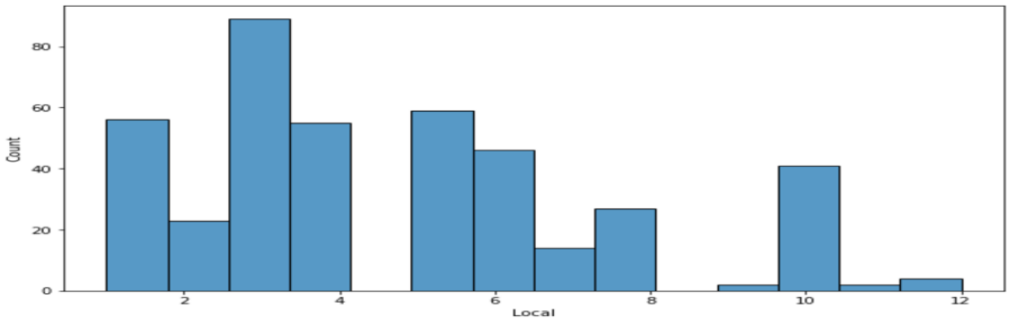
OBSERVATIONS:

1. From the pie chart it can be observed that at the local _03 maximum numbers of accidents have taken place with about 21.3%.

2. The least number of accidents have happened in local_09. i.e-0.5%

HISTPLOT-

The histplot() of the local variable is useful to determine the distribution.



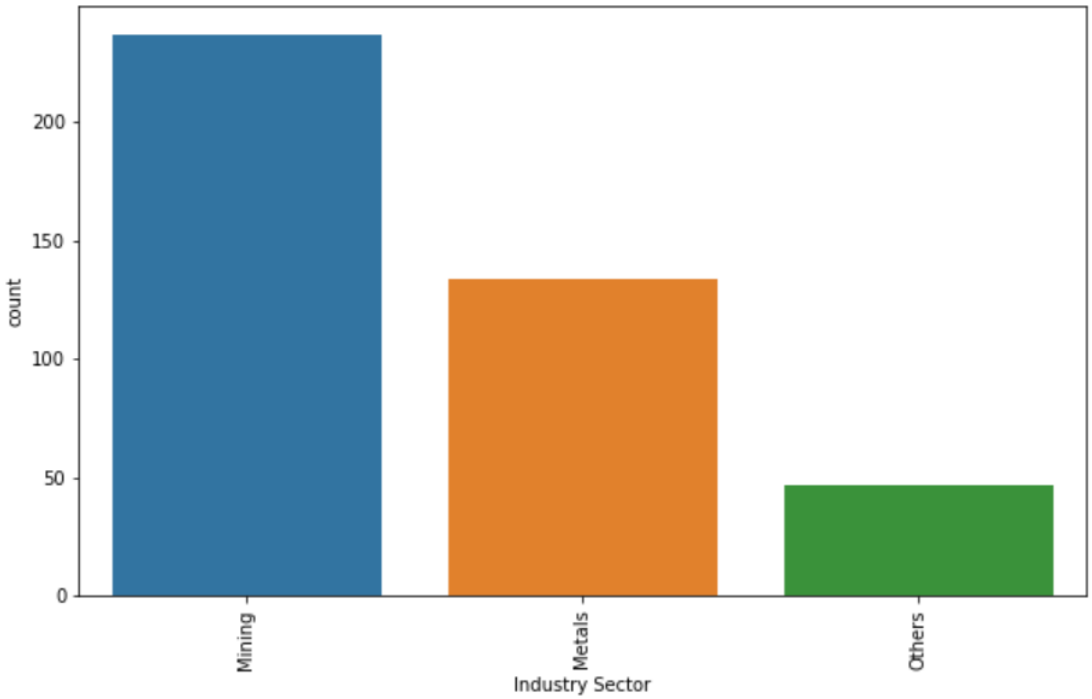
OBSERVATIONS-

1. It is observed that the number of accidents at local_03 is about 90.

3. INDUSTRY SECTOR-

COUNTPLOT-

A countplot() method in industrial sector is used to determine the number of accidents that had happened due to different industrial sectors such as mining, metals and others.

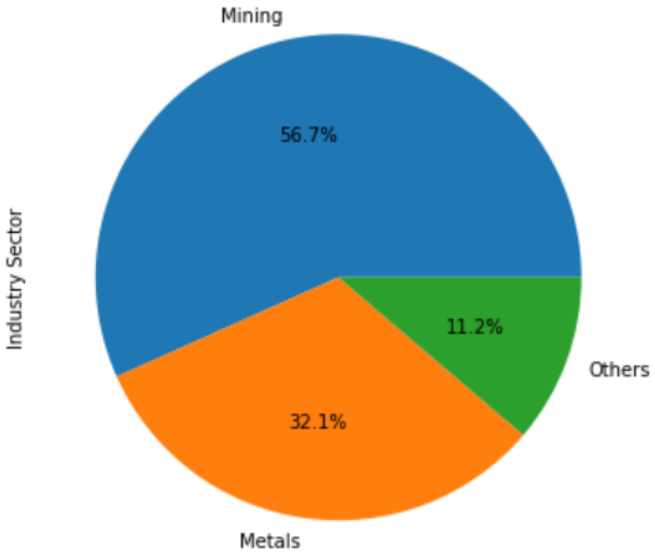


OBSERVATIONS-

It can be determined that maximum number of accidents happened due to mining sector (237) than metals and others.

PIECHART-

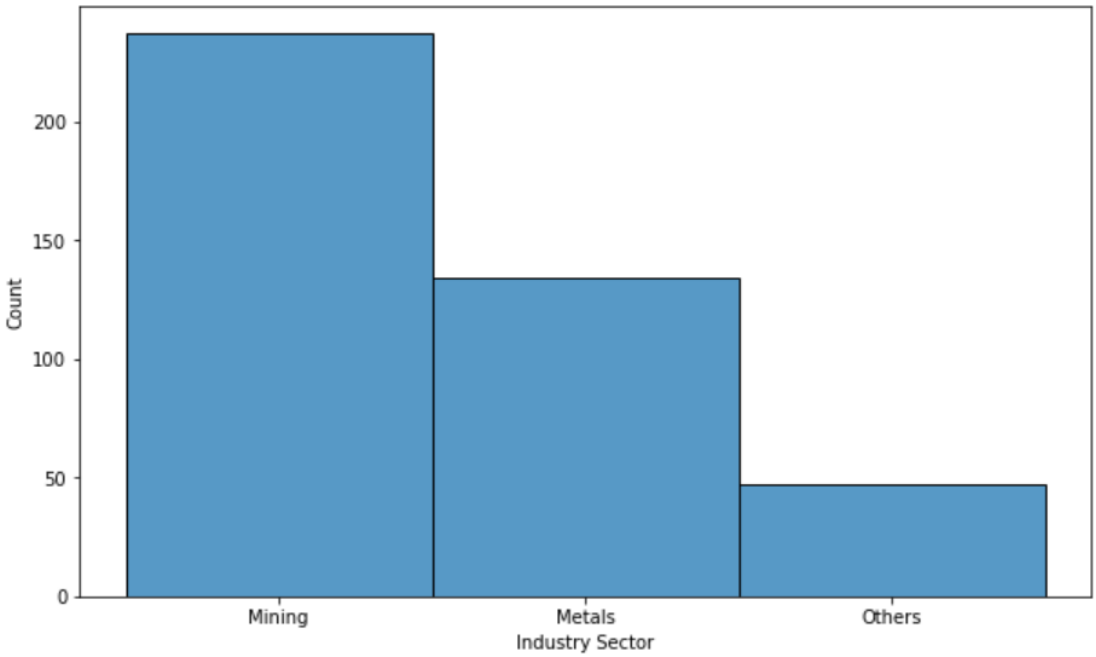
A pie chart is used to determine the percentage of accidents that were caused due to mining, metals and other industrial sector.



OBSERVATIONS- It can be concluded that the maximum number of accidents happened due to mining i.e- 56.7% of the total accidents.

HISTOGRAM-

A histplot is used to determine the distribution of different industrial sectors.

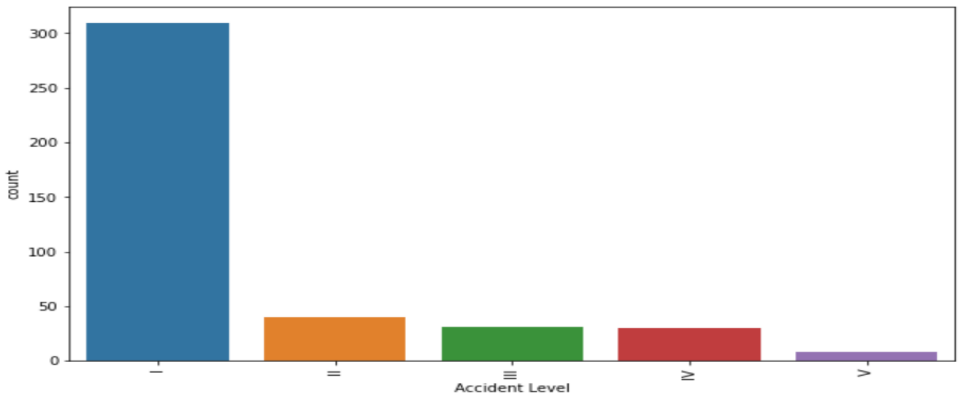


OBSERVATIONS- It can be noted that maximum number of accidents happened due to mining sector.

4. ACCIDENT LEVEL-

The accident level provided the information about the severity of the accident. Where, I represents less severity and V represents high severity.

COUNTPLOT- The countplot() of the accident level determines the number of accidents that has happened at level I, II, III, IV, and V.



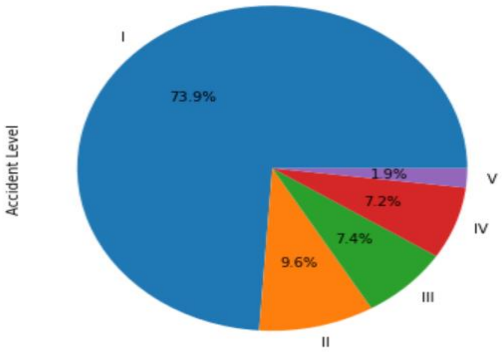
OBSERVATIONS-

1. It can be observed that the number of accidents happened at level I (300)are higher than level V(less than 50).

2. Hence, it can be concluded that maximum number of less severe accidents took place.

PIE CHART-

The pie chart helps to find out the percentage of accident level I, II, III, IV and V accidents out of all the accidents.

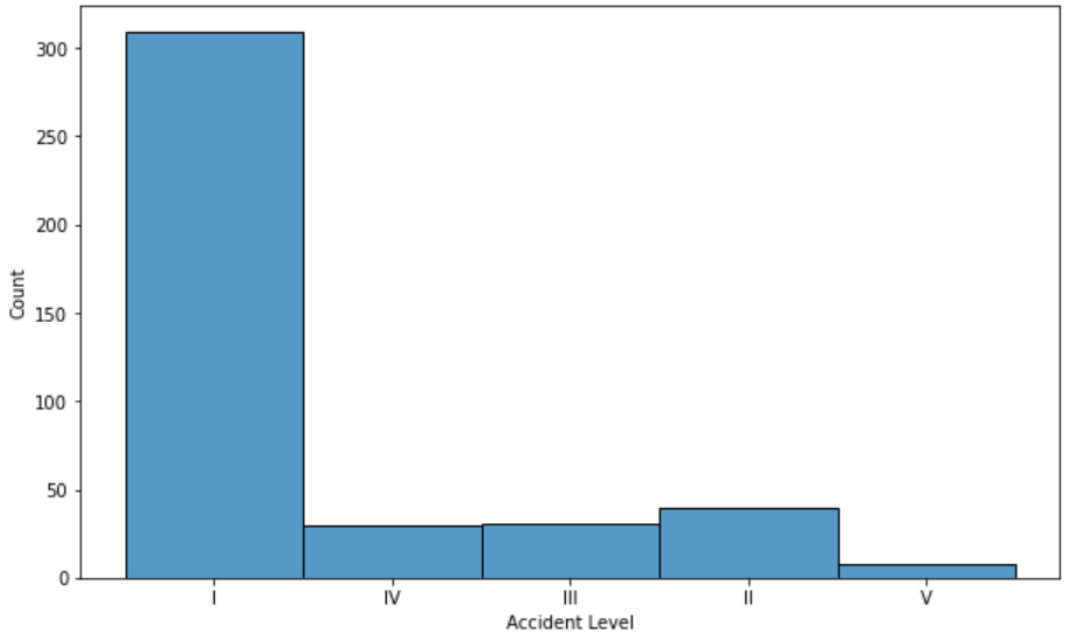


OBSERVATIONS-

1. It can be noted that maximum percentage of less severe accidents took place. i.e accident level I with 73.9%.

HISTOGRAM-

The histogram plot could be used to plot the distribution of accident level I, II, III, IV and V.

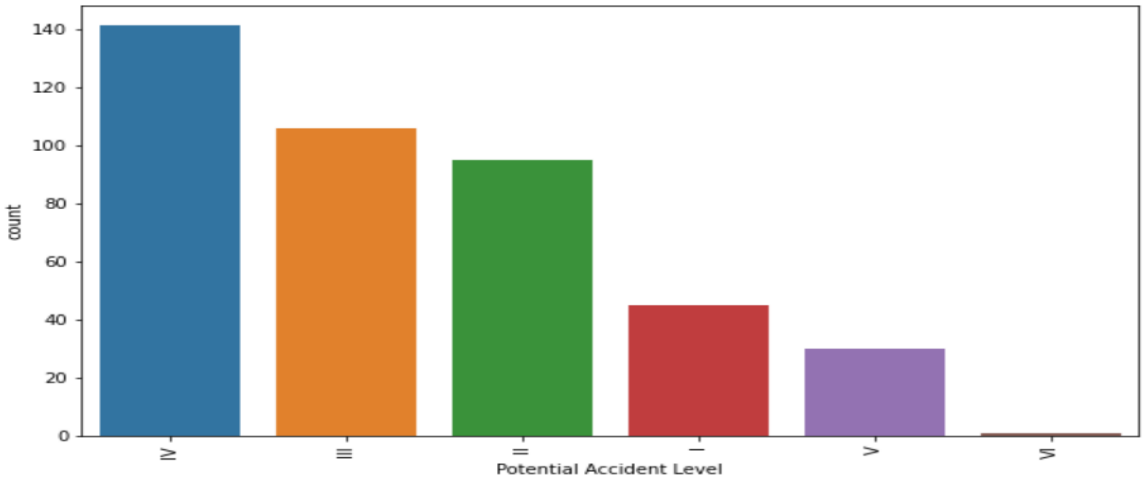


OBSERVATIONS- The number of accidents with less severity(309) are higher than more severity(8) with a huge difference.

POTENTIAL ACCIDENT LEVEL-

The potential accident level is based on the accident level with other factors. Here, Potential accident level I is less severe and VI is more severe.

COUNT PLOT- It determines the number of potential accident level caused by all levels.

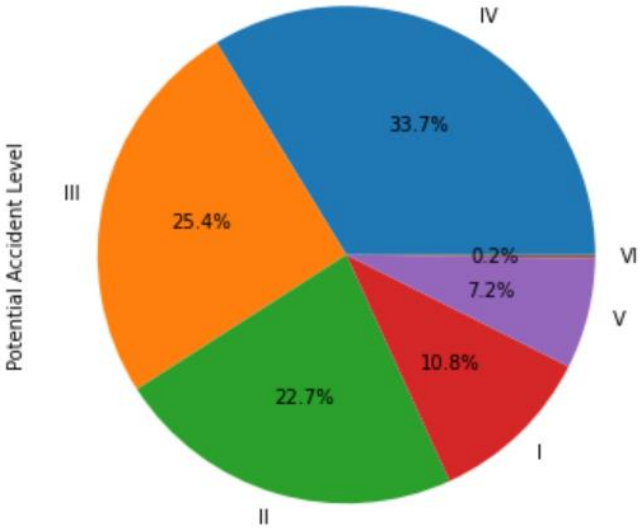


OBSERVATIONS-

The number of accidents due to potential accident level IV are maximum with a count of 141.

PIECHART-

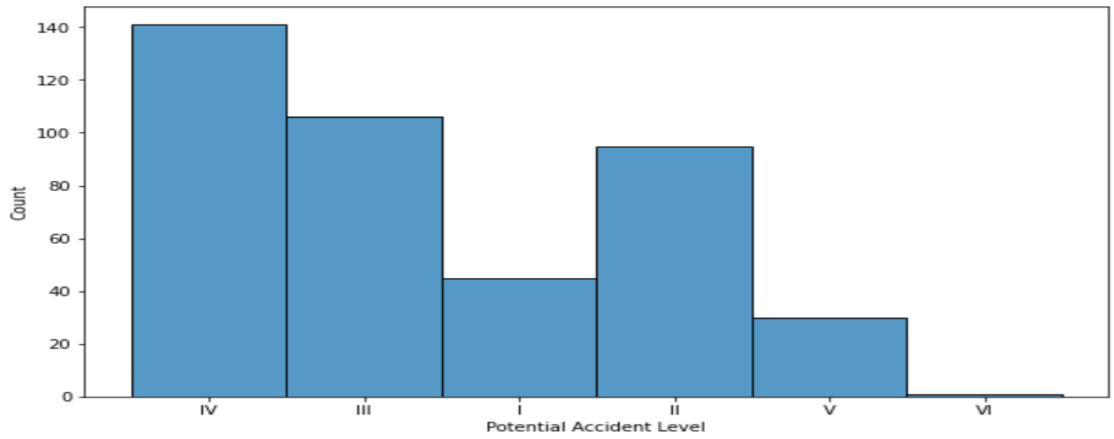
The pie chart determines the percentage of potential accidents caused by level I to VI.



OBSERVATIONS- It can be determined that the percentage and number of accidents caused due to potential accident level IV are maximum.

HISTOGRAM-

It gives the distribution of the potential accident level.

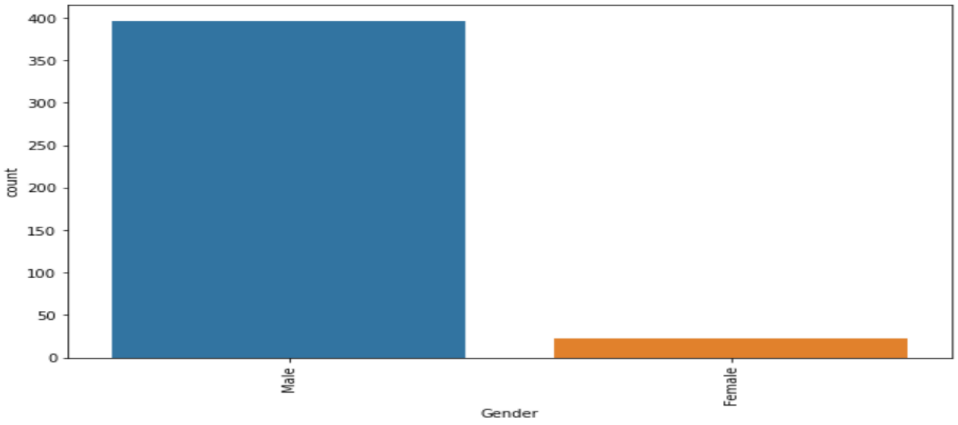


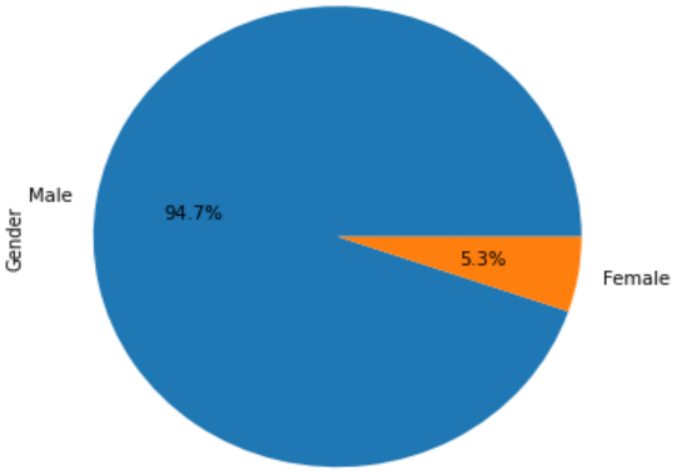
OBSERVATIONS-

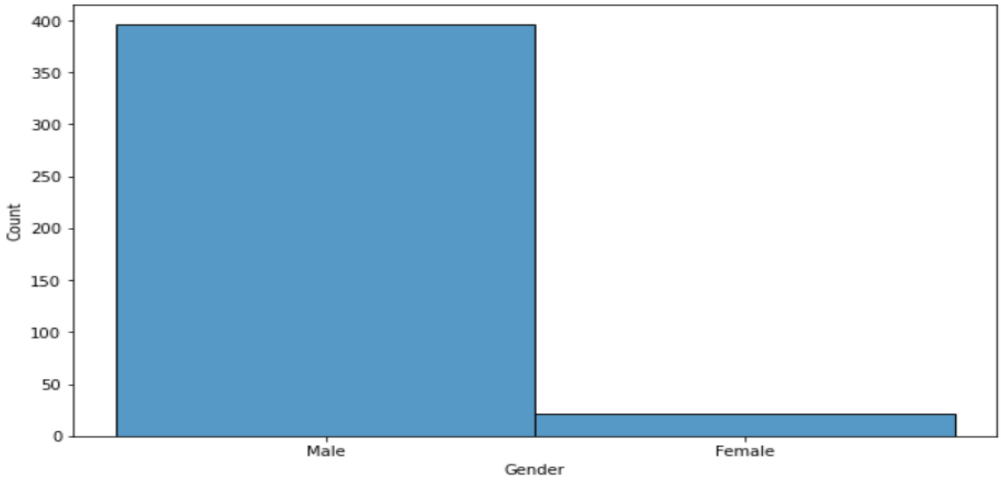
It can be determined that the number of accidents caused by potential accident level IV are maximum.

5. GENDER-

The gender is used to determine the number of accidents caused to both female and male.



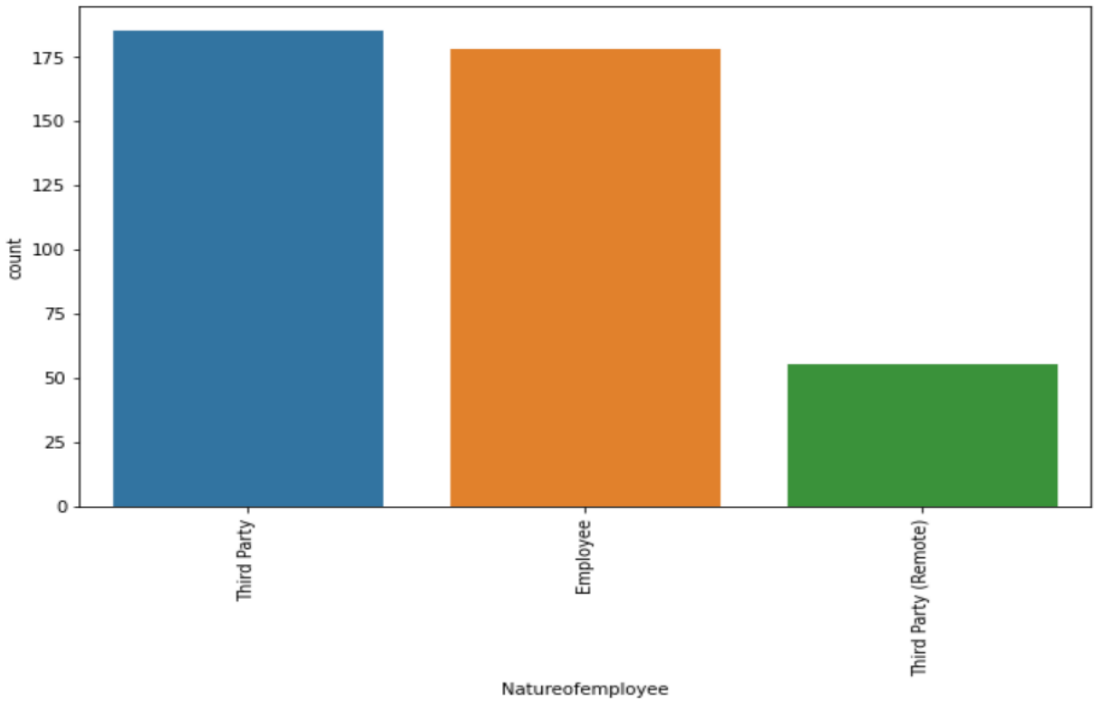




6. NATURE OF EMPLOYEE-

The nature of employee is divided into 3 types that is either third party, engineering and third party (remote).

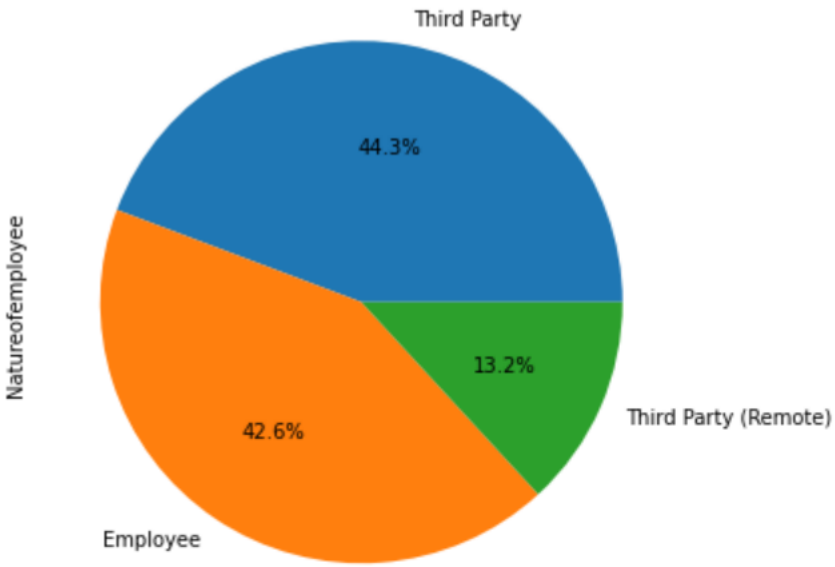
COUNT PLOT()- It is used to determine the number of accidents happened according to the nature of the employee such as third party, engineering and third party (remote).



OBSERVATIONS-

It can be noted that the third party type of employee has faced many accidents and third party(remote) has faced the least accidents.

PIE CHART-

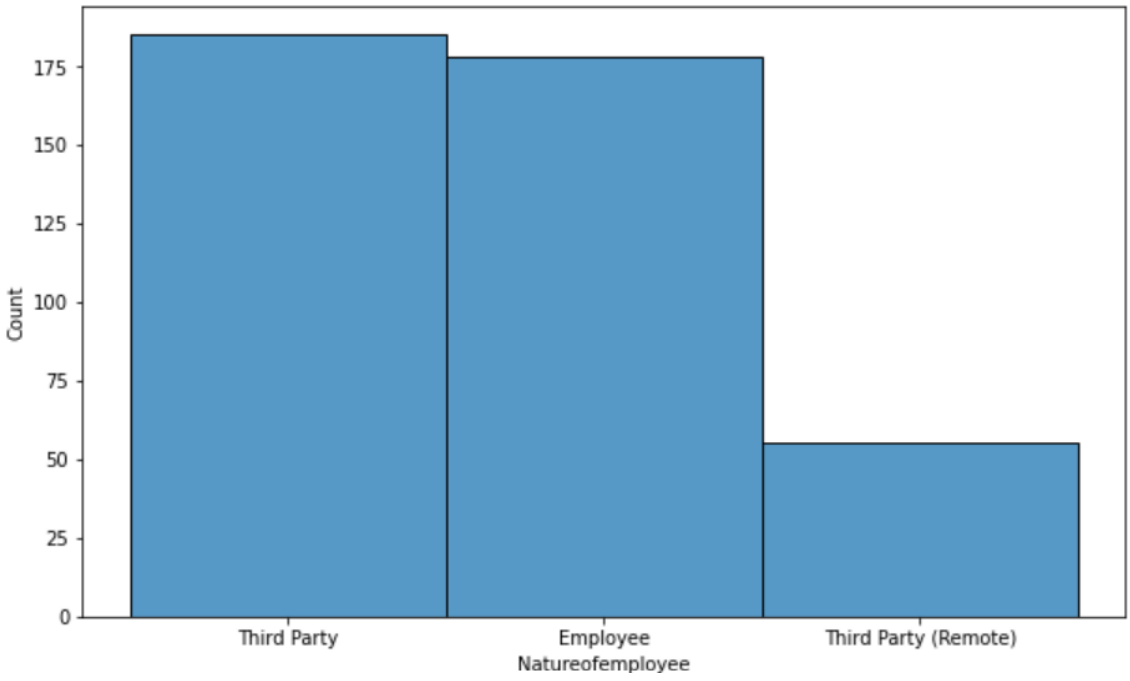


OBSERVATIONS-

It can be noted that the third party employee has happened to face most of the percentage of accidents. i.e- 44.3%.

HISTOGRAM-

It is used to show the distribution of the nature of employee such as third party, employee and third party(remote).

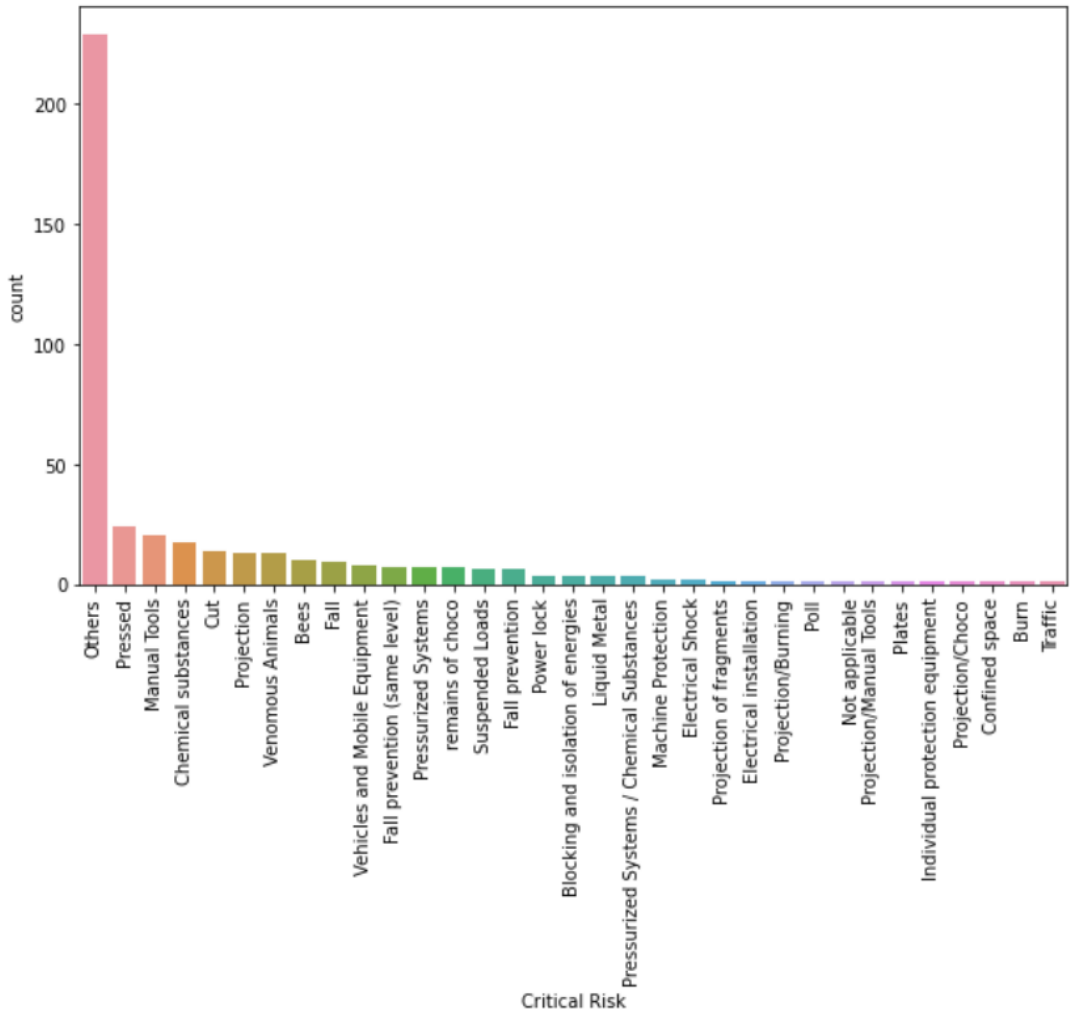


7. CRITICAL RISK-

It is used to determine the number of accidents caused by different critical risks such as pressed, pressurized system, electric shock and others.

COUNT PLOT-

The countplot() method is used to give the count of the number of accidents caused by different critical risks

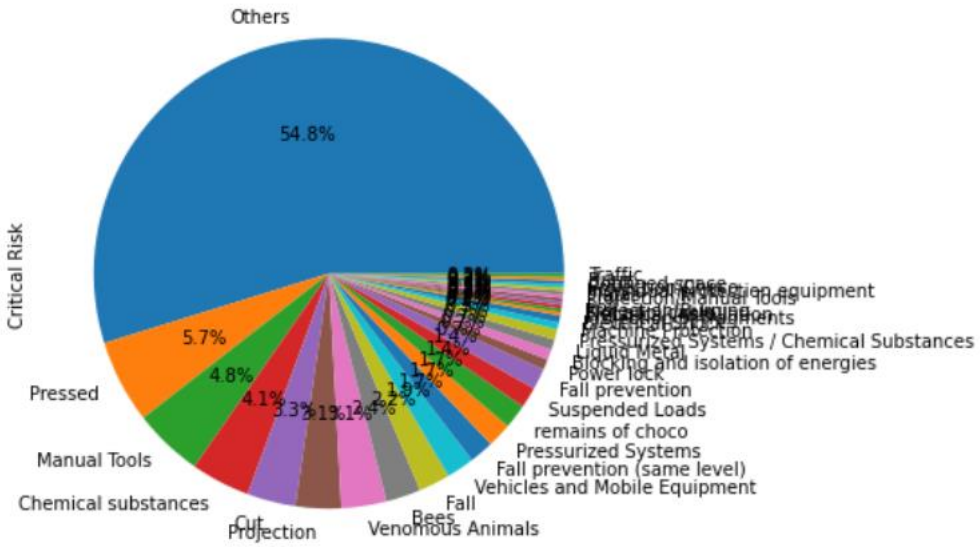


OBSERVATONS-

It can be noted that the maximum number of accidents have taken place with the critical risk “others” and minimum number of accidents took place with risk poll or traffic.

PIE CHART-

It is used to show the distribution of the different critical risks.

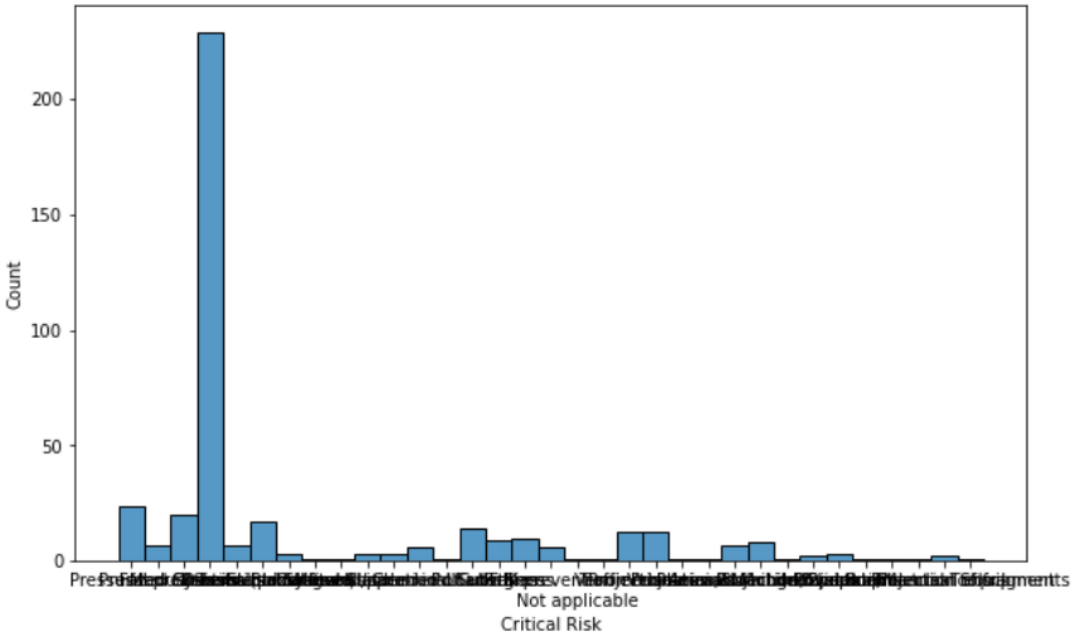


OBSERVATIONS –

It can be noted that most of the percentage of accidents have taken place with the critical risk “others”.

HISTPLOT-

It is used to show the distribution of the critical risks.

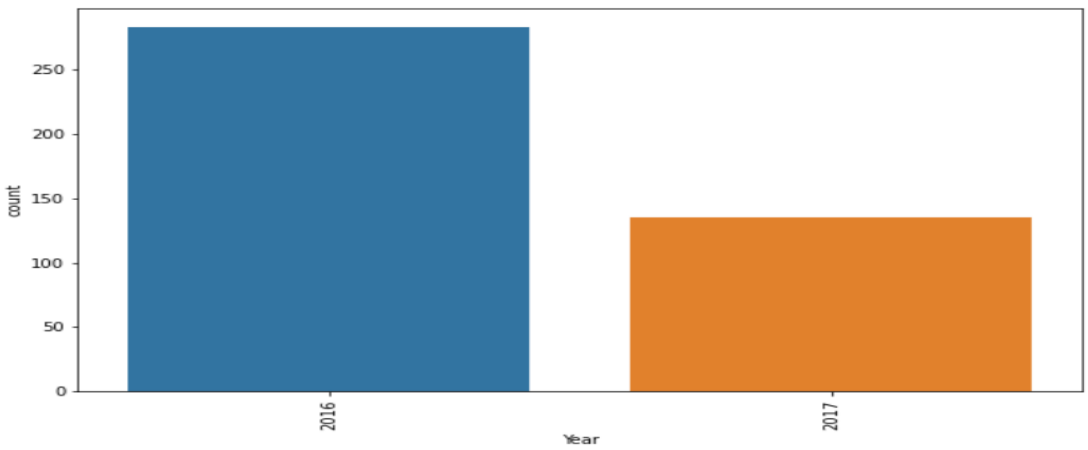


```
-----------------
Value Counts for feature:  Critical Risk
-------------------
Others                                        229
Pressed                                        24
Manual Tools                                   20
Chemical substances                            17
Cut                                            14
Projection                                     13
Venomous Animals                               13
Bees                                           10
Fall                                            9
Vehicles and Mobile Equipment                   8
Fall prevention (same level)                    7
Pressurized Systems                             7
remains of choco                                7
Suspended Loads                                 6
Fall prevention                                 6
Power lock                                      3
Blocking and isolation of energies              3
Liquid Metal                                    3
Pressurized Systems / Chemical Substances       3
Machine Protection                              2
Electrical Shock                                2
Burn                                            1
Projection of fragments                         1
Electrical installation                         1
Projection/Burning                              1
Poll                                            1
\nNot applicable                                1
Projection/Manual Tools                         1
Plates                                          1
Individual protection equipment                 1
Projection/Choco                                1
Confined space                                  1
Traffic                                         1
Name: Critical Risk, dtype: int64
```

8. YEAR-

The year attribute is used to determine the number of accidents that had happened according to the years .i.e- 2016 and 2017.

COUNT PLOT- The countplot() method is used to predict the number of accidents taken place in 2016 and 2017

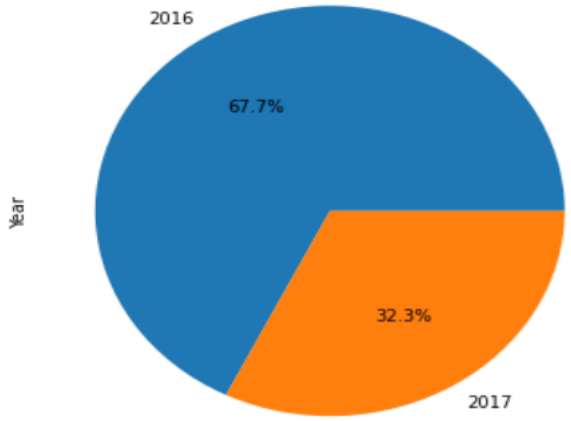


OBSERVATIONS –

It can be noted that the number of accidents that happened in 2016 (above 250) are higher than 2017(150).

PIE CHART-

It is used to show the percentage of accidents that had taken place according to the years.

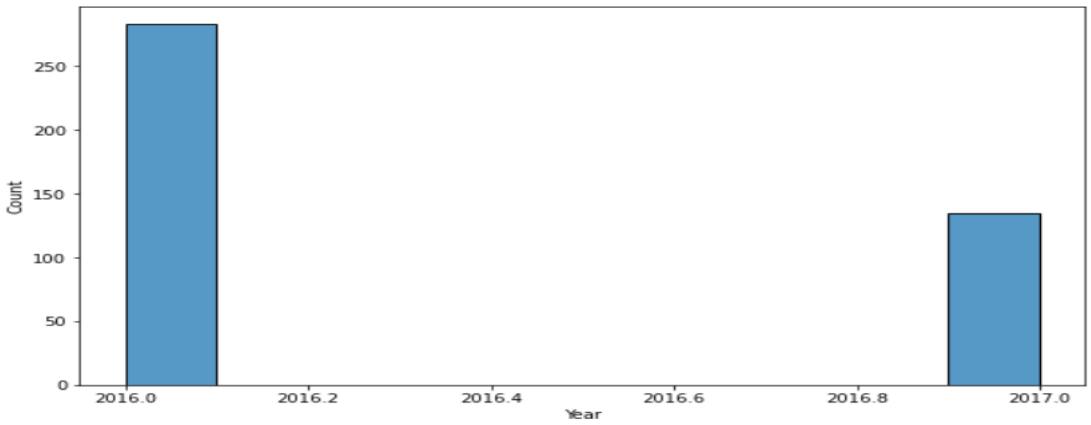


OBSERVATIONS-

It can be noted that the percentage of accidents occurred in 2016 are maximum

HISTPLOT-

It shows the distribution of the accidents that had taken place according to the years



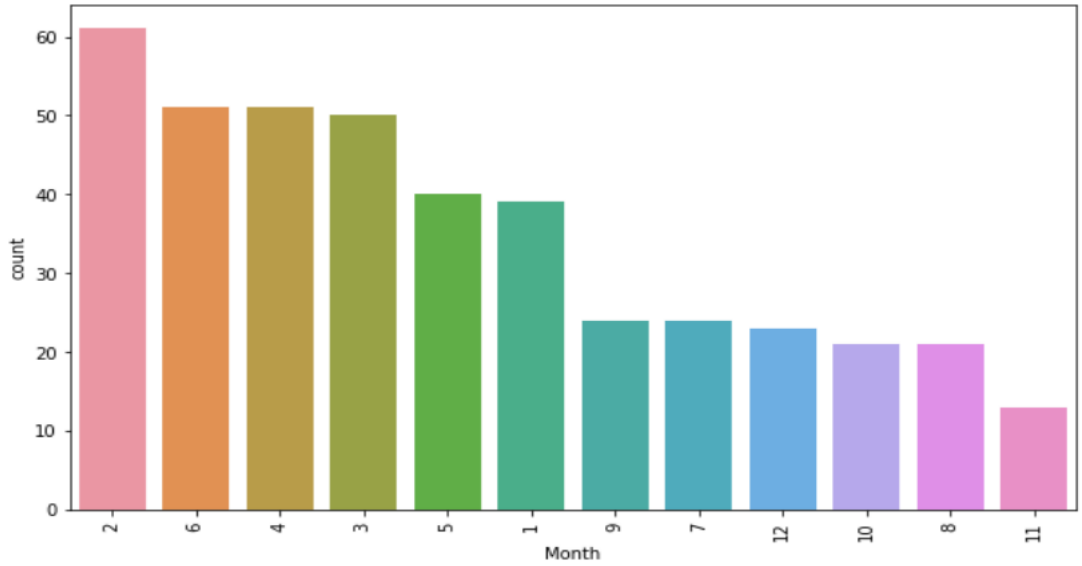

```
------------------
Value Counts for feature:  Year
--------------------
2016    283
2017    135
Name: Year, dtype: int64
```

10. MONTH-

It is used to determine the number of accidents that had occurred in all the months of the years.

COUNTPLOT-

It is used to determine the number of accidents that had occurred according to their months.

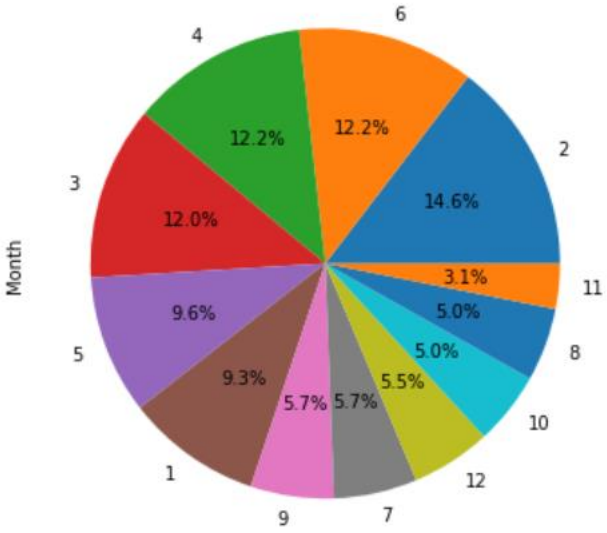


OBSERVATIONS –

It can be noted that the maximum number of accidents happened in Feb and minimum number of accidents occurred in November.

PIECHART-

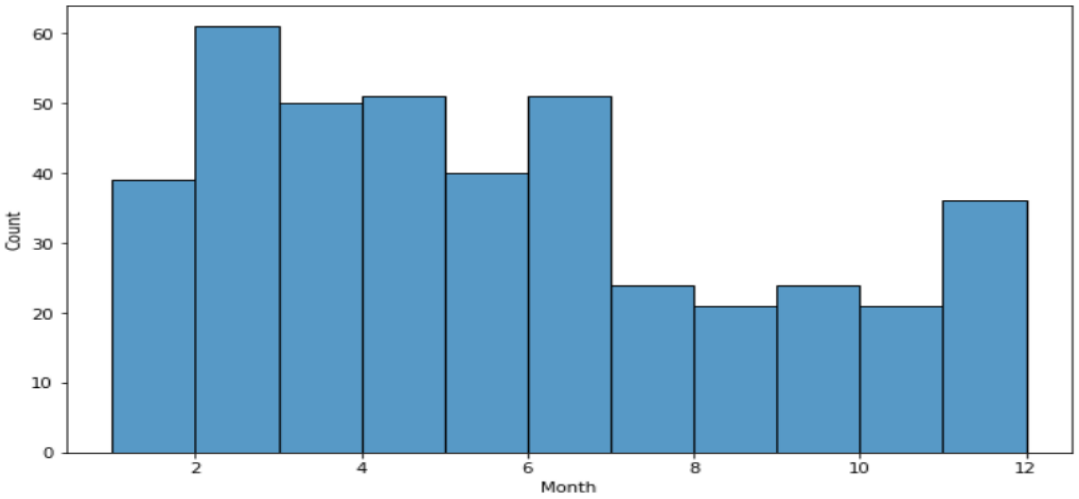
It is used to show the number of accidents that had happened in all the months in terms of percentage.



OBSERVATION- It can be noted from the above piechart that the maximum percentage of accidents happened in the month February.

HISTOGRAM-

The histplot is used to determine the distribution of accidents according to the months.

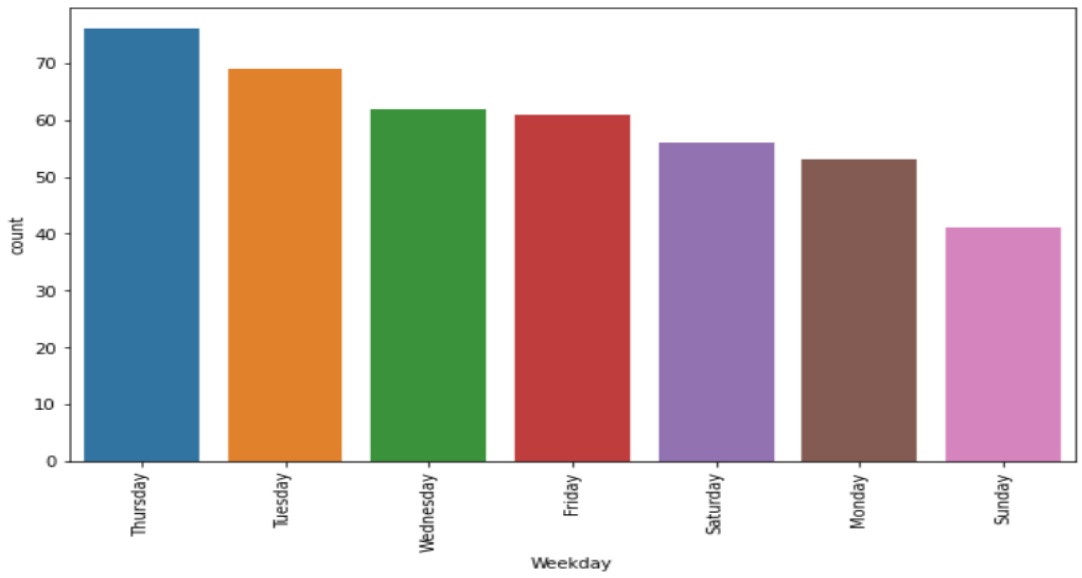


11. WEEKDAY-

It is used to determine the number of accidents on the different days of a week.

COUNTPLOT-

The countplot() method is used to determine the number of accidents that had occurred in the days of the week.

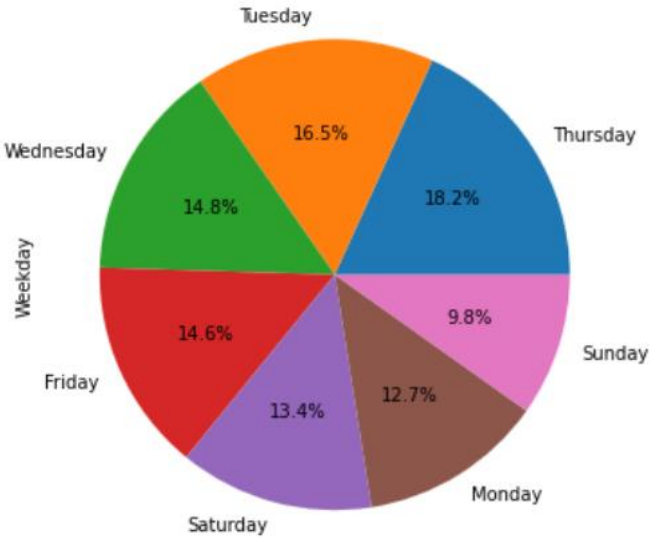


OBSERVATIONS –

It can be determined that maximum number of accidents have taken place on Thursday with a count of about 76 and least number of accidents took place on Sunday.

PIECHART-

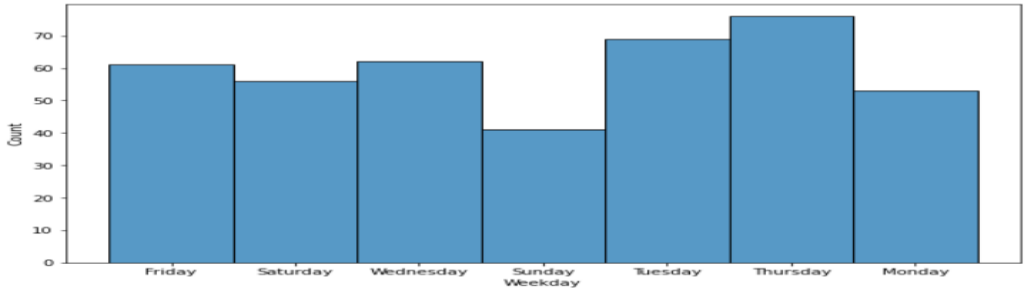
It is used to determine the percentage of accidents that took place in days of a week.



OBSERVATION – It can be observed that maximum percentage of accidents happened on Thursday with 16.5% and least number of accidents happened on Sunday.

HISTPLOT-

It is used to determine the distribution of accidents according to the days.




```
--------------------
Value Counts for feature:  Weekday
--------------------
Thursday      76
Tuesday       69
Wednesday     62
Friday        61
Saturday      56
Monday        53
Sunday        41
Name: Weekday, dtype: int64
```

OBSERVATION –

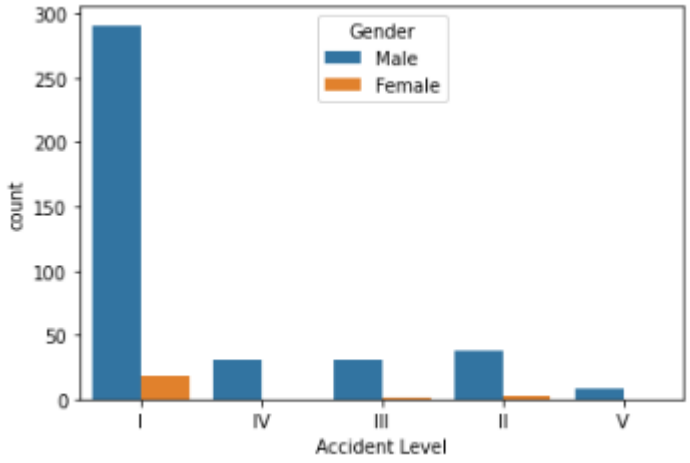
It can be noted that maximum number of accidents have taken place on Thursday

BIVARIATE ANALYSIS:

1 Gender vs RestAll

4.1 Gender vs Accident level

Count Plot:



Cross table Analysis:

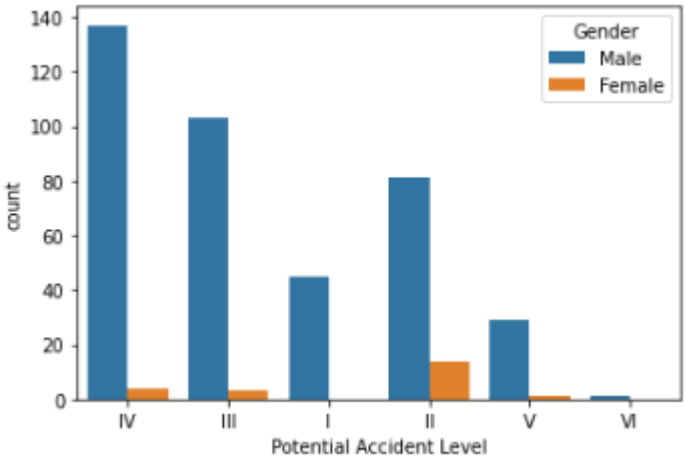
Gender	Female	Male
Accident Level		
I	18	291
II	3	37
III	1	30
IV	0	30
V	0	8

Observation –

From the above plots, it can be determined that the most of the accidents happened at level I with gender male.

4.2 Gender Vs Potential Accident Level

Count Plot:



Cross table Analysis:

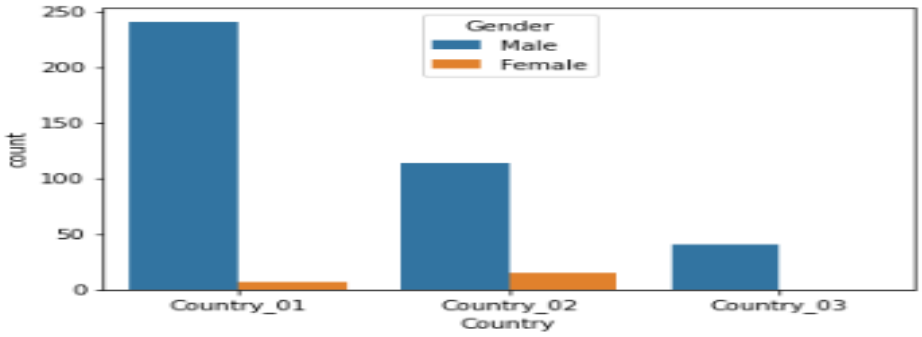
Gender	Female	Male
Potential Accident Level		
I	0	45
II	14	81
III	3	103
IV	4	137
V	1	29
VI	0	1

Observation –

From the above, it can be determined that most of the potential level accidents happened to male compared to female, of which Potential Accident Level of IV is dominant.

4.3 Gender vs Country:

Count Plot:



Cross table Analysis:

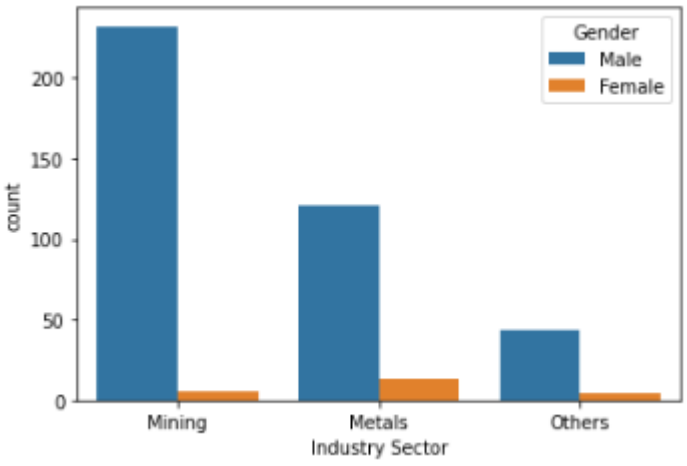
Gender	Female	Male
Country		
Country_01	7	241
Country_02	15	114
Country_03	0	41

Observation –

From the above countplot, it can be determined that the maximum number of accidents took place in country_01 to males and they are about 241.

4.4 Gender vs Industry Sector:

Count Plot:



Cross table Analysis:

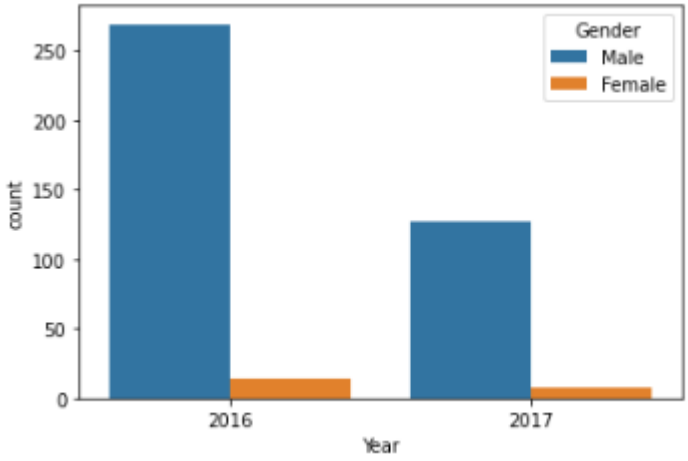
Gender	Female	Male
Industry Sector		
Metals	13	121
Mining	5	232
Others	4	43

Observation –

From the above plots, it is evident that most of the accidents happened to Male in the mining sector, around 232.

4.5 Gender vs Year

Count Plot:



Cross table Analysis:

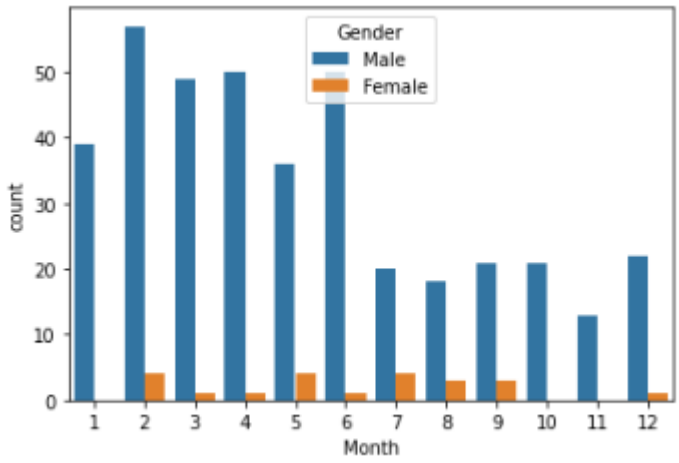
Gender	Female	Male
Year		
2016	14	269
2017	8	127

Observation –

From the above countplot, it is clearly evident that maximum accidents took place in 2016 to the male when compared to female with a count of 269.

4.6 Gender vs month:

Count Plot:



Cross table Analysis:

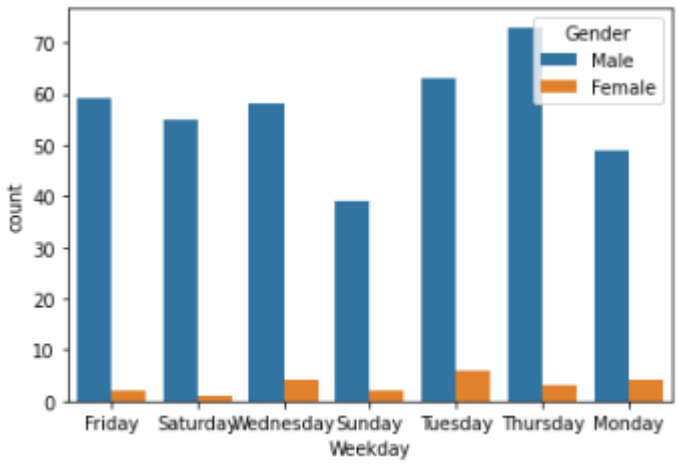
Gender	Female	Male
Month		
1	0	39
2	4	57
3	1	49
4	1	50
5	4	36
6	1	50
7	4	20
8	3	18
9	3	21
10	0	21
11	0	13
12	1	22

Observation –

From the above count plot, it is determined that maximum number of accidents happened to male in the month feb with a count 57.

4.7 Gender vs weekday:

Count Plot:



Cross table Analysis:

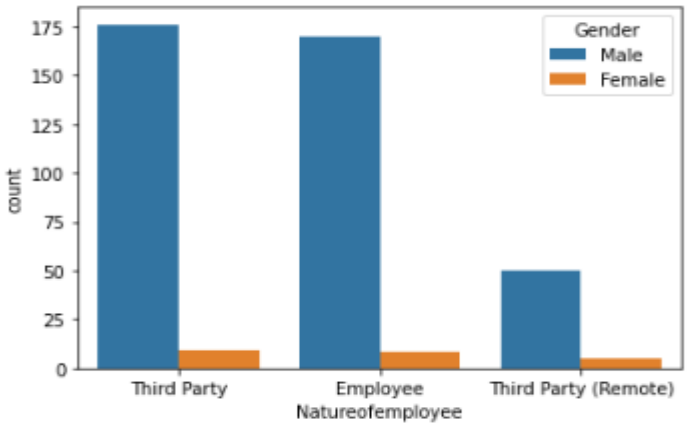
Gender	Female	Male
Weekday		
Friday	2	59
Monday	4	49
Saturday	1	55
Sunday	2	39
Thursday	3	73
Tuesday	6	63
Wednesday	4	58

Observation –

Max accidents happened to male on Thursday with a count of more than 73

4.8 Gender vs Nature of Employee:

Count Plot:



Cross table Analysis:

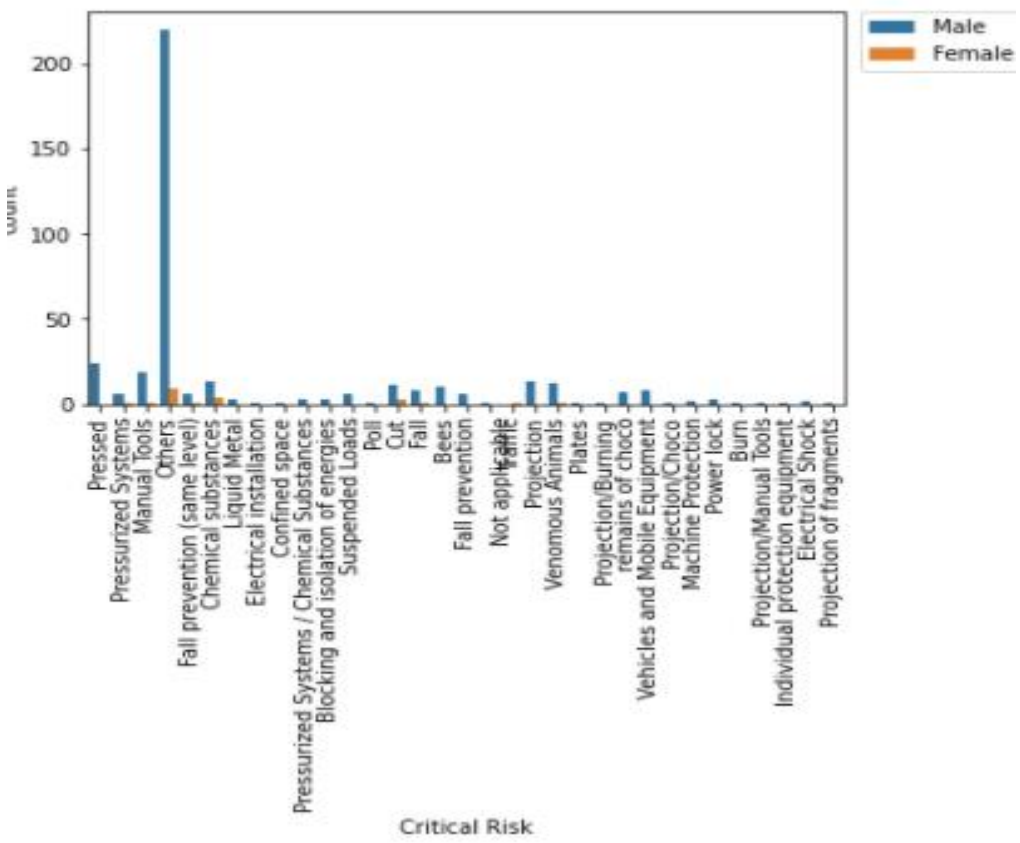
Gender	Female	Male
Natureofemployee		
Employee	8	170
Third Party	9	176
Third Party (Remote)	5	50

Observation –

From the above output, it is clearly evident that maximum accidents happened to third party male employees. i.e- 176.

4.9 Gender vs Critical Risk:

Count Plot:



Cross table Analysis:

Gender	Female	Male
Critical Risk		
\nNot applicable	0	1
Bees	0	10
Blocking and isolation of energies	0	3
Burn	0	1
Chemical substances	4	13
Confined space	0	1
Cut	3	11
Electrical Shock	0	2
Electrical installation	0	1
Fall	1	8
Fall prevention	0	6
Fall prevention (same level)	1	6
Individual protection equipment	0	1
Liquid Metal	0	3
Machine Protection	0	2
Manual Tools	1	19
Others	9	220

Plates	0	1
Poll	0	1
Power lock	0	3
Pressed	0	24
Pressurized Systems	1	6
Pressurized Systems / Chemical Substances	0	3
Projection	0	13
Projection of fragments	0	1
Projection/Burning	0	1
Projection/Choco	0	1
Projection/Manual Tools	0	1
Suspended Loads	0	6
Traffic	1	0
Vehicles and Mobile Equipment	0	8
Venomous Animals	1	12
remains of choco	0	7

Observation –

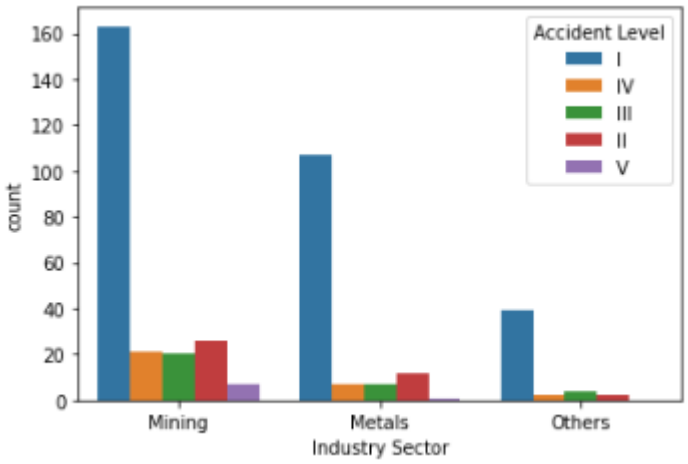
Critical Risk of type "Others" is dominant across both Male and Female Genders.

2 Industry sector vs RestAll:

5.

5.1 Industry Sector Vs Accident Level:

Count Plot:



Cross table Analysis:

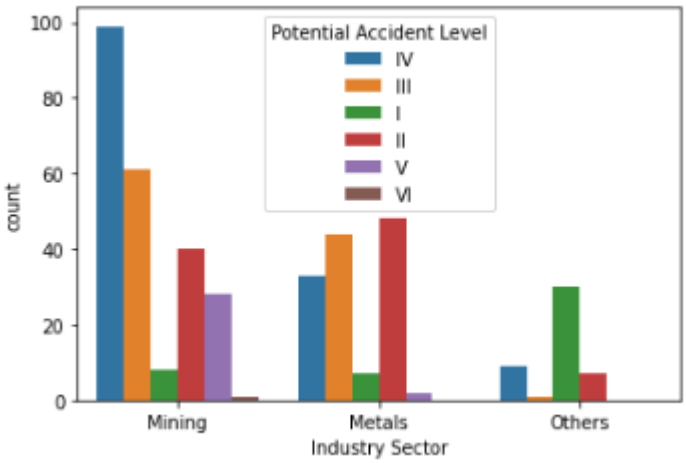
Accident Level	I	II	III	IV	V
Industry Sector					
Metals	107	12	7	7	1
Mining	163	26	20	21	7
Others	39	2	4	2	0

Observation –

Maximum number of accidents happened in the mining sector with accident Level I. i.e- 163.

5.2 Industry sector vs potential accident level

Count Plot:



Cross table Analysis:

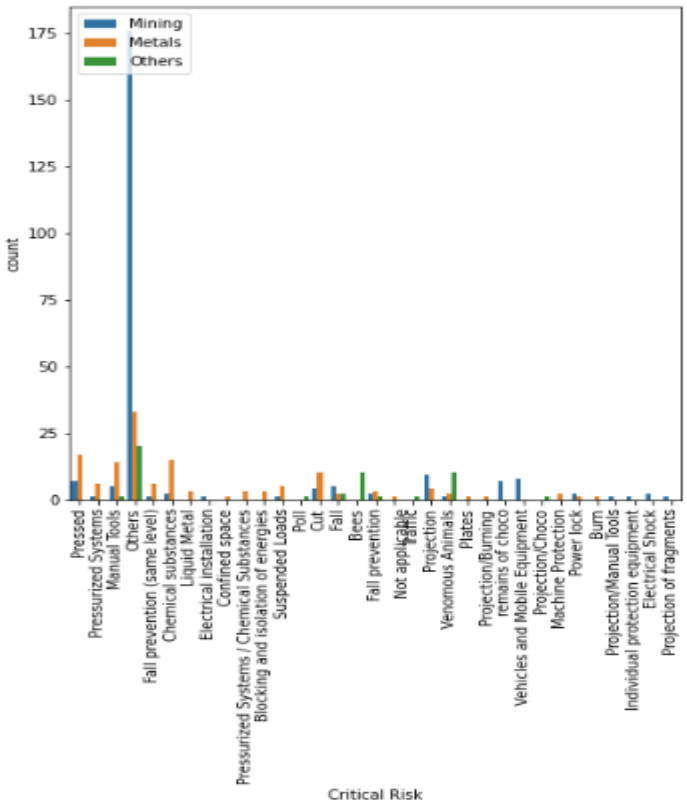
Potential Accident Level	I	II	III	IV	V	VI
Industry Sector						
Metals	7	48	44	33	2	0
Mining	8	40	61	99	28	1
Others	30	7	1	9	0	0

Observation –

Maximum number of accidents happened in the potential accident level 4 and mining sector with a count 99. Minimum number of accidents took place in the mining sector at a potential accident level 6.

5.3 Industry Sector vs Critical Risk¶

Count Plot:



Cross table Analysis:

Industry Sector	Metals	Mining	Others
Critical Risk			
\nNot applicable	1	0	0
Bees	0	0	10
Blocking and isolation of energies	3	0	0
Burn	1	0	0
Chemical substances	15	2	0
Confined space	1	0	0
Cut	10	4	0
Electrical Shock	0	2	0
Electrical installation	0	1	0
Fall	2	5	2
Fall prevention	3	2	1
Fall prevention (same level)	6	1	0
Individual protection equipment	0	1	0
Liquid Metal	3	0	0
Machine Protection	2	0	0
Manual Tools	14	5	1
Others	33	176	20
Plates	1	0	0
Poll	0	0	1
Power lock	1	2	0
Pressed	17	7	0
Pressurized Systems	6	1	0
Pressurized Systems / Chemical Substances	3	0	0
Projection	4	9	0

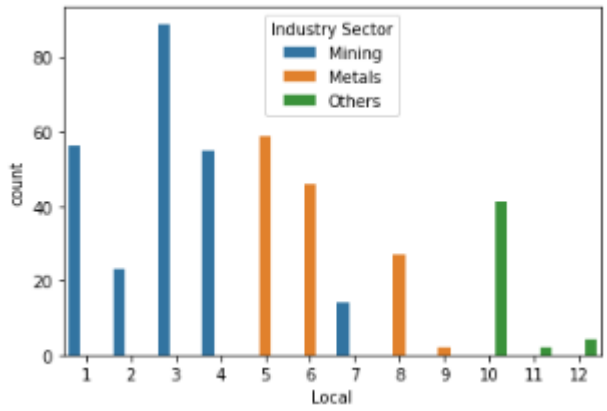
Projection of fragments	0	1	0
Projection/Burning	1	0	0
Projection/Choco	0	0	1
Projection/Manual Tools	0	1	0
Suspended Loads	5	1	0
Traffic	0	0	1
Vehicles and Mobile Equipment	0	8	0
Venomous Animals	2	1	10
remains of choco	0	7	0

Observation –

From the above count plot, it is evident that maximum number of accidents happened in mining with a critical risk of others. i.e about 175.

5.4 Industry sector vs Local:

Count Plot:



Cross table Analysis:

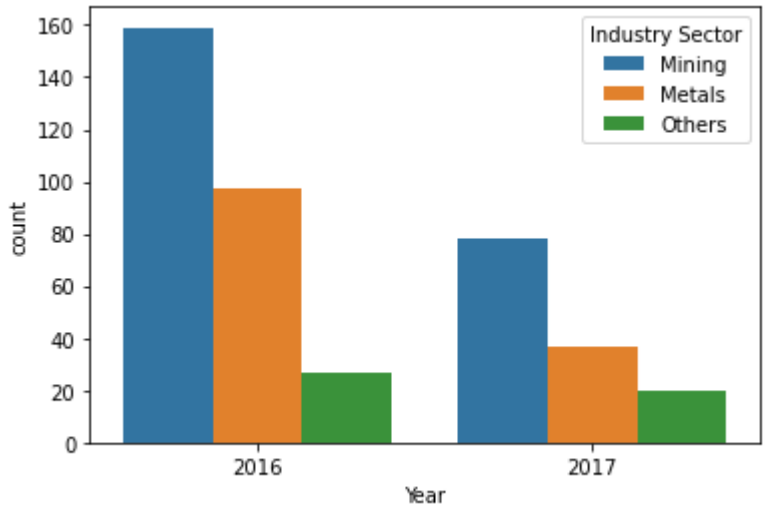
Local	1	2	3	4	5	6	7	8	9	10	11	12
Industry Sector												
Metals	0	0	0	0	59	46	0	27	2	0	0	0
Mining	56	23	89	55	0	0	14	0	0	0	0	0
Others	0	0	0	0	0	0	0	0	0	41	2	4

Observation –

Many accidents happened with a local 3 and industrial sector mining. i.e- more than 80. Least accidents took place with local 11 and industrial sector others.

5.5 Industry Sector Vs Year:

Count Plot:



Cross table Analysis:

Year	2016	2017
Industry Sector		
Metals	97	37
Mining	159	78
Others	27	20

Observation –

From the above plot, the following could be determined

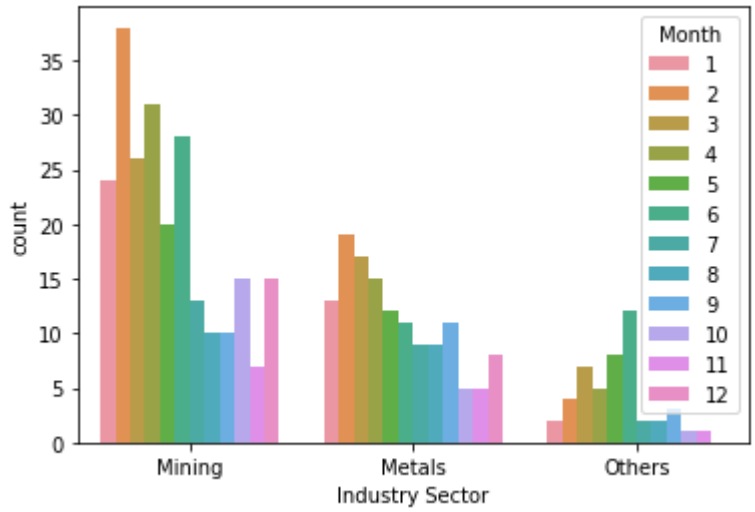
- The number of accidents taken place in year 2016 for mining sector is 160.
- The number of accidents taken place in year 2016 wrt metals sector is about 100.
- The number of accidents taken place in the year 2016 wrt others sector is about 30. Hence, it can be determined that maximum accidents took place in mining sector in the year 2016.
- The number of accidents taken place in the year 2017 wrt mining sector is 80.
- The number of accidents taken place in the year 2017 wrt metals sector is about 40.

- The number of accidents taken place in the year 2017 wrt others sector is 20.

Hence, it can be determined that max accidents took place in mining sector in the year 2017.

5.6 Industry Sector Vs Month:

Count Plot:



Cross table Analysis:

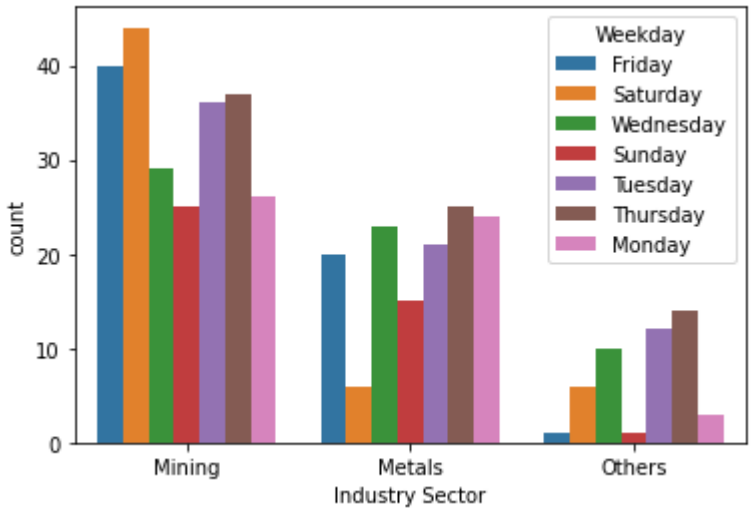
Month	1	2	3	4	5	6	7	8	9	10	11	12
Industry Sector												
Metals	13	19	17	15	12	11	9	9	11	5	5	8
Mining	24	38	26	31	20	28	13	10	10	15	7	15
Others	2	4	7	5	8	12	2	2	3	1	1	0

Observation –

Maximum number of accidents happened in the month feb and mining sector. The least number of accidents took place in the others sector and month December.

5.7 Industry sector vs weekday:

Count Plot:



Cross table Analysis:

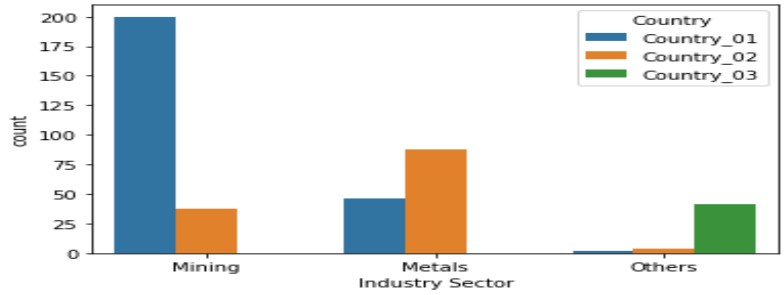
Weekday	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday
Industry Sector							
Metals	20	24	6	15	25	21	23
Mining	40	26	44	25	37	36	29
Others	1	3	6	1	14	12	10

Observation –

Maximum number of accidents happened on the day Saturday in the mining sector. i.e.- more than 40. The least number of accidents happened on the day Sunday in the others sector.

5.8 Industry sector vs Country:

Count Plot:



Cross table Analysis:

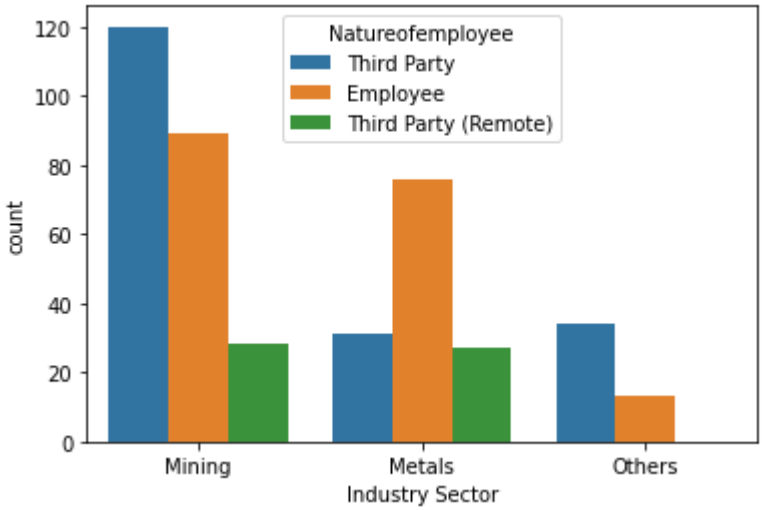
Country	Country_01	Country_02	Country_03
Industry Sector			
Metals	46	88	0
Mining	200	37	0
Others	2	4	41

Observation –

From the above count plot, it is evident that the maximum number of accidents took place in country_01 and mining sector i.e.- 200. The least number of accidents took place in country _01 and others sector.

5.9 Industry Sector Vs nature of employee:

Count Plot:



Cross table Analysis:

Natureofemployee	Employee	Third Party	Third Party (Remote)
Industry Sector			
Metals	76	31	27
Mining	89	120	28
Others	13	34	0

Observation –

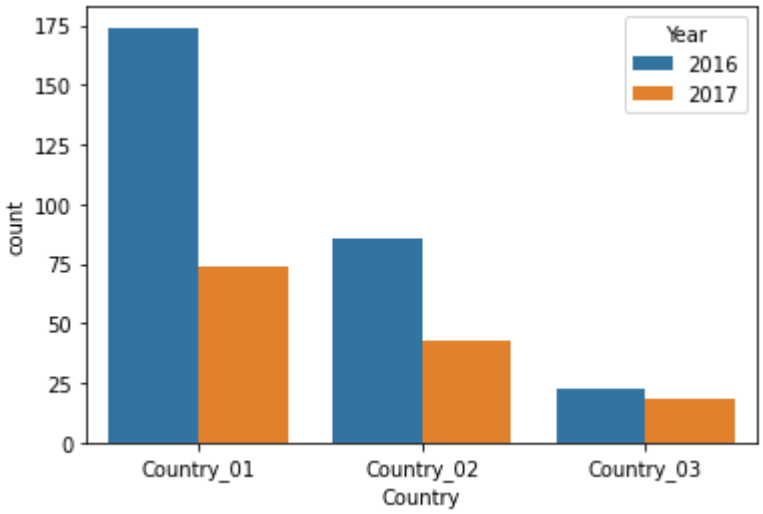
From the above count plot, it is clearly evident that the maximum accidents took place in the mining sector with the third-party employee type. i.e.- about 120. The least number of accidents took place in the others sectors with the nature of employee as employee.

3 Country vs RestAll:

6.

6.1 Country vs Year

Count Plot:



Cross table Analysis:

Year	2016	2017
Country		
Country_01	174	74
Country_02	86	43
Country_03	23	18

Observation –

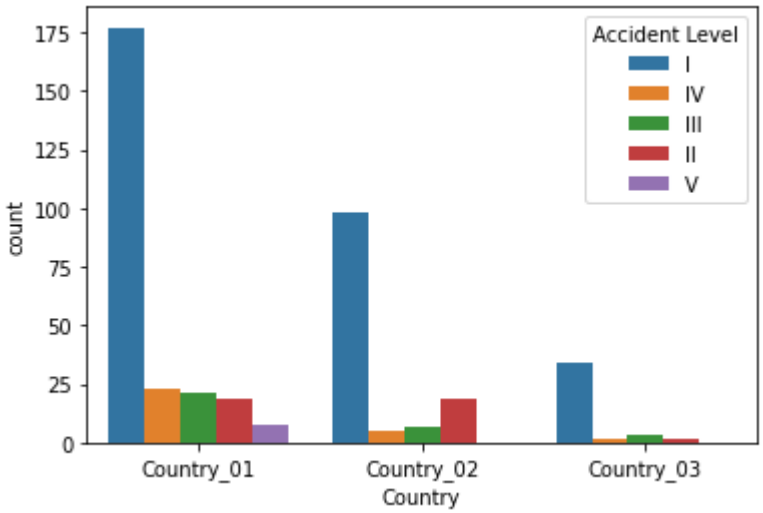
From the above output, the following can be determined-

- The number of accidents taken place in country_01 and year 2016 is 174.
- The number of accidents taken place in country_01 and year 2017 is about 74.

- The number of accidents taken place in country_02 and year 2016 is more than 86.
- The number of accidents taken place in country_02 and year 2017 is about 43.
- The number of accidents taken place in country_03 and year 2016 is about 23.
- The number of accidents taken place in country_03 and year 2017 is about 18.

6.2 Country Vs accident level:

Count Plot:



Cross table Analysis:

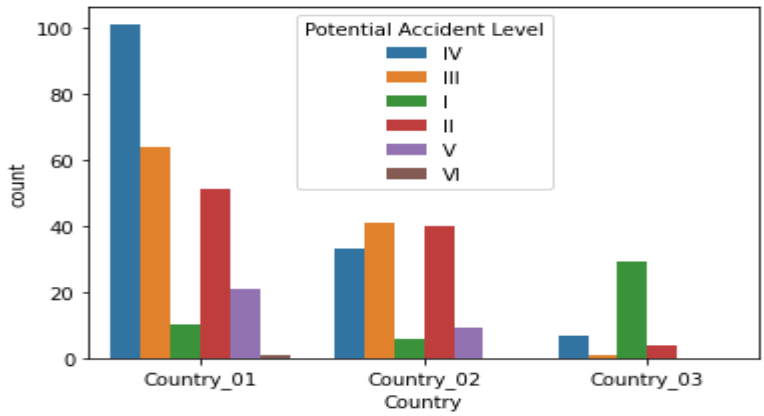
Accident Level	I	II	III	IV	V
Country					
Country_01	177	19	21	23	8
Country_02	98	19	7	5	0
Country_03	34	2	3	2	0

Observation –

From the above count plot, it is clearly evident that the maximum number of accidents took place in accident level 1 and country_01.

6.3 Country Vs Potential Accident Level:

Count Plot:



Cross table Analysis:

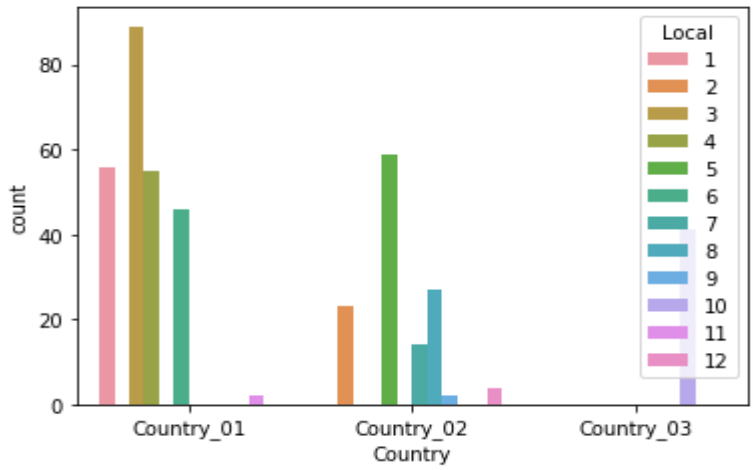
Potential Accident Level	I	II	III	IV	V	VI
Country						
Country_01	10	51	64	101	21	1
Country_02	6	40	41	33	9	0
Country_03	29	4	1	7	0	0

Observation –

From the above plot, it is evident that the maximum accidents occurred in country_01 and potential accident level 3

6.4 Country Vs Local:

Count Plot:



Cross table Analysis:

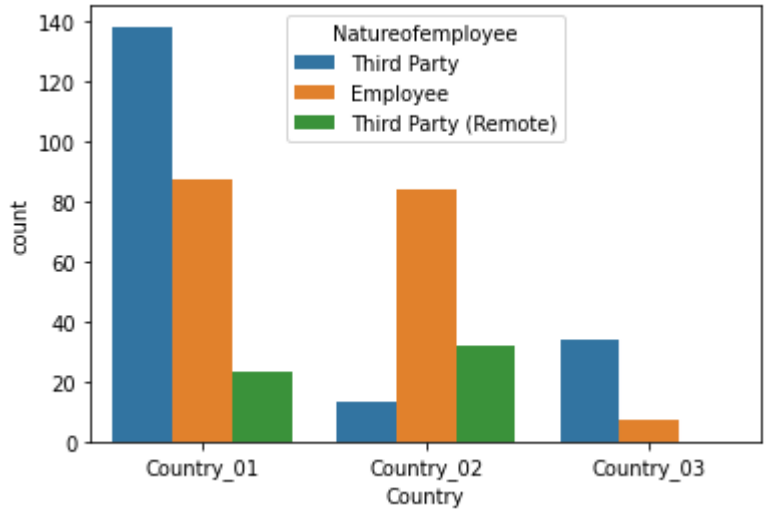
Local	1	2	3	4	5	6	7	8	9	10	11	12
Country												
Country_01	56	0	89	55	0	46	0	0	0	0	2	0
Country_02	0	23	0	0	59	0	14	27	2	0	0	4
Country_03	0	0	0	0	0	0	0	0	0	41	0	0

Observation –

Country 1 is more dominant in local 3 region and least dominant in Local 12.

6.5 Country Vs Nature of Employee

Count Plot:



Cross table Analysis:

Natureofemployee	Employee	Third Party	Third Party (Remote)
Country			
Country_01	87	138	23
Country_02	84	13	32
Country_03	7	34	0

Observation –

Accidents in Country 01 is more dominant in Third Party type of employee, country 03 is least dominant in Third Party (Remote).

6.6 Country Vs Critical Risk:

Cross table Analysis:

Country / Critical Risk	Country 01	Country 02	Country 03
\nNot applicable	0	1	0
Bees	0	0	10
Blocking and isolation of energies	1	2	0
Burn	0	1	0
Chemical substances	4	13	0
Confined space	0	1	0
Cut	5	9	0
Electrical Shock	2	0	0
Electrical installation	1	0	0
Fall	6	1	2
Fall prevention	3	2	1
Fall prevention (same level)	5	2	0
Individual protection equipment	1	0	0
Liquid Metal	0	3	0
Machine Protection	1	1	0
Manual Tools	7	12	1
Others	169	45	15
Plates	1	0	0
Poll	0	0	1
Power lock	3	0	0
Pressed	9	15	0
Pressurized Systems	1	6	0
Pressurized Systems / Chemical Substances	2	1	0
Projection	9	4	0
Projection of fragments	1	0	0
Projection/Burning	0	1	0
Projection/Choco	0	0	1
Projection/Manual Tools	1	0	0
Suspended Loads	3	3	0
Traffic	0	1	0
Vehicles and Mobile Equipment	7	1	0
Venomous Animals	0	3	10
remains of choco	6	1	0

Observation –

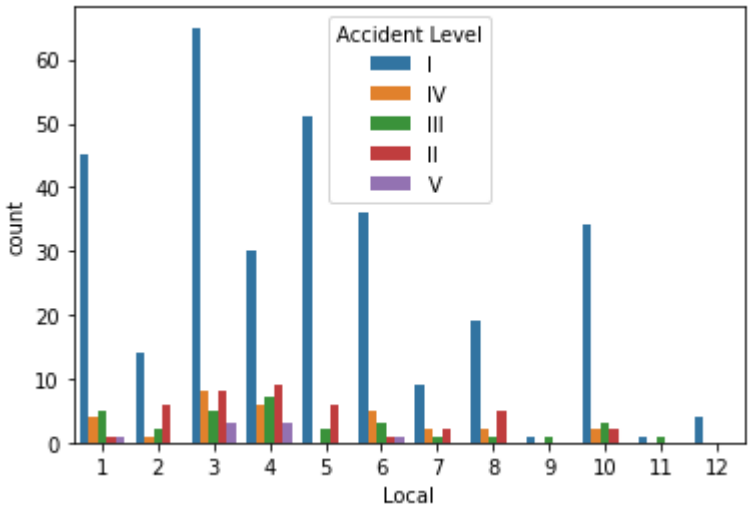
Country 01 is more dominant in Others Critical Risk and Critical Risk is least dominant in Country 03.

4 Local Vs Rest All

7.

7.1 Local Vs Accident Level

Count Plot:



Cross table Analysis:

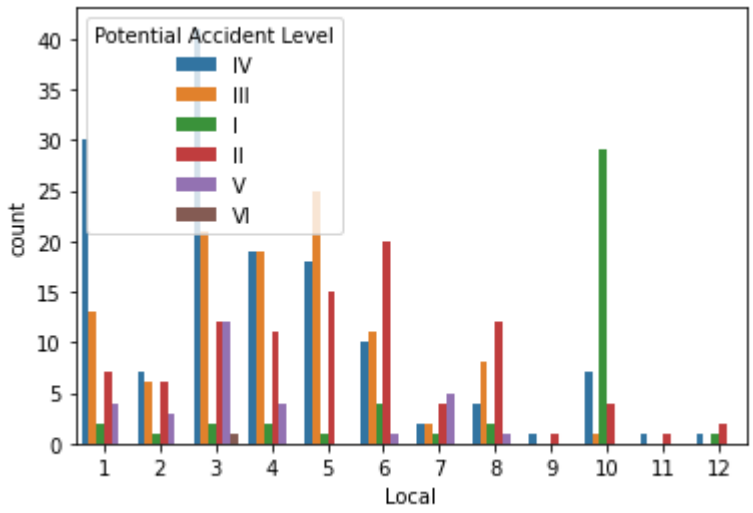
Local	1	2	3	4	5	6	7	8	9	10	11	12
Accident Level												
I	45	14	65	30	51	36	9	19	1	34	1	4
II	1	6	8	9	6	1	2	5	0	2	0	0
III	5	2	5	7	2	3	1	1	1	3	1	0
IV	4	1	8	6	0	5	2	2	0	2	0	0
V	1	0	3	3	0	1	0	0	0	0	0	0

Observation –

Accident level 1 is more dominant in Local 2 region with 65 accidents, while Accident Level V is least across all Locals

7.2 Local Vs Potential Accident Level:

Count Plot:



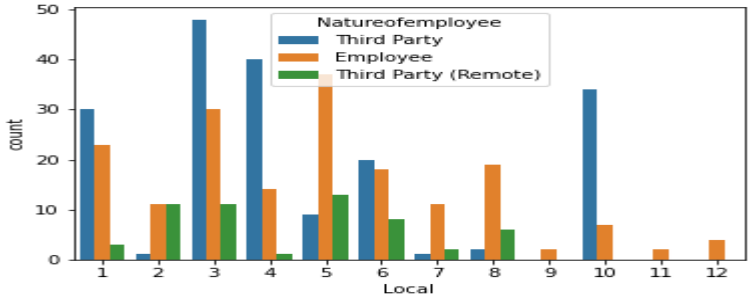
Cross table Analysis:

Local	1	2	3	4	5	6	7	8	9	10	11	12
Potential Accident Level												
I	2	1	2	2	1	4	1	2	0	29	0	1
II	7	6	12	11	15	20	4	12	1	4	1	2
III	13	6	21	19	25	11	2	8	0	1	0	0
IV	30	7	41	19	18	10	2	4	1	7	1	1
V	4	3	12	4	0	1	5	1	0	0	0	0
VI	0	0	1	0	0	0	0	0	0	0	0	0

Observation: Overall Local 3 is more prone to Multiple potential accidents, while local 12 is the least.

7.3 Local Vs Natureofemployee:

Count Plot:



Cross table Analysis:

Natureofemployee \ Local	1	2	3	4	5	6	7	8	9	10	11	12
Employee	23	11	30	14	37	18	11	19	2	7	2	4
Third Party	30	1	48	40	9	20	1	2	0	34	0	0
Third Party (Remote)	3	11	11	1	13	8	2	6	0	0	0	0

Observation –

Type Employee is more dominant across all Locals, while Type Third Party(Remote) is least dominant across all Locals.

7.4 Local Vs Critical Risk:

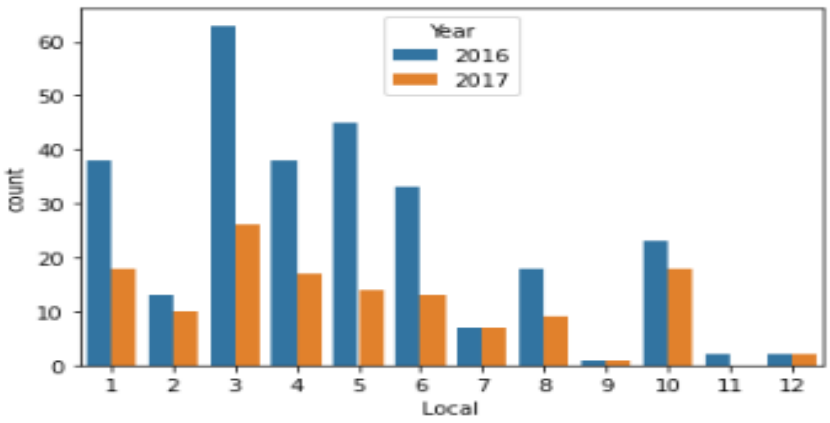
Cross table Analysis:

Critical Risk \ Local	1	2	3	4	5	6	7	8	9	10	11	12
\nNot applicable	0	0	0	0	0	0	0	1	0	0	0	0
Bees	0	0	0	0	0	0	0	0	0	10	0	0
Blocking and isolation of energies	0	0	0	0	1	1	0	1	0	0	0	0
Burn	0	0	0	0	1	0	0	0	0	0	0	0
Chemical substances	1	1	0	0	11	3	0	1	0	0	0	0
Confined space	0	0	0	0	1	0	0	0	0	0	0	0
Cut	0	0	2	1	7	2	1	1	0	0	0	0
Electrical Shock	0	0	2	0	0	0	0	0	0	0	0	0
Electrical installation	1	0	0	0	0	0	0	0	0	0	0	0
Fall	2	0	1	2	1	1	0	0	0	2	0	0
Fall prevention	0	0	1	0	1	2	1	0	0	1	0	0
Fall prevention (same level)	0	0	0	1	2	4	0	0	0	0	0	0
Individual protection equipment	0	0	1	0	0	0	0	0	0	0	0	0
Liquid Metal	0	0	0	0	2	0	0	1	0	0	0	0
Machine Protection	0	0	0	0	1	1	0	0	0	0	0	0
Manual Tools	1	2	1	1	5	4	0	4	1	1	0	0
Others	41	16	68	43	10	15	8	8	0	15	2	3
Plates	0	0	0	0	0	1	0	0	0	0	0	0
Poll	0	0	0	0	0	0	0	0	0	1	0	0
Power lock	0	0	2	0	0	1	0	0	0	0	0	0
Pressed	2	2	1	2	6	4	0	7	0	0	0	0
Pressurized Systems	0	1	0	0	3	1	0	2	0	0	0	0
Pressurized Systems / Chemical Substances	0	0	0	0	1	2	0	0	0	0	0	0
Projection	3	0	4	0	1	2	2	1	0	0	0	0
Projection of fragments	0	0	0	1	0	0	0	0	0	0	0	0
Projection/Burning	0	0	0	0	1	0	0	0	0	0	0	0
Projection/Choco	0	0	0	0	0	0	0	0	0	1	0	0
Projection/Manual Tools	0	0	0	1	0	0	0	0	0	0	0	0
Suspended Loads	1	0	0	0	3	2	0	0	0	0	0	0
Traffic	0	0	0	0	0	0	0	0	0	0	0	1
Vehicles and Mobile Equipment	4	1	2	1	0	0	0	0	0	0	0	0
Venomous Animals	0	0	0	0	1	0	1	0	1	10	0	0

Observation: Critical Risk of type "Others" is dominant across all Locals

7.5 Local Vs Year

Count Plot:



Cross table Analysis:

Local	1	2	3	4	5	6	7	8	9	10	11	12
Year												
2016	38	13	63	38	45	33	7	18	1	23	2	2
2017	18	10	26	17	14	13	7	9	1	18	0	2

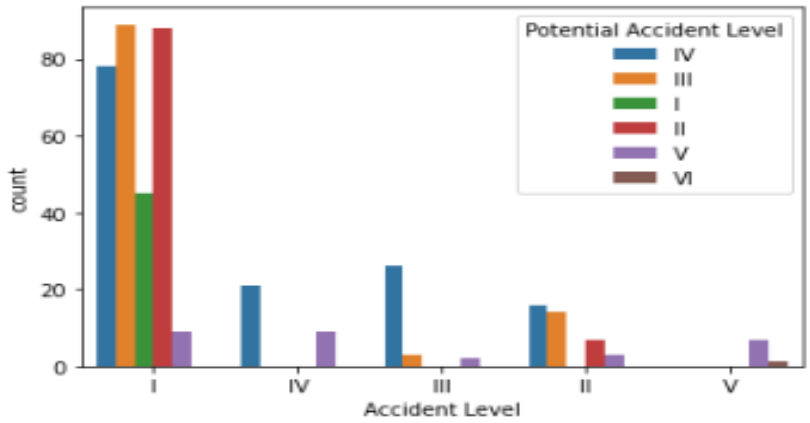
Observation: Year 2016 has more accidents across all Local regions compared to 2017.

5 Accident Level Vs Rest All:

8.

8.1 Accident Level Vs Potential Accident Level:

Count Plot:



Cross table Analysis:

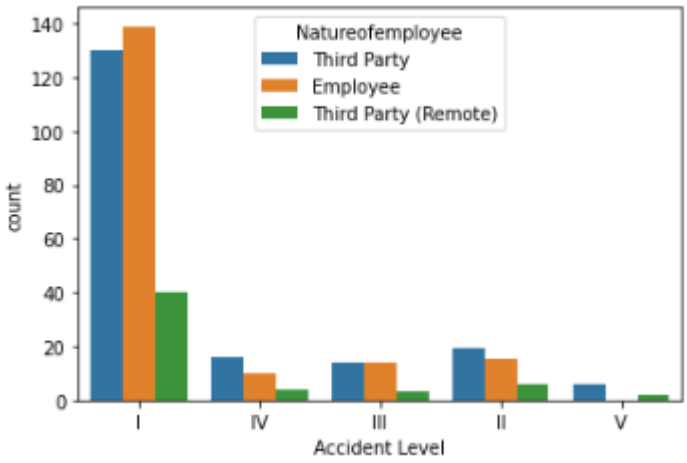
Accident Level	I	II	III	IV	V
Potential Accident Level					
I	45	0	0	0	0
II	88	7	0	0	0
III	89	14	3	0	0
IV	78	16	26	21	0
V	9	3	2	9	7
VI	0	0	0	0	1

Observation –

Accident Level I is more related to Potential Accident levels of I, II, III, IV, V, VI.

8.2 Accident Level Vs Natureofemployee:

Count Plot:



Cross table Analysis:

Natureofemployee	Employee	Third Party	Third Party (Remote)
Accident Level			
I	139	130	40
II	15	19	6
III	14	14	3
IV	10	16	4
V	0	6	2

Observation: Accident Level I is more dominant across all Employee types, where Level V is least across all types.

8.3 Accident Level Vs Critical Risk:

Cross table Analysis:

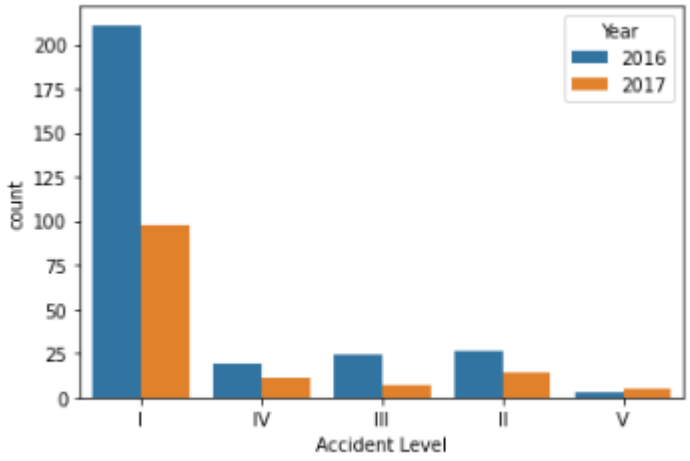
Accident Level	I	II	III	IV	V
Critical Risk					
\nNot applicable	0	0	0	1	0
Bees	10	0	0	0	0
Blocking and isolation of energies	3	0	0	0	0
Burn	0	0	1	0	0
Chemical substances	15	2	0	0	0
Confined space	1	0	0	0	0
Cut	11	2	1	0	0
Electrical Shock	2	0	0	0	0
Electrical installation	0	0	0	1	0
Fall	6	0	0	2	1
Fall prevention	5	0	0	1	0
Fall prevention (same level)	6	0	0	1	0
Individual protection equipment	0	1	0	0	0
Liquid Metal	3	0	0	0	0
Machine Protection	2	0	0	0	0
Manual Tools	12	5	3	0	0
Others	169	21	23	13	3
Plates	1	0	0	0	0
Poll	0	0	0	1	0
Power lock	0	0	0	1	2
Pressed	17	1	2	4	0
Pressurized Systems	6	1	0	0	0
Pressurized Systems / Chemical Substances	2	1	0	0	0
Projection	10	2	0	1	0
Projection of fragments	1	0	0	0	0
Projection/Burning	0	1	0	0	0
Projection/Choco	1	0	0	0	0
Projection/Manual Tools	1	0	0	0	0
Suspended Loads	4	0	1	1	0
Traffic	1	0	0	0	0
Vehicles and Mobile Equipment	5	1	0	1	1
Venomous Animals	13	0	0	0	0

Observation –

Accident Level I is more dominant with Other critical Risk type

8.4 Accident Level Vs Year :

Count Plot:



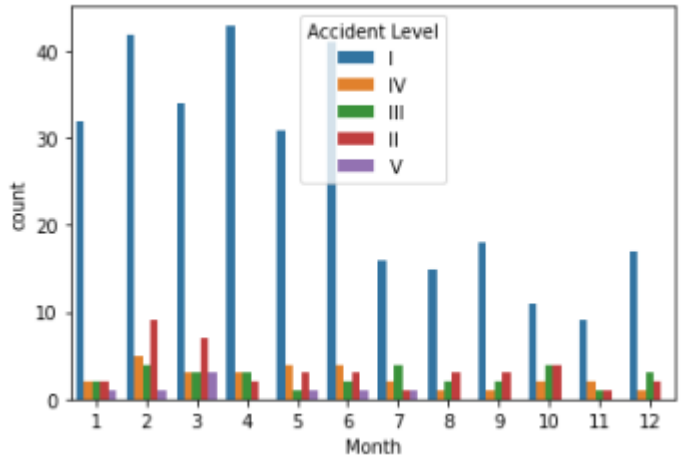
Cross table Analysis:

Accident Level	I	II	III	IV	V
Year					
2016	211	26	24	19	3
2017	98	14	7	11	5

Observation: Accident Level I is more dominant in across 2016 and 2017 years, and Level V is minimum.

8.5 Accident Level Vs Month:

Count Plot:



Cross table Analysis:

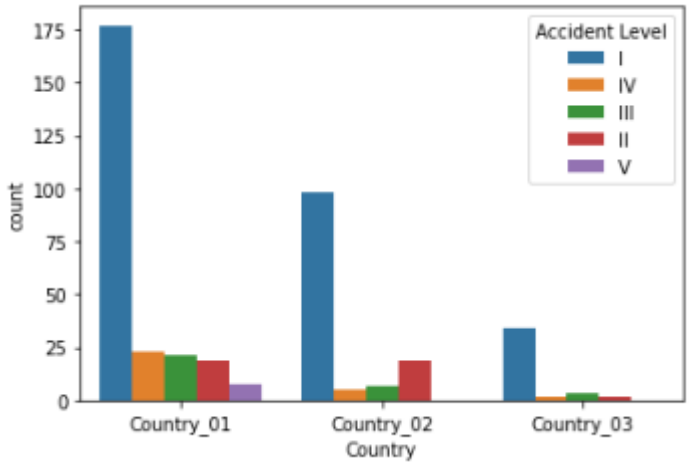
Month	1	2	3	4	5	6	7	8	9	10	11	12
Accident Level												
I	32	42	34	43	31	41	16	15	18	11	9	17
II	2	9	7	2	3	3	1	3	3	4	1	2
III	2	4	3	3	1	2	4	2	2	4	1	3
IV	2	5	3	3	4	4	2	1	1	2	2	1
V	1	1	3	0	1	1	1	0	0	0	0	0

Observation –

Accident Level 1 dominates across all Months while Level V is minimum.

8.6 Accident Level Vs Country:

Count Plot:



Cross table Analysis:

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

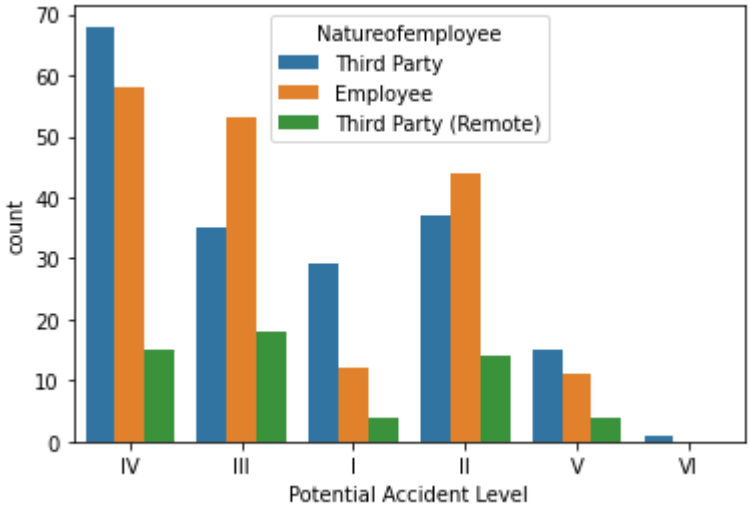
Observation: Accident Level I is more dominant across all Countries, while Accident Level V is least dominant across all countries.

6 Potential Accident Level Vs Rest All:

9.

9.1 Potential Accident Level Vs Natureofemployee:

Count Plot:



Cross table Analysis:

Natureofemployee	Employee	Third Party	Third Party (Remote)
Potential Accident Level			
I	12	29	4
II	44	37	14
III	53	35	18
IV	58	68	15
V	11	15	4
VI	0	1	0

Observation:

Potential Accident level IV dominents in ThirdParty, while VI is least dominant in Third Party(Remote) across all

9.2 Potential Accident Level Vs Critical Risk:

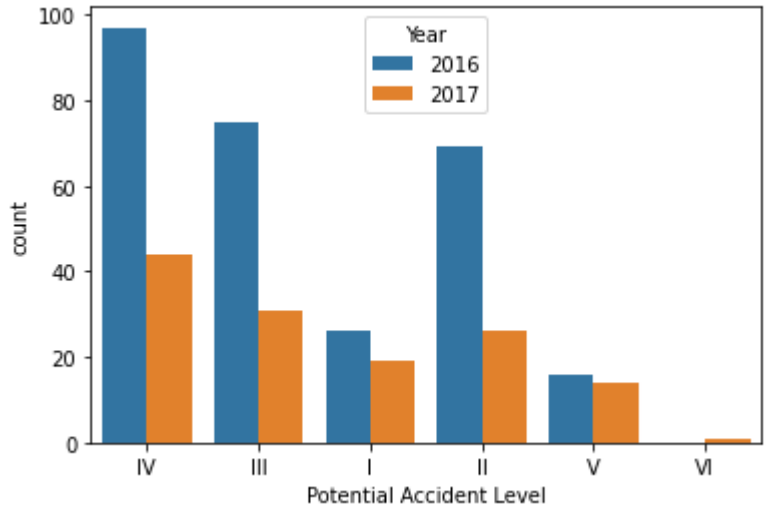
Cross table Analysis:

Potential Accident Level Critical Risk	I	II	III	IV	V	VI
\nNot applicable	0	0	0	0	1	0
Bees	10	0	0	0	0	0
Blocking and isolation of energies	0	1	2	0	0	0
Burn	0	0	0	1	0	0
Chemical substances	0	5	8	4	0	0
Confined space	0	0	1	0	0	0
Cut	1	6	5	2	0	0
Electrical Shock	0	0	0	2	0	0
Electrical installation	0	0	0	0	1	0
Fall	1	1	4	2	1	0
Fall prevention	1	0	0	5	0	0
Fall prevention (same level)	1	1	3	2	0	0
Individual protection equipment	0	0	0	1	0	0
Liquid Metal	1	0	0	2	0	0
Machine Protection	0	0	2	0	0	0
Manual Tools	2	5	9	4	0	0
Others	16	60	53	85	15	0
Plates	0	1	0	0	0	0
Poll	0	0	0	1	0	0
Power lock	0	0	0	0	3	0
Pressed	2	5	9	7	1	0
Pressurized Systems	0	2	3	2	0	0
Pressurized Systems / Chemical Substances	0	1	0	2	0	0
Projection	0	2	2	7	2	0
Projection of fragments	0	0	0	1	0	0
Projection/Burning	0	0	0	1	0	0
Projection/Choco	0	1	0	0	0	0
Projection/Manual Tools	0	0	1	0	0	0
Suspended Loads	0	1	0	5	0	0
Traffic	0	1	0	0	0	0
Vehicles and Mobile Equipment	0	0	2	2	4	0
Venomous Animals	10	2	1	0	0	0
remains of choco	0	0	1	3	2	1

Observation: Among all Critical Risk with Type as "Others" is dominant across all Potential Accident Levels.

9.3 Potential Accident Level Vs Year:

Count Plot:



Cross table Analysis:

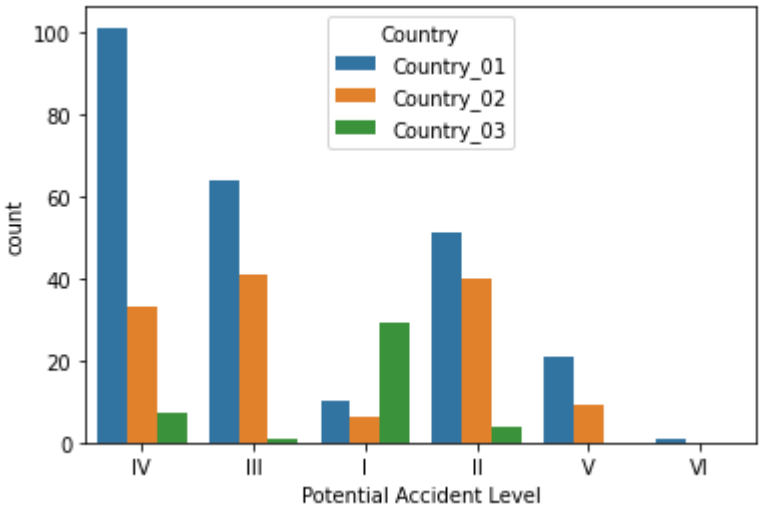
Year	2016	2017
Potential Accident Level		
I	26	19
II	69	26
III	75	31
IV	97	44
V	16	14
VI	0	1

Observation:

There is Decrease in Number of accidents across all Potential Accident level from 2016 to 2017. Potential Accident level IV is dominant in both 2016 and 2017

9.4 Potential Accident Level Vs Country:

Count Plot:



Cross table Analysis:

Country	Country_01	Country_02	Country_03
Potential Accident Level			
I	10	6	29
II	51	40	4
III	64	41	1
IV	101	33	7
V	21	9	0
VI	1	0	0

Observation –

Potential Accident Level IV is dominant across all countries, while with VI least number of accidents happened.

7 Natureofemployee Vs RestAll:

10.

10.1 Natureofemployee Vs Critical Risk:

Cross table Analysis:

Natureofemployee Critical Risk	Employee	Third Party	Third Party (Remote)
\nNot applicable	1	0	0
Bees	1	9	0
Blocking and isolation of energies	2	0	1
Burn	1	0	0
Chemical substances	9	2	6
Confined space	1	0	0
Cut	8	5	1
Electrical Shock	0	0	2
Electrical installation	0	1	0
Fall	0	5	4
Fall prevention	2	3	1
Fall prevention (same level)	3	4	0
Individual protection equipment	0	1	0
Liquid Metal	3	0	0
Machine Protection	1	1	0
Manual Tools	7	7	6
Others	99	109	21
Plates	1	0	0
Poll	0	1	0
Power lock	0	0	3
Pressed	12	7	5
Pressurized Systems	4	1	2
Pressurized Systems / Chemical Substances	1	1	1
Projection	7	6	0
Projection of fragments	0	1	0
Projection/Burning	1	0	0
Projection/Choco	0	1	0
Projection/Manual Tools	0	1	0
Suspended Loads	4	1	1
Traffic	1	0	0
Vehicles and Mobile Equipment	3	5	0
Venomous Animals	3	9	1
remains of choco	3	4	0

Observation –

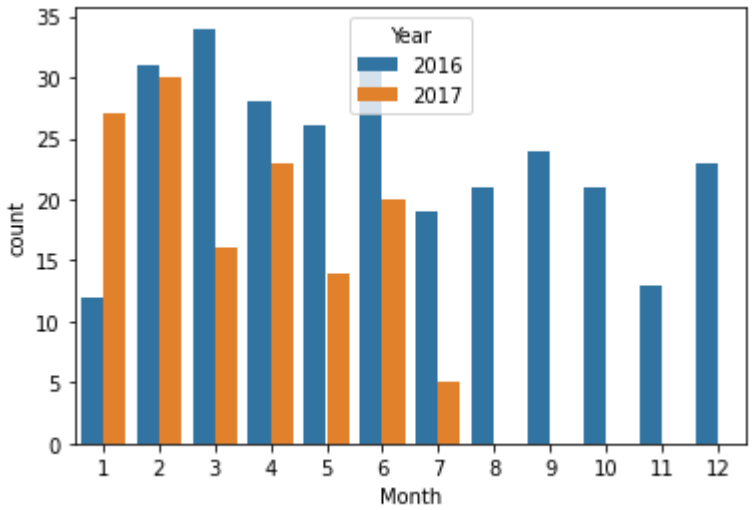
Critical Risk of type "Others" is dominant across all Types of Employees.

8 Year Vs RestAll:

11.

11.1 Year vs month:

Count Plot:



Cross table Analysis:

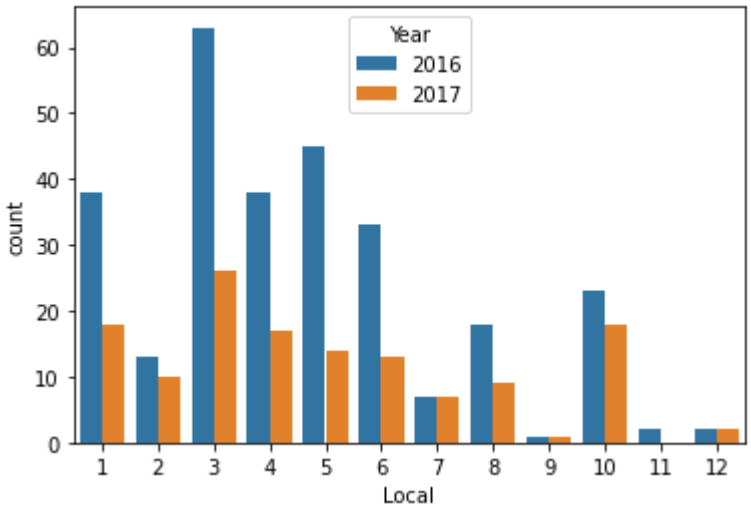
Year	2016	2017
Month		
1	12	27
2	31	30
3	34	16
4	28	23
5	26	14
6	31	20
7	19	5
8	21	0
9	24	0
10	21	0
11	13	0
12	23	0

Observation –

From the above plot, it is evident that the max accidents happened in the year 2016 and march.

11.2 Year vs Local:

Count Plot:



Cross table Analysis:

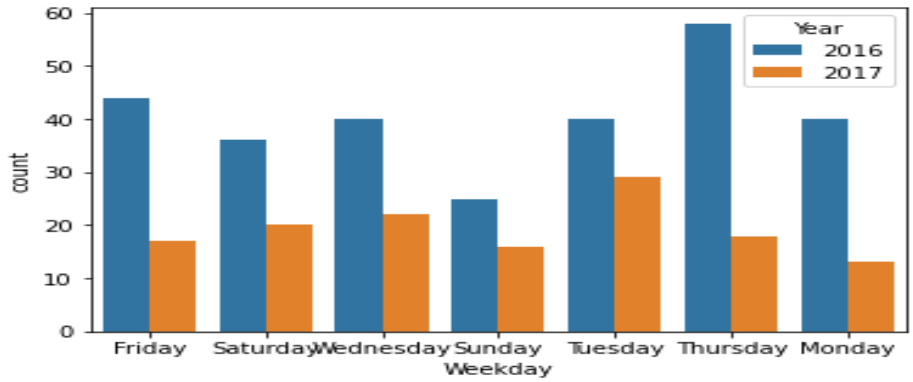
Local	1	2	3	4	5	6	7	8	9	10	11	12
Year												
2016	38	13	63	38	45	33	7	18	1	23	2	2
2017	18	10	26	17	14	13	7	9	1	18	0	2

Observation –

From the above plot, it can be determined that the maximum accidents took place in the local 3 and year 2016.

11.3 Year vs Weekday:

Count Plot:



Cross table Analysis:

Weekday	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday
Year							
2016	44	40	36	25	58	40	40
2017	17	13	20	16	18	29	22

Observation –

From the above plot, it is clearly evident that maximum number of accidents took place on Thursday and year 2016.

11.4 Year vs Critical Risk:

Cross table Analysis:

Year	2016	2017
Critical Risk		
\nNot applicable	1	0
Bees	10	0
Blocking and isolation of energies	3	0
Burn	0	1
Chemical substances	13	4
Confined space	1	0
Cut	6	8
Electrical Shock	0	2
Electrical installation	1	0
Fall	2	7
Fall prevention	1	5
Fall prevention (same level)	6	1
Individual protection equipment	0	1
Liquid Metal	2	1
Machine Protection	0	2
Manual Tools	14	6
Others	189	40

Plates	1	0
Poll	1	0
Power lock	0	3
Pressed	14	10
Pressurized Systems	7	0
Pressurized Systems / Chemical Substances	3	0
Projection	1	12
Projection of fragments	0	1
Projection/Burning	0	1
Projection/Choco	0	1
Projection/Manual Tools	0	1
Suspended Loads	5	1
Traffic	1	0
Vehicles and Mobile Equipment	0	8
Venomous Animals	1	12
remains of choco	0	7

Observation: From the above plot, it is clearly evident that maximum number of accidents took place with "Others" and year 2016.

Groupby Analysis:

1. **Year wise distribution of accidents and potential accident levels:**

data.groupby(['Year','Accident Level','Potential Accident Level'])[['Accident Level']].count()

			Accident Level
Year	Accident Level	Potential Accident Level	
2016	I	I	26
		II	62
		III	64
		IV	53
		V	6
	II	II	7
		III	9
		IV	10
	III	III	2
		IV	20
		V	2
	IV	IV	14
		V	5
	V	V	3
2017	I	I	19
		II	26
		III	25
		IV	25
		V	3
	II	III	5
		IV	6
		V	3
	III	III	1
		IV	6
	IV	IV	7
		V	4
	V	V	4
		VI	1

-Year 2016 with Accident Level I have maximum accidents of 64 with Potential Accident Level III and 62 with Potential Accident Level II

- Year 2017 with Accident Level I have maximum accidents of 26 with Potential Accident Level II and 25 with Potential Accident Level III, IV

2. **Year wise distribution of Industry Sector and accident levels:**

data.groupby(['Year','Industry Sector','Accident Level'])[['Accident Level']].count()

Year	Industry Sector	Accident Level	Accident Level
2016	Metals	I	79
		II	9
		III	4
		IV	5
	Mining	I	112
		II	15
		III	17
		IV	12
		V	3
	Others	I	20
		II	2
		III	3
		IV	2
2017	Metals	I	28
		II	3
		III	3
		IV	2
		V	1
	Mining	I	51
		II	11
		III	3
		IV	9
		V	4
	Others	I	19
		III	1

1. Year 2016 with Industry Sector of Type "Metals" has maximum accidents of 79 with Accident Level I
2. Year 2016 with Industry Sector of Type "Mining" has maximum accidents of 112 with Accident Level I
3. Year 2016 with Industry Sector of Type "Others" has maximum accidents of 20 with Accident Level I
4. Year 2017 with Industry Sector of Type "Metals" has maximum accidents of 28 with Accident Level I
5. Year 2017 with Industry Sector of Type "Mining" has maximum accidents of 51 with Accident Level I
6. Year 2017 with Industry Sector of Type "Others" has maximum accidents of 19 with Accident Level I

3. **Industry Sector wise distribution of Country and accident levels**

data.groupby(['Industry Sector','Country','Accident Level'])[['Accident Level']].count()

Industry Sector	Country	Accident Level	Accident Level
Metals	Country_01	I	36
		II	1
		III	3
		IV	5
		V	1
	Country_02	I	71
		II	11
		III	4
		IV	2
Mining	Country_01	I	140
		II	18
		III	17
		IV	18
		V	7
	Country_02	I	23
		II	6
		III	3
		IV	3
Others	Country_01	I	1
		III	1
	Country_02	I	4
	Country_03	I	34
		II	2
		III	3
		IV	2

1. Metals in Country_01 has maximum accidents with Level 1 with 36 count
2. Metals in Country_02 has maximum accidents with Level 1 with 71 count
3. Mining in Country_01 has maximum accidents with Level 1 with 140 count
4. Mining in Country_02 has maximum accidents with Level 1 with 23 count
5. Others in Country_03 has maximum accidents with Level 1 with 34 count

Word Cloud Analysis:

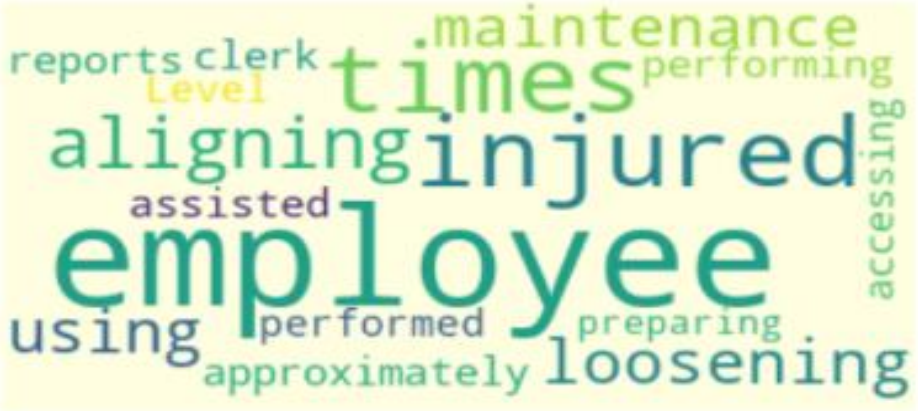
Accident Level:

1. **WordCloud for Accident Level : I**

2. **WordCloud for Accident Level : IV**

3. **WordCloud for Accident Level : III**

4. **WordCloud for Accident Level : II**



5. **WordCloud for Accident Level : V**

Potential Accident Level:

1. **WordCloud for Potential Accident Level : IV**

2. **WordCloud for Potential Accident Level : III**

3. **WordCloud for Potential Accident Level : I**

4. **WordCloud for Potential Accident Level : II**

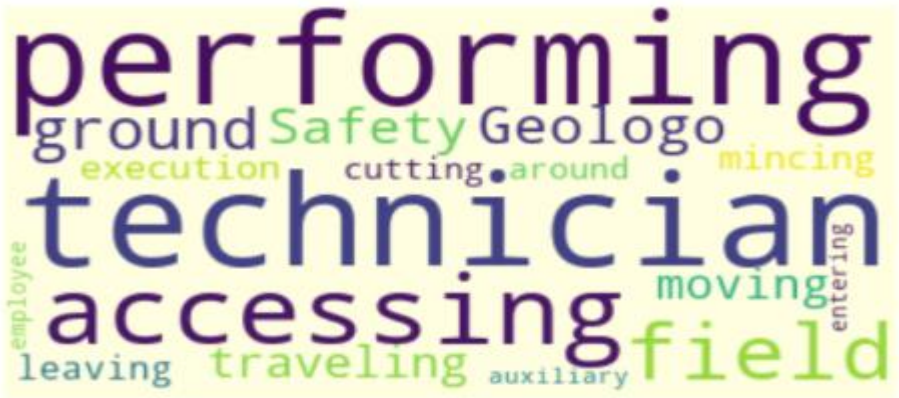
5. **WordCloud for Potential Accident Level : V**

Industry Sector

1. **WordCloud for Industry type : Mining**

2. **WordCloud for Industry type : Metals**

3. **WordCloud for Industry type : Others**



Country:

1. **WordCloud for Country : Country_01**

2. **WordCloud for Country : Country_02**

3. **WordCloud for Country : Country_03**

INFERENCE:

Inference from Univariate Analysis:

- Country_01 has 59.3% (highest) Proportion of Total number of accidents that took place in 2016 and 2017.
- Local_03 has 21.3% (highest) Proportion in Total number of accidents that took place in 2016 and 2017.
- Mining has 56.7% (highest) Proportion in Total number of accidents that took place in 2016 and 2017.
- Accident level I has 73.9% (highest) Proportion in Total number of accidents that took place in 2016 and 2017.
- PotentialAccidentlevel IV has 33.7%(highest) Proportion in Total number of accidents that took place in 2016 and 2017.
- Male Gender has 94.7%(highest) Proportion in Total number of accidents that took place in 2016 and 2017.
- Nature of Employee of type "Third Party" has 44.3% (highest) Proportion in Total number of accidents that took place in 2016 and 2017.
- Critical Risk of type "Others" has 54.8% (highest) Proportion in Total number of accidents that took place in 2016 and 2017.
- Year 2016 has 67.7% (highest) Proportion in Total number of accidents that took place among the whole.
- Feb Month has 14.6% (highest) Proportion in Total number of accidents, followed by April and June with 12.2%, that took place in 2016 and 2017.
- Thursday has 18.2% (highest) Proportion in Total number of accidents that took place in 2016 and 2017, followed by Tuesday and Wednesday. While the Lowest seems to be on Sunday.

Inference from Bivariate Analysis:

- Accident level I is dominant among Male and Female Genders.
- Potential Accident Level of IV is dominant among Male, while II is dominant among Female.
- Country_01 is dominant among Male accidents, while Country_02 is dominant among Female accidents.
- Mining sector is dominant among Male accidents while Metals is dominant among Female accidents.
- Third Party employee type is dominant in accidents among Male and Female Gender
- Accident Level I has maximum number of Accidents happened in the Mining sector.
- Potential accident level 4 has maximum number of Accidents happened in the Mining sector.
- Local 3 has maximum number of Accidents happened in the Mining sector. Local 11 has least number of accidents in industrial sector.
- Mining sector has maximum number of accidents took place in Country_01 while Others has least number of accidents in Country_01.
- Maximum number of accidents took place in accident level 1 and country_01.
- Maximum number of accidents took place in country_01 and potential accident level 3.
- Country 1 is more dominant among all Local Regions while Country 03 is least dominant.
- Country 1 is more dominant among all Types of Employees while Country 03 is least dominant.
- Overall Local 3 is more prone to Multiple potential Accidents, while Local 12 is the least dominant.
- Employee is more dominant across all Local Regions, while Type Third Party (Remote) is least dominant across all Local Regions.
- Accident Level I is more related to Potential Accident levels of all Levels, while Accident Level V is least related.
- Accident Level I is more dominant across all Employee types and all Countries and all
- Regions, where Level V is least across all Employee Types, all Countries and all Regions.
- Potential Accident level IV dominents in ThirdParty, while VI is least dominant in Third

Party (Remote) across all.

- Critical Risk of type "Others" is dominant across all Types of Employees, Regions, Industry Types.
- Maximum accidents took place in the Local Region 3 in both 2016 and 2017 while Local Region 12 is the least among both years.

Summary:

1. Employees in Country_01 are more prone to Accidents. More Precautionary measures and training should be given to avoid similar accidents.
2. Most accidents are of less Severity Comparatively. More Precautionary measures and training will help avoid similar accidents.
3. Study reveals more potential accidents can happen with High Severity of IV. Minimizing the Less Severity accidents can help avoid High Severity Potential Accidents.
4. Mining industry needs special Care and attention by providing Industry level Safety Measures to help Employees avoid accidents.
5. More Study and Analysis is required on what Accident category types that Critical Risk of type "Others" is associated with and necessary Precautionary measures should be implemented.
6. Though there is a decline in Number of Accidents from 2016 and 2017, the less Severity accidents are still dominant. Special attention is required to minimize these from Management.
7. Feb, April, June are Critical Months which are more prone to accidents. Monitoring the Weather, Industrial Environments and Employee Work Mood should be analysed and precautioned with proper arrangements.
8. Thursday is most prone to accidents. Needs to check if proper Senior Technicians are available and monitor the situation with Timely maintenance to avoid Accidents happening due to any failures.

OVERVIEW OF THE FINAL PROCESS

DATA MODELLING:

Both Machine Learning and Deep Learning algorithms have been used for modelling and compared their performances and selected the model with highest score. Objective is to select the best accuracy score model by predicting both "Accident Level" and "Potential Accident Level" labels.

Machine Learning Models:

1) Support Vector Classification (SVC)
2) Random Forest
3) Gradient Boosting
4) XG Boost
5) K-neighbours
6) Naive Bayes
7) Bagging
8) Ada Boost

Deep Learning Model:

Long Term Short Memory (LSTM) architecture with Word2Vec

Text Processing Techniques:

1) Count Vectorizer
2) TF-IDF

STEP-BY-STEP WALK THROUGH THE SOLUTION

Accident Level

Count Vectorization

```
#Count vectorization
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer

X = data1['description_processed']
y = data1['Accident Level']

count_vec = CountVectorizer(analyzer='word', ngram_range=(1, 2))
Xc = count_vec.fit_transform(X).toarray()
Xc_train, Xc_test, yc_train, yc_test = train_test_split(Xc, y, test_size=0.15, random_state=42)
```

```
Xc_train.shape
```

```
(355, 13242)
```

```
Xc_test.shape
```

```
(63, 13242)
```

VISUALIZATIONS

1) Support Vector Classification (SVC)

```
#SVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.svm import LinearSVC

svc = LinearSVC( max_iter=2500)
svc.fit(Xc_train, yc_train)
yc_pred_SVC = svc.predict(Xc_test)
```

```
acc_svc = accuracy_score(yc_test, yc_pred_SVC)
acc_svc_tr = svc.score(Xc_train, yc_train)
print("Train accuracy of the SVC model : {:.2f}".format(acc_svc_tr*100))
print("Test accuracy of the SVC model : {:.2f}".format(acc_svc*100))
```

```
Train accuracy of the SVC model : 99.44
Test accuracy of the SVC model : 79.37
```

```
print('Classification report:',classification_report(yc_test, yc_pred_SVC))
```

```
Classification report:               precision    recall  f1-score   support

           I       0.80      1.00      0.89        48
          II       0.00      0.00      0.00         5
         III       1.00      0.25      0.40         4
          IV       1.00      0.20      0.33         5
           V       0.00      0.00      0.00         1

    accuracy                           0.79        63
   macro avg       0.56      0.29      0.32        63
weighted avg       0.75      0.79      0.73        63
```

2) Random Forest

```
#Random forest
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(random_state=42)
rfc.fit(Xc_train, yc_train)
yc_pred_rfc = rfc.predict(Xc_test)

acc_rfc = accuracy_score(yc_test, yc_pred_rfc)
acc_rfc_tr = rfc.score(Xc_train,yc_train)

print("Train Accuracy of the Random Forest model : {:.2f}".format(acc_rfc_tr*100))
print("Test Accuracy of the Random Forest model : {:.2f}".format(acc_rfc*100))
print('Classification report:',classification_report(yc_test, yc_pred_rfc))
print('Confusion matrix:', confusion_matrix(yc_test,yc_pred_rfc))
```

```
Train Accuracy of the Random Forest model : 99.44
Test Accuracy of the Random Forest model : 76.19
Classification report:               precision    recall  f1-score   support

           I       0.76      1.00      0.86        48
          II       0.00      0.00      0.00         5
         III       0.00      0.00      0.00         4
          IV       0.00      0.00      0.00         5
           V       0.00      0.00      0.00         1

    accuracy                           0.76        63
   macro avg       0.15      0.20      0.17        63
weighted avg       0.58      0.76      0.66        63

Confusion matrix: [[48  0  0  0  0]
 [ 5  0  0  0  0]
 [ 4  0  0  0  0]
 [ 5  0  0  0  0]
 [ 1  0  0  0  0]]
```

3) Gradient Boosting

```
#Gradient boosting
from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier(n_estimators=100)
gbc.fit(Xc_train, yc_train)
yc_pred_gb = gbc.predict(Xc_test)
acc_gbc = accuracy_score(yc_test, yc_pred_gb)
acc_gbc_tr = gbc.score(Xc_train,yc_train)

print(" Test accuracy of the Gradient boosting model : {:.2f}".format(acc_gbc*100))
print("Train accuracy of the Gradient boosting model : {:.2f}".format(acc_gbc_tr*100))
print('Classification report:',classification_report(yc_test,yc_pred_gb ))
```

```
 Test accuracy of the Gradient boosting model : 71.43
Train accuracy of the Gradient boosting model : 99.44
Classification report:               precision    recall  f1-score   support

           I       0.79      0.94      0.86        48
          II       0.00      0.00      0.00         5
         III       0.00      0.00      0.00         4
          IV       0.00      0.00      0.00         5
           V       0.00      0.00      0.00         1

    accuracy                           0.71        63
   macro avg       0.16      0.19      0.17        63
weighted avg       0.60      0.71      0.65        63
```

```
print('Confusion matrix:', confusion_matrix(yc_test,yc_pred_gb))
```

```
Confusion matrix: [[45  1  0  1  1]
 [ 5  0  0  0  0]
 [ 3  0  0  1  0]
 [ 3  0  2  0  0]
 [ 1  0  0  0  0]]
```

```
print('Confusion matrix:', confusion_matrix(yc_test,yc_pred_gb))
```

```
Confusion matrix: [[45  1  0  1  1]
 [ 5  0  0  0  0]
 [ 3  0  0  1  0]
 [ 3  0  2  0  0]
 [ 1  0  0  0  0]]
```

4) XG Boost

```
#XGBOOST
from xgboost import XGBClassifier

xgbc = XGBClassifier()
xgbc.fit(Xc_train, yc_train)
yc_pred_xg = xgbc.predict(Xc_test)
acc_xgbc = accuracy_score(yc_test, yc_pred_xg)
acc_xgbc_tr = xgbc.score(Xc_train,yc_train)

print(" Test accuracy of the XGBoost model : {:.2f}".format(acc_xgbc*100))
print("Train accuracy of the XGBoost model : {:.2f}".format(acc_xgbc_tr*100))
print('Classification report:',classification_report(yc_test,yc_pred_xg))
print('Confusion matrix:', confusion_matrix(yc_test,yc_pred_xg))
```

```
 Test accuracy of the XGBoost model : 73.02
Train accuracy of the XGBoost model : 93.80
Classification report:               precision    recall  f1-score   support

           I       0.75      0.96      0.84        48
          II       0.00      0.00      0.00         5
         III       0.00      0.00      0.00         4
          IV       0.00      0.00      0.00         5
           V       0.00      0.00      0.00         1

    accuracy                           0.73        63
   macro avg       0.15      0.19      0.17        63
weighted avg       0.57      0.73      0.64        63

Confusion matrix: [[46  0  1  0  1]
 [ 5  0  0  0  0]
 [ 4  0  0  0  0]
 [ 5  0  0  0  0]
 [ 1  0  0  0  0]]
```

5) K Neighbors

```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(Xc_train, yc_train)
```

```
KNeighborsClassifier(n_neighbors=3)
```

```
yc_pred_k = neigh.predict(Xc_test)
acc_neigh = accuracy_score(yc_test, yc_pred_k)
acc_neigh_tr = neigh.score(Xc_train,yc_train)
```

```
print(" Test accuracy of the KNeighbours model : {:.2f}".format(acc_neigh*100))
print("Train accuracy of the Kneighbours model : {:.2f}".format(acc_neigh_tr*100))
print('Classification report:',classification_report(yc_test,yc_pred_k))
print('Confusion matrix:', confusion_matrix(yc_test,yc_pred_k))
```

```
 Test accuracy of the KNeighbours model : 76.19
Train accuracy of the Kneighbours model : 74.08
Classification report:               precision    recall  f1-score   support

           I       0.77      1.00      0.87        48
          II       0.00      0.00      0.00         5
         III       0.00      0.00      0.00         4
          IV       0.00      0.00      0.00         5
           V       0.00      0.00      0.00         1

    accuracy                           0.76        63
   macro avg       0.15      0.20      0.17        63
weighted avg       0.59      0.76      0.66        63

Confusion matrix: [[48  0  0  0  0]
 [ 5  0  0  0  0]
 [ 4  0  0  0  0]
 [ 4  1  0  0  0]
 [ 1  0  0  0  0]]
```

6) Naive Bayes

```
#naive bayes
from sklearn.naive_bayes import GaussianNB
g_model1 = GaussianNB()
g_model1.fit(Xc_train, yc_train)
g_pred1 = g_model1.predict(Xc_test)

#print('Accuracy on Training data:',g_model1.score(Xc_train, yc_train) ) #accuracy of training data
#print('Accuracy on Test data:',g_model1.score(Xc_test, yc_test) ) #accuracy data of testing data
```

```
print('Accuracy on Training data:',g_model1.score(Xc_train, yc_train) ) #accuracy of training data
print('Accuracy on Test data:',g_model1.score(Xc_test, yc_test) ) #accuracy data of testing data
```

```
Accuracy on Training data: 0.9943661971830986
Accuracy on Test data: 0.746031746031746
```

```
print('Classification report:',classification_report(yc_test,g_pred1))
print('Confusion matrix:', confusion_matrix(yc_test,g_pred1))
```

```
Classification report:               precision    recall  f1-score   support

           I       0.76      0.98      0.85        48
          II       0.00      0.00      0.00         5
         III       0.00      0.00      0.00         4
          IV       0.00      0.00      0.00         5
           V       0.00      0.00      0.00         1

    accuracy                           0.75        63
   macro avg       0.15      0.20      0.17        63
weighted avg       0.58      0.75      0.65        63

Confusion matrix: [[47  0  0  1  0]
 [ 5  0  0  0  0]
 [ 4  0  0  0  0]
 [ 5  0  0  0  0]
 [ 1  0  0  0  0]]
```

7) Bagging Classifier

```
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import GridSearchCV, StratifiedShuffleSplit
n_estimators = [10,30,50,70,80,150,160, 170,175,180,185];
cv = StratifiedShuffleSplit(n_splits=10, test_size=.30, random_state=15)

parameters = {'n_estimators':n_estimators,

        }
grid = GridSearchCV(BaggingClassifier(base_estimator= None, ## If None, then the base estimator is a decision tree.
                                      bootstrap_features=False),
                                param_grid=parameters,
                                cv=cv,
                                n_jobs = -1)
grid.fit(Xc_train, yc_train)
```

```
GridSearchCV(cv=StratifiedShuffleSplit(n_splits=10, random_state=15, test_size=0.3,
            train_size=None),
             estimator=BaggingClassifier(), n_jobs=-1,
             param_grid={'n_estimators': [10, 30, 50, 70, 80, 150, 160, 170,
                                          175, 180, 185]})
```

```
yc_pred_b = grid.predict(Xc_test)
acc_b = accuracy_score(yc_test, yc_pred_b)
acc_b_tr = grid.score(Xc_train,yc_train)

print(" Test accuracy of the Bagging model : {:.2f}".format(acc_b*100))
print("Train accuracy of the Bagging model : {:.2f}".format(acc_b_tr*100))
print('Classification report:',classification_report(yc_test,yc_pred_b))
print('Confusion matrix:', confusion_matrix(yc_test,yc_pred_b))
```

```
 Test accuracy of the Bagging model : 76.19
Train accuracy of the Bagging model : 99.15
Classification report:               precision    recall  f1-score   support

           I       0.77      1.00      0.87        48
          II       0.00      0.00      0.00         5
         III       0.00      0.00      0.00         4
          IV       0.00      0.00      0.00         5
           V       0.00      0.00      0.00         1

    accuracy                           0.76        63
   macro avg       0.15      0.20      0.17        63
weighted avg       0.59      0.76      0.66        63

Confusion matrix: [[48  0  0  0  0]
 [ 5  0  0  0  0]
 [ 4  0  0  0  0]
 [ 4  0  1  0  0]
 [ 1  0  0  0  0]]
```

8) Ada Boosting Classifier

```
from sklearn.ensemble import AdaBoostClassifier
clf_ada = AdaBoostClassifier(n_estimators=100, random_state=0)
clf_ada.fit(Xc_train, yc_train)
```

```
AdaBoostClassifier(n_estimators=100, random_state=0)
```

```
yc_pred_ada = clf_ada.predict(Xc_test)
acc_ada = accuracy_score(yc_test, yc_pred_ada)
acc_ada_tr = clf_ada.score(Xc_train,yc_train)

print(" Test accuracy of the ADA model : {:.2f}".format(acc_ada*100))
print("Train accuracy of the ADA model : {:.2f}".format(acc_ada_tr*100))
print('Classification report:',classification_report(yc_test,yc_pred_ada))
print('Confusion matrix:', confusion_matrix(yc_test,yc_pred_ada))
```

```
 Test accuracy of the ADA model : 73.02
Train accuracy of the ADA model : 75.77
Classification report:               precision    recall  f1-score   support

           I       0.75      0.96      0.84        48
          II       0.00      0.00      0.00         5
         III       0.00      0.00      0.00         4
          IV       0.00      0.00      0.00         5
           V       0.00      0.00      0.00         1

    accuracy                           0.73        63
   macro avg       0.15      0.19      0.17        63
weighted avg       0.57      0.73      0.64        63

Confusion matrix: [[46  1  0  1  0]
 [ 5  0  0  0  0]
 [ 4  0  0  0  0]
 [ 5  0  0  0  0]
 [ 1  0  0  0  0]]
```

Potential Accident Level

```
#Target variable- potential accident level
tfidf_vecp = TfidfVectorizer(ngram_range=(1,2))
Xpa = tfidf_vecp.fit_transform(X).toarray()

Xpa_train, Xpa_test, ypa_train, ypa_test = train_test_split(Xpa, yp, test_size=0.15, random_state=42)
```

1) Support Vector Machine (SVC)

```
#SVC Model Training and Evaluation
svc_pa = LinearSVC(max_iter=5500)
svc_pa.fit(Xpa_train, ypa_train)
ypa_pred_pa = svc_pa.predict(Xpa_test)

# Evaluation
accpa_svc = accuracy_score(ypa_test,ypa_pred_pa  )
accpa_svc_pa_tr = svc_pa.score(Xpa_train,ypa_train)

print("Test accuracy of the SVC model : {:.2f}".format(accpa_svc*100))
print("Train accuracy of the SVC model : {:.2f}".format(accpa_svc_pa_tr*100))
```

```
Test accuracy of the SVC model : 42.86
Train accuracy of the SVC model : 99.72
```

```
print('Classification report:',classification_report(ypa_test,ypa_pred_pa))
print('Confusion matrix:', confusion_matrix(ypa_test,ypa_pred_pa))
```

```
Classification report:               precision    recall  f1-score   support

           I       0.50      0.14      0.22         7
          II       0.56      0.45      0.50        11
         III       0.43      0.16      0.23        19
          IV       0.40      0.78      0.53        23
           V       0.00      0.00      0.00         3

    accuracy                           0.43        63
   macro avg       0.38      0.31      0.30        63
weighted avg       0.43      0.43      0.37        63

Confusion matrix: [[ 1  1  2  3  0]
 [ 0  5  1  5  0]
 [ 0  0  3 16  0]
 [ 1  3  1 18  0]
 [ 0  0  0  3  0]]
```

2) Random Forest

```
#Random forest
rf_pa = RandomForestClassifier(random_state=42)
rf_pa.fit(Xpa_train, ypa_train)
ypa_pred_rf = rf_pa.predict(Xpa_test)

acc_pa_rf = accuracy_score(ypa_test, ypa_pred_rf)
acc_pa_rf_tr = rf_pa.score(Xpa_train, ypa_train)

print("Test accuracy of the Random Forest model : {:.2f}".format(acc_pa_rf*100))
print("Train accuracy of the Random Forest model : {:.2f}".format(acc_pa_rf_tr*100))
print('Classification report:',classification_report(ypa_test,ypa_pred_rf))
print('Confusion matrix:', confusion_matrix(ypa_test,ypa_pred_rf))
```

```
Test accuracy of the Random Forest model : 41.27
Train accuracy of the Random Forest model : 99.72
Classification report:               precision    recall  f1-score   support

           I       1.00      0.14      0.25         7
          II       0.75      0.27      0.40        11
         III       0.36      0.21      0.27        19
          IV       0.38      0.78      0.51        23
           V       0.00      0.00      0.00         3

    accuracy                           0.41        63
   macro avg       0.50      0.28      0.29        63
weighted avg       0.49      0.41      0.37        63

Confusion matrix: [[ 1  0  1  5  0]
 [ 0  3  2  6  0]
 [ 0  0  4 15  0]
 [ 0  1  4 18  0]
 [ 0  0  0  3  0]]
```

3) Gradient Boosting

```
#Gradient boosting
gb_pa = GradientBoostingClassifier(n_estimators=100)
gb_pa.fit(Xpa_train, ypa_train)
ypa_pred_gb = gb_pa.predict(Xpa_test)
acc_pa_gb = accuracy_score(ypa_test,ypa_pred_gb )
acc_pa_gb_tr = gb_pa.score(Xpa_train, ypa_train)

print("Test accuracy of the Gradient boosting model : {:.2f}".format(acc_pa_gb*100))
print("Train accuracy of the Gradient boosting model : {:.2f}".format(acc_pa_gb_tr*100))
print('Classification report:',classification_report(ypa_test,ypa_pred_gb))
print('Confusion matrix:', confusion_matrix(ypa_test,ypa_pred_gb))
```

```
Test accuracy of the Gradient boosting model : 41.27
Train accuracy of the Gradient boosting model : 99.72
Classification report:               precision    recall  f1-score   support

           I       1.00      0.14      0.25         7
          II       0.00      0.00      0.00        11
         III       0.36      0.42      0.39        19
          IV       0.47      0.74      0.58        23
           V       0.00      0.00      0.00         3

    accuracy                           0.41        63
   macro avg       0.37      0.26      0.24        63
weighted avg       0.39      0.41      0.36        63

Confusion matrix: [[ 1  1  3  2  0]
 [ 0  0  5  5  1]
 [ 0  2  8  9  0]
 [ 0  0  6 17  0]
 [ 0  0  0  3  0]]
```

4) XG Boost

```
#XGBoost Model for Training and Evaluation
xgb_pa = XGBClassifier()
xgb_pa.fit(Xpa_train, ypa_train)
ypa_pred_pa = xgb_pa.predict(Xpa_test)
acc_pa_xgb = accuracy_score(ypa_test,ypa_pred_pa )
acc_pa_xgb_tr = xgb_pa.score(Xpa_train,ypa_train)

print(" Test accuracy of the XGBoost model : {:.2f}".format(acc_pa_xgb*100))
print("Train accuracy of the XGBoost model : {:.2f}".format(acc_pa_xgb_tr*100))
print('Classification report:',classification_report(ypa_test,ypa_pred_pa))
print('Confusion matrix:', confusion_matrix(ypa_test,ypa_pred_pa))
```

```
 Test accuracy of the XGBoost model : 42.86
Train accuracy of the XGBoost model : 97.18
Classification report:               precision    recall  f1-score   support

           I       1.00      0.14      0.25         7
          II       0.30      0.27      0.29        11
         III       0.39      0.37      0.38        19
          IV       0.48      0.70      0.57        23
           V       0.00      0.00      0.00         3

    accuracy                           0.43        63
   macro avg       0.43      0.30      0.30        63
weighted avg       0.46      0.43      0.40        63

Confusion matrix: [[ 1  1  3  2  0]
 [ 0  3  4  4  0]
 [ 0  3  7  8  1]
 [ 0  3  4 16  0]
 [ 0  0  0  3  0]]
```

5) K Neighbors

```
#Kneighbours
from sklearn.neighbors import KNeighborsClassifier
neigh_4 = KNeighborsClassifier(n_neighbors=3)
neigh_4.fit(Xpa_train, ypa_train)
```

```
KNeighborsClassifier(n_neighbors=3)
```

```
ypa_pred_k1 = neigh_4.predict(Xpa_test)
acc_pa_neigh1 = accuracy_score(ypa_test, ypa_pred_k1)
acc_pa_neigh_tr1 = neigh_4.score(Xpa_train,ypa_train)
```

```
print(" Test accuracy of the KNeighbours model : {:.2f}".format(acc_pa_neigh1*100))
print("Train accuracy of the Kneighbours model : {:.2f}".format(acc_pa_neigh_tr1*100))
print('Classification report:',classification_report(ypa_test,ypa_pred_k1))
print('Confusion matrix:', confusion_matrix(ypa_test,ypa_pred_k1))
```

```
 Test accuracy of the KNeighbours model : 39.68
Train accuracy of the Kneighbours model : 63.94
Classification report:               precision    recall  f1-score   support

           I       0.11      0.14      0.12         7
          II       0.39      0.64      0.48        11
         III       0.40      0.21      0.28        19
          IV       0.50      0.57      0.53        23
           V       0.00      0.00      0.00         3

    accuracy                           0.40        63
   macro avg       0.28      0.31      0.28        63
weighted avg       0.38      0.40      0.38        63

Confusion matrix: [[ 1  4  1  1  0]
 [ 1  7  1  2  0]
 [ 4  3  4  8  0]
 [ 3  3  4 13  0]
 [ 0  1  0  2  0]]
```

6) Naïve Bayes

```
#naive bayes
from sklearn.naive_bayes import GaussianNB
g_model_pa = GaussianNB()
g_model_pa.fit(Xpa_train, ypa_train)
g_pred_pa = g_model_pa.predict(Xpa_test)
```

```
print('Accuracy on Training data:',g_model_pa.score(Xpa_train, ypa_train) ) #accuracy of training data
print('Accuracy on Test data:',g_model_pa.score(Xpa_test, ypa_test) ) #accuracy data of testing data
```

```
Accuracy on Training data: 0.9971830985915493
Accuracy on Test data: 0.4444444444444444
```

```
print('Classification report:',classification_report(ypa_test,g_pred_pa))
print('Confusion matrix:', confusion_matrix(ypa_test,g_pred_pa))
```

```
Classification report:               precision    recall  f1-score   support

           I       0.25      0.14      0.18         7
          II       0.50      0.36      0.42        11
         III       0.42      0.26      0.32        19
          IV       0.47      0.78      0.59        23
           V       0.00      0.00      0.00         3

    accuracy                           0.44        63
   macro avg       0.33      0.31      0.30        63
weighted avg       0.41      0.44      0.41        63

Confusion matrix: [[ 1  2  2  1  1]
 [ 1  4  1  5  0]
 [ 0  1  5 13  0]
 [ 2  0  3 18  0]
 [ 0  1  1  1  0]]
```

7) Bagging

```
#Bagging
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import GridSearchCV, ShuffleSplit
n_estimators_pa = [10,30,50,70,80,150,160, 170,175,180,185];
cvpa = ShuffleSplit(n_splits=10, test_size=.30, random_state=15)

parameters_pa = {'n_estimators':n_estimators_pa,

        }
grid_pa = GridSearchCV(BaggingClassifier(base_estimator= None, ## If None, then the base estimator is a decision tree.
                                         bootstrap_features=False),
                               param_grid=parameters_pa,
                               cv=cvpa,
                               n_jobs = -1)
```

```
grid_pa.fit(Xpa_train, ypa_train)
```

```
GridSearchCV(cv=ShuffleSplit(n_splits=10, random_state=15, test_size=0.3, train_size=None),
             estimator=BaggingClassifier(), n_jobs=-1,
             param_grid={'n_estimators': [10, 30, 50, 70, 80, 150, 160, 170,
                                          175, 180, 185]})
```

```
ypa_pred_b1 = grid_pa.predict(Xpa_test)
acc_pa_b1 = accuracy_score(ypa_test, ypa_pred_b1)
acc_pa_b1_tr = grid_pa.score(Xpa_train,ypa_train)

print(" Test accuracy of the Bagging model : {:.2f}".format(acc_pa_b1*100))
print("Train accuracy of the Bagging model : {:.2f}".format(acc_pa_b1_tr*100))
print('Classification report:',classification_report(ypa_test,ypa_pred_b1))
print('Confusion matrix:', confusion_matrix(ypa_test,ypa_pred_b1))
```

```
 Test accuracy of the Bagging model : 47.62
Train accuracy of the Bagging model : 99.72
Classification report:               precision    recall  f1-score   support

           I       1.00      0.14      0.25         7
          II       0.57      0.36      0.44        11
         III       0.43      0.32      0.36        19
          IV       0.46      0.83      0.59        23
           V       0.00      0.00      0.00         3

    accuracy                           0.48        63
   macro avg       0.49      0.33      0.33        63
weighted avg       0.51      0.48      0.43        63

Confusion matrix: [[ 1  0  2  4  0]
 [ 0  4  3  4  0]
 [ 0  2  6 11  0]
 [ 0  1  3 19  0]
 [ 0  0  0  3  0]]
```

8) Ada Boosting

```
#ADA boosting
from sklearn.ensemble import AdaBoostClassifier
clf_pa_ada1 = AdaBoostClassifier(n_estimators=100, random_state=0)
clf_pa_ada1.fit(Xpa_train, ypa_train)
```

```
AdaBoostClassifier(n_estimators=100, random_state=0)
```

```
ypa_pred_ada1 = clf_pa_ada1.predict(Xpa_test)
acc_pa_ada1 = accuracy_score(ypa_test, ypa_pred_ada1)
acc_pa_ada1_tr = clf_pa_ada1.score(Xpa_train,ypa_train)

print(" Test accuracy of the ADA model : {:.2f}".format(acc_pa_ada1*100))
print("Train accuracy of the ADA model : {:.2f}".format(acc_pa_ada1_tr*100))
print('Classification report:',classification_report(ypa_test,ypa_pred_ada1))
print('Confusion matrix:', confusion_matrix(ypa_test,ypa_pred_ada1))
```

```
 Test accuracy of the ADA model : 28.57
Train accuracy of the ADA model : 43.38
Classification report:               precision    recall  f1-score   support

           I       1.00      0.14      0.25         7
          II       0.25      0.09      0.13        11
         III       0.00      0.00      0.00        19
          IV       0.32      0.70      0.44        23
           V       0.00      0.00      0.00         3

    accuracy                           0.29        63
   macro avg       0.31      0.19      0.16        63
weighted avg       0.27      0.29      0.21        63

Confusion matrix: [[ 1  0  1  5  0]
 [ 0  1  0  9  1]
 [ 0  1  0 17  1]
 [ 0  2  3 16  2]
 [ 0  0  0  3  0]]
```

Comparing all Machine Learning Models

1) **Accident Level**

- **Count Vectorizer**

Score Model	SVC	Random Forest	Gradient Boosting	XG Boost	K-Neighbors	Naïve Bayes	Bagging	Ada Boost
Training Score	99.44	99.44	99.44	93.8	74.08	99	99.15	75.77
Test Score	79.37	76.19	71.43	73.02	76.19	74	76.19	73.02

- **TF-IDF**

Score Model	SVC	Random Forest	Gradient Boosting	XG Boost	K-Neighbors	Naïve Bayes	Bagging	Ada Boost
Training Score	99.15	99.44	99.44	96.62	77.18	99.43	99.15	74.08
Test Score	76.19	76.19	68.25	76.19	74.6	74.6	74.6	71.43

2) **Potential Accident Level**

- **Count Vectorizer**

Score Model	SVC	Random Forest	Gradient Boosting	XG Boost	K-Neighbors	Naïve Bayes	Bagging	Ada Boost
Training Score	99.72	99.72	99.72	90.99	54.65	99.71	99.72	43.38
Test Score	44.44	38.1	38.1	31.75	25.4	42.85	36.51	36.51

- **TF-IDF**

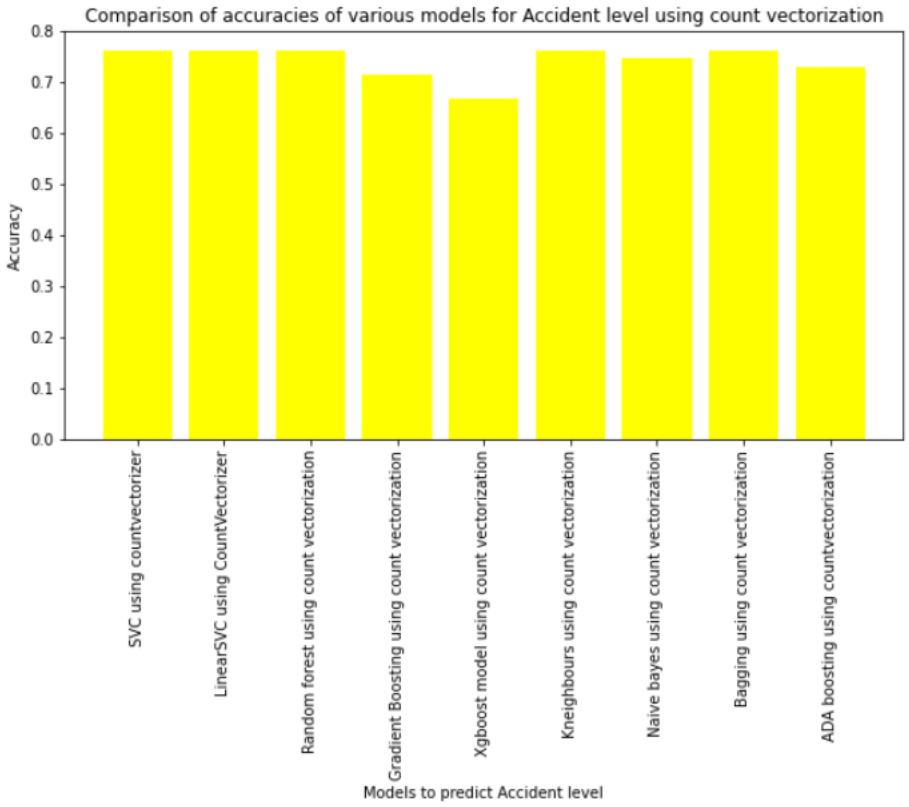
Score Model	SVC	Random Forest	Gradient Boosting	XG Boost	K-Neighbors	Naïve Bayes	Bagging	Ada Boost
Training Score	99.72	99.72	99.72	97.18	63.94	99.71	99.72	43.38
Test Score	42.86	41.27	41.27	42.86	39.68	44.44	47.62	28.57

Comparing Machine Learning Models with target variable as Accident Level and Count vectorization

```
#Comparing all machine learning models
#Machine learning models with target variable as accident level and count vectorization
models_list_accidentlevel = ["SVC using CountVectorizer","Random forest using count vectorization","Gradient Boosting using count vectorization","Xgbo

accuracy_accidentlevel = [acc_svc,acc_rfc,acc_gbc,acc_xgbc,acc_neigh,g_model1.score(Xc_test, yc_test),acc_b,acc_ada]
print(accuracy_accidentlevel)
```

```
[0.7619047619047619, 0.7619047619047619, 0.7142857142857143, 0.7301587301587301, 0.7619047619047619, 0.746031746031746, 0.7619047619047619, 0.73015873
01587301]
```

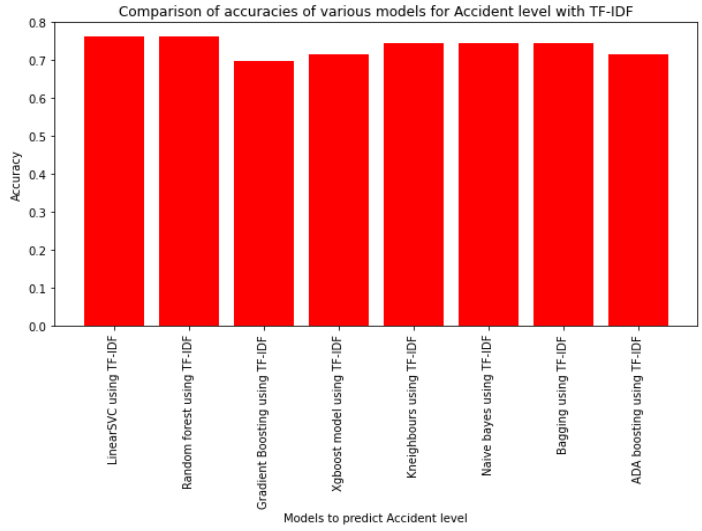
```
# creating the bar plot
fig = plt.figure(figsize = (10, 5))
plt.bar(models_list_accidentlevel, accuracy_accidentlevel, color ='yellow',width=0.8,)

plt.xlabel("Models to predict Accident level")
plt.xticks(rotation = 'vertical')
plt.ylabel("Accuracy")
plt.title("Comparison of accuracies of various models for Accident level using count vectorization")
plt.show()
```



```
models_list_accidentlevel_tf = ["LinearSVC using TF-IDF","Random forest using TF-IDF","Gradient Boosting using TF-IDF","Xgboost model using TF-IDF"

accuracy_accidentlevel_tf = [acct_svctf,acct_rfc_tf,acct_gb_tf,acct_xgb_tf,acc_neigh2,g_model3.score(Xt_test, yt_test),acc_b3,acc_ada_t1
]
print(accuracy_accidentlevel_tf)
```

```
[0.7619047619047619, 0.7619047619047619, 0.6984126984126984, 0.7142857142857143, 0.746031746031746, 0.746031746031746, 0.746031746031746, 0.7142857
142857143]
```

```
# creating the bar plot
fig = plt.figure(figsize = (10, 5))
plt.bar(models_list_accidentlevel_tf, accuracy_accidentlevel_tf, color ='red',width=0.8,)

plt.xlabel("Models to predict Accident level")
plt.xticks(rotation = 'vertical')
plt.ylabel("Accuracy")
plt.title("Comparison of accuracies of various models for Accident level with TF-IDF")
plt.show()
```

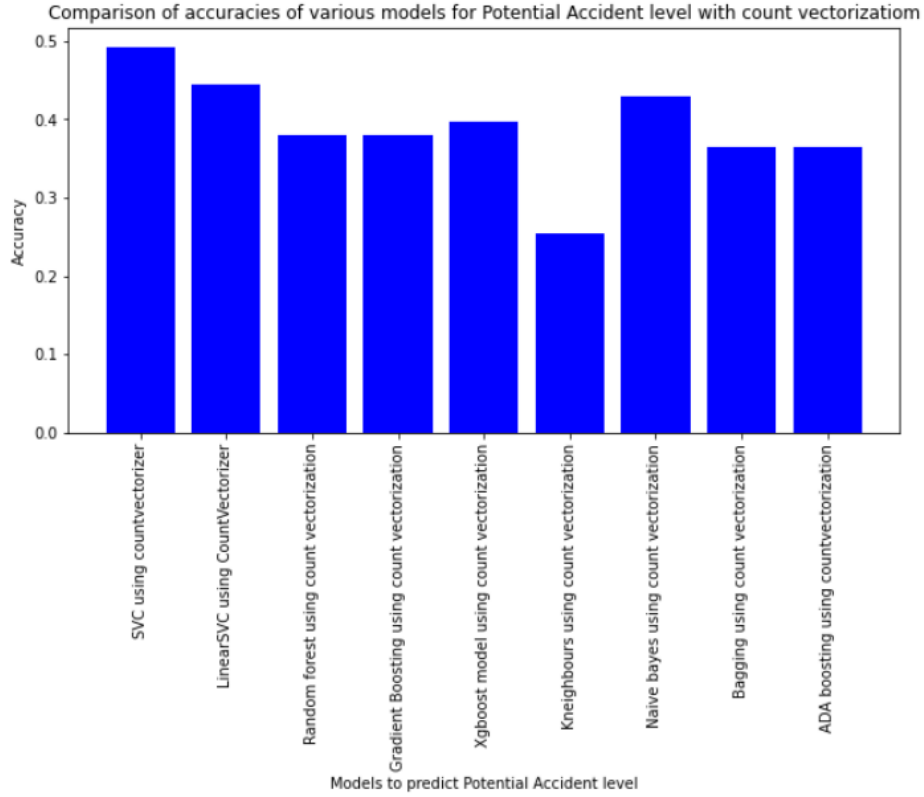


Comparing Machine Learning Models with target variable as Potential Accident Level

```
#Comparing machine learning models with target variable as potential accidentlevel.
#comparing count vectorization models
models_list_potentialaccidentlevel = ["SVC using CountVectorizer","Random forest using count vectorization","Gradient Boosting using count vectorizati

accuracy_paccidentlevel = [accp_svc,accp_rfc,accp_gbc,accp_xgbc,acc_neigh1,g_model2.score(Xp_test, yp_test),acc_b1,acc_ada1]
print(accuracy_paccidentlevel)
```

```
[0.4444444444444444, 0.38095238095238093, 0.38095238095238093, 0.31746031746031744, 0.25396825396825395, 0.42857142857142855, 0.36507936507936506, 0.36
507936507936506]
```

```
# creating the bar plot
fig = plt.figure(figsize = (10, 5))
plt.bar(models_list_potentialaccidentlevel, accuracy_paccidentlevel, color ='blue',width=0.8,)

plt.xlabel("Models to predict Potential Accident level")
plt.xticks(rotation = 'vertical')
plt.ylabel("Accuracy")
plt.title("Comparison of accuracies of various models for Potential Accident level with count vectorizatiom")
plt.show()
```

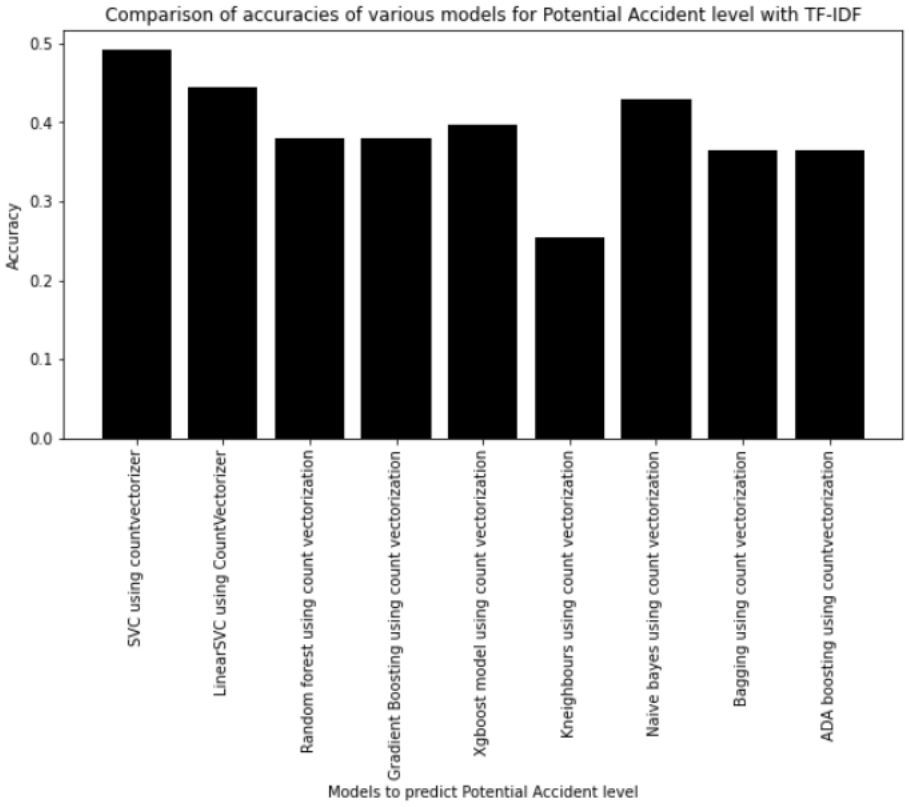


Comparing Machine Learning Models with TF-IDF for the target variable Potential Accident Level

```
#comparing machine learning models with TF-IDF for the target variable potential accident level.
models_list_paccidentlevel_tf = ["SVC using TF-IDF","Random forest using TF-IDF","Gradient Boosting using TF-IDF","Xgboost model using TF-IDF","Kneigh

accuracy_paccidentlevel_tf = [accpa_svc,acc_pa_rf,acc_pa_gb,acc_pa_xgb,acc_pa_neigh1,g_model_pa.score(Xpa_test, ypa_test),acc_pa_b1,acc_pa_ada1]
print(accuracy_paccidentlevel_tf)
```

```
[0.42857142857142855, 0.4126984126984127, 0.4126984126984127, 0.42857142857142855, 0.3968253968253968, 0.4444444444444444, 0.47619047619047616, 0.28571
42857142857]
```

```
# creating the bar plot
fig = plt.figure(figsize = (10, 5))
plt.bar(models_list_potentialaccidentlevel, accuracy_paccidentlevel, color ='black',width=0.8,)

plt.xlabel("Models to predict Potential Accident level")
plt.xticks(rotation = 'vertical')
plt.ylabel("Accuracy")
plt.title("Comparison of accuracies of various models for Potential Accident level with TF-IDF")
plt.show()
```



LSTM WITH WORD2VEC

ACCIDENT LEVEL	POTENTIAL ACCIDENT LEVEL
76.19	38.09

EVALUTAION OF MODEL

```
#Evaluation of the model
results_q = model_w.evaluate(X_testq, y_testq)
```

```
2/2 [==============================] - 0s 4ms/step - loss: 2.5079 - accuracy: 0.2698
```

```
ann_loss1, ann_accuracy1 = model_w.evaluate(X_testq, y_testq, verbose=1)
print('Test loss:', ann_loss1)
print('Test accuracy:', ann_accuracy1)
```

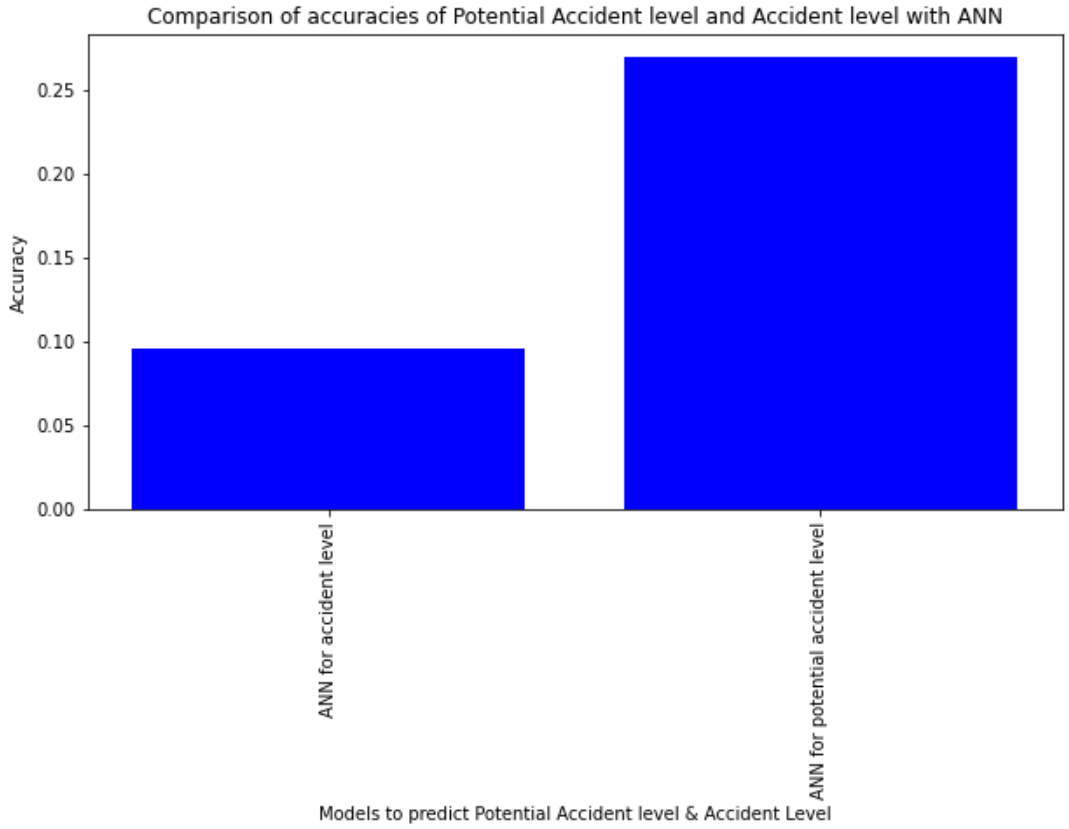
```
2/2 [==============================] - 0s 4ms/step - loss: 2.5079 - accuracy: 0.2698
Test loss: 2.507936477661133
Test accuracy: 0.2698412835597992
```

Comparing accuracies for ANN with target variable accident level and potential accident level

```
models_tlist_accidentlevel_tf = ["ANN for accident level","ANN for potential accident level"]

accuracyt_accidentlevel_tf = [ann_accuracy,ann_accuracy1]
print(accuracyt_accidentlevel_tf)
```

```
[0.095238097012043, 0.2698412835597992]
```

```
# creating the bar plot
fig = plt.figure(figsize = (10, 5))
plt.bar(models_tlist_accidentlevel_tf, accuracyt_accidentlevel_tf, color ='blue',width=0.8,)

plt.xlabel("Models to predict Potential Accident level & Accident Level")
plt.xticks(rotation = 'vertical')
plt.ylabel("Accuracy")
plt.title("Comparison of accuracies of Potential Accident level and Accident level with ANN")
plt.show()
```

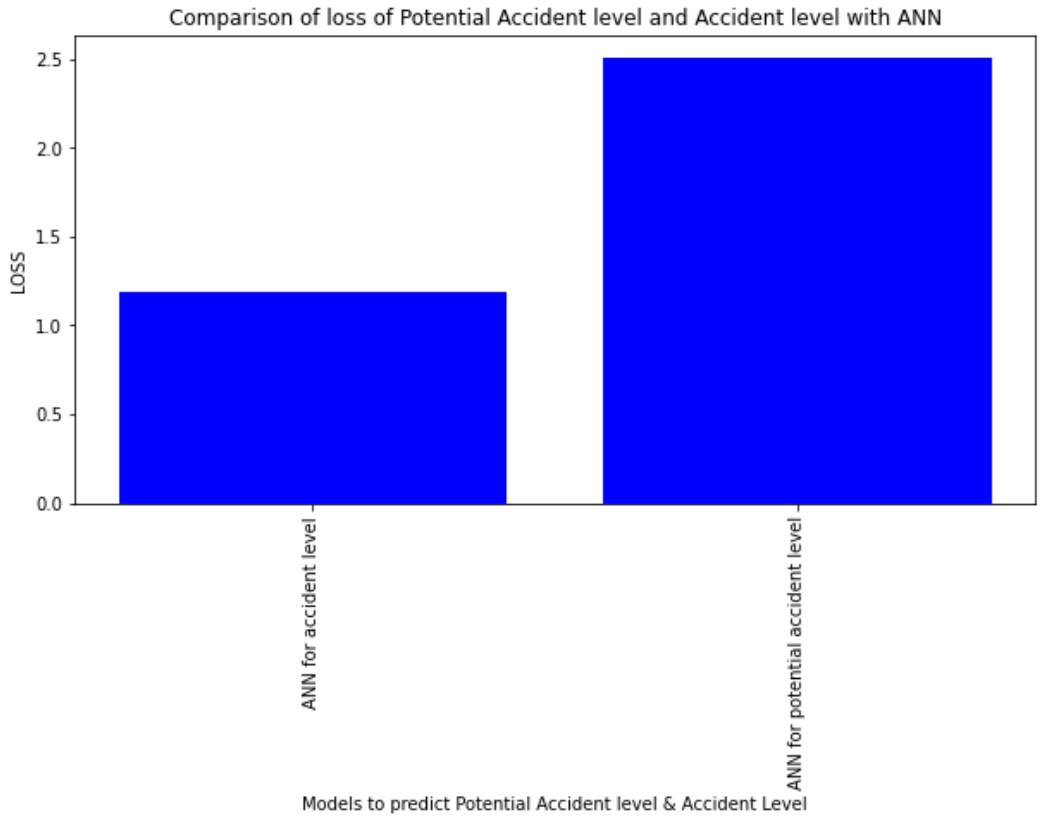


Comparing Loss of ANN with potential and accident level

```
#Comparing Loss of ANN with potential and accident level.
models1_tlist_accidentlevel_tf = ["ANN for accident level","ANN for potential accident level"]

losst1_accidentlevel_tf = [ann_loss,ann_loss1]
print(losst1_accidentlevel_tf)
```

```
[1.1904761791229248, 2.507936477661133]
```

```
#Bar plot
fig = plt.figure(figsize = (10, 5))
plt.bar(models1_tlist_accidentlevel_tf, losst1_accidentlevel_tf, color ='blue',width=0.8,)

plt.xlabel("Models to predict Potential Accident level & Accident Level")
plt.xticks(rotation = 'vertical')
plt.ylabel("LOSS")
plt.title("Comparison of loss of Potential Accident level and Accident level with ANN")
plt.show()
```



Design, train and test RNN or LSTM classifiers
LSTM for target variable accident level

```
df1=data1.copy()
```

```
country_encoder = LabelEncoder()
df1['Country'] = country_encoder.fit_transform(df1['Country'])

local_encoder = LabelEncoder()
df1['Local'] = local_encoder.fit_transform(df1['Local'])

industry_sector_encoder = LabelEncoder()
df1['Industry Sector'] = industry_sector_encoder.fit_transform(df1['Industry Sector'])

gender_encoder = LabelEncoder()
df1['Gender'] = gender_encoder.fit_transform(df1['Gender'])

employee_encoder = LabelEncoder()
df1['Natureofemployee'] = employee_encoder.fit_transform(df1['Natureofemployee'])

risk_encoder = LabelEncoder()
df1['Critical Risk'] = risk_encoder.fit_transform(df1['Critical Risk'])

acident_level_encoder = LabelEncoder()
df1['Accident Level'] = acident_level_encoder.fit_transform(df1['Accident Level'])

potentital_accident_level_encoder = LabelEncoder()
df1['Potential Accident Level'] = potentital_accident_level_encoder.fit_transform(df1['Potential Accident Level'])
```

```
from tensorflow.keras.layers import TextVectorization
```

```
text_vectorizer = TextVectorization()
```

```
# Get the vocabulary from the text vectorization layer
import numpy as np

words_in_vocab = text_vectorizer.get_vocabulary()
len(words_in_vocab), words_in_vocab[:10]
```

```
(2, ['', '[UNK]'])
```

AFTER LABEL ENCODING

```
df1.head(5) # after label ecoding
```

	Date	Country	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	Natureofemployee	Critical Risk	Description	Year	Month	Weekday	Season	processed_text	description_process
0	2016-01-01	0	0	1	0	3	1	1	20	While removing the drill rod of the Jumbo 08 f...	2016	1	Friday	Summer	[while, removing, drill, rod, jumbo, maintenan...	while removing d rod jum maintenance su
1	2016-01-02	1	1	1	0	3	1	0	21	During the activation of a sodium sulphide pum...	2016	1	Saturday	Summer	[during, activation, sodium, sulphide, pump, p...	during activati sodium sulphide pur piping
2	2016-01-06	0	2	1	0	2	1	2	15	In the sub-station MILPO located at level +170...	2016	1	Wednesday	Summer	[substation, milpo, located, level, collaborat...	substation mil located le collaborator e
3	2016-01-08	0	3	1	0	0	1	1	16	Being 9:45 am. approximately in the Nv. 1880 C...	2016	1	Friday	Summer	[being, approximately, personnel, begin, task,...	being approximat personnel begin ta unlo
4	2016-01-10	0	3	1	3	3	1	1	16	Approximately at 11:45 a.m. in circumstances t...	2016	1	Sunday	Summer	[approximately, am, circumstance, mechanic, an...	approximately a circumstance mecha anthon

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 16 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Date                      418 non-null    object
 1   Country                   418 non-null    int32 
 2   Local                     418 non-null    int64 
 3   Industry Sector           418 non-null    int32 
 4   Accident Level            418 non-null    int32 
 5   Potential Accident Level  418 non-null    int32 
 6   Gender                    418 non-null    int32 
 7   Natureofemployee          418 non-null    int32 
 8   Critical Risk             418 non-null    int32 
 9   Description               418 non-null    object
 10  Year                      418 non-null    int64 
 11  Month                     418 non-null    int64 
 12  Weekday                   418 non-null    object
 13  Season                    418 non-null    object
 14  processed_text            418 non-null    object
 15  description_processed     418 non-null    object
dtypes: int32(7), int64(3), object(6)
memory usage: 40.9+ KB
```

```
from keras.utils import np_utils
import numpy as np
X = data1['description_processed'].to_numpy()
uniques, ids = np.unique(df1['Accident Level'], return_inverse=True)

Y = np_utils.to_categorical(ids, len(uniques))

X.shape, Y.shape, ids.shape
```

```
((418,), (418, 5), (418,))
```

```
from sklearn.model_selection import train_test_split
num_classes = 5

# Use train_test_split to split training data into training and validation sets
X_train, X_test, Y_train, Y_test = train_test_split(X,
                                                    Y,
                                                    test_size=0.1, # dedicate 10% of samples to validation set
                                                    random_state=42) # random state for reproducibility
```

```
Y[0]
```

```
array([1., 0., 0., 0., 0.], dtype=float32)
```

```
X_train.shape, Y_train.shape
```

```
((376,), (376, 5))
```

```
# Fit the text vectorizer to the training text
text_vectorizer.adapt(X_train)
```

```
import tensorflow as tf
```

```
import random

# Choose a random sentence from the training dataset and tokenize it

random_sentence = random.choice(X_train)
print(f"Original text:\n{random_sentence}\
      \n\nVectorized version:")
text_vectorizer([random_sentence])
```

```
Original text:
during torch cutting activity new evaporator treatment fixing rupture hose near torch pen causing injury

Vectorized version:
<tf.Tensor: shape=(1, 15), dtype=int64, numpy=
array([[  24,  957,  135,   10, 1875, 2194,  951, 1234, 1045,  102,  166,
         957, 1107,    5,   11]], dtype=int64)>
```

```
# Get the unique words in the vocabulary
words_in_vocab = text_vectorizer.get_vocabulary()
top_5_words = words_in_vocab[:5] # most common tokens (notice the [UNK] token for "unknown" words)
bottom_5_words = words_in_vocab[-5:] # least common tokens
print(f"Number of words in vocab: {len(words_in_vocab)}")
print(f"Top 5 most common words: {top_5_words}")
print(f"Bottom 5 least common words: {bottom_5_words}")

max_vocab_length = len(words_in_vocab)
max_length = round(max([len(i.split()) for i in X_train]))
```

```
Number of words in vocab: 2598
Top 5 most common words: ['', '[UNK]', 'the', 'hand', 'employee']
Bottom 5 least common words: ['absorbing', 'absorbent', 'abratech', 'abdomen', 'abb']
```

```
max_length
```

```
95
```

```
import tensorflow as tf
from tensorflow.keras import layers
tf.random.set_seed(42)

embedding = layers.Embedding(input_dim=max_vocab_length, # set input shape
                             output_dim=128, # set size of embedding vector
                             embeddings_initializer="uniform", # default, intialize randomly
                             input_length=max_length, # how long is each input
                             name="embedding_1")

embedding
```

```
<keras.layers.embeddings.Embedding at 0x25b6c8c0070>
```

```
# Get a random sentence from training set
random_sentence = random.choice(X_train)
print(f"Original text:\n{random_sentence}\
      \n\nEmbedded version:")

# Embed the random sentence (turn it into numerical representation)
sample_embed = embedding(text_vectorizer([random_sentence]))
sample_embed
```

```
Original text:
when cutting vegetation open bite using sickle assistant struck vine twice when liana ruptured top branch projected face auxiliary causing cut upper l
ip

Embedded version:
<tf.Tensor: shape=(1, 23, 128), dtype=float32, numpy=
array([[[-0.03624376,  0.04096097,  0.00048   , ..., -0.0216538 ,
          0.00666716,  0.01672911],
        [-0.00767387,  0.03244089, -0.00273246, ...,  0.0495437 ,
         -0.02859933, -0.03865745],
        [ 0.03118776, -0.04945146,  0.00428406, ..., -0.01781521,
          0.03616799, -0.00525054],
        ...,
        [-0.01617707,  0.0422763 ,  0.00466955, ...,  0.0381538 ,
          0.03315575,  0.04610637],
        [-0.0357952 , -0.03596241,  0.0240985 , ..., -0.03533927,
         -0.00157645,  0.01281765],
        [ 0.04466719, -0.04735221,  0.02906582, ...,  0.01456075,
         -0.03965676,  0.0335649 ]]], dtype=float32)>
```

CREATING TENSORBOARD CALLBACK FOR EACH MODEL

```
# Create tensorboard callback (need to create a new one for each model)

def create_tensorboard_callback(dir_name, experiment_name):
  """
  Creates a TensorBoard callback instand to store log files.

  Stores log files with the filepath:
    "dir_name/experiment_name/current_datetime/"

  Args:
    dir_name: target directory to store TensorBoard log files
    experiment_name: name of experiment directory (e.g. efficientnet_model_1)
  """
  log_dir = dir_name + "/" + experiment_name + "/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
  tensorboard_callback = tf.keras.callbacks.TensorBoard(
      log_dir=log_dir
  )
  print(f"Saving TensorBoard log files to: {log_dir}")
  return tensorboard_callback

# Create directory to save TensorBoard logs
SAVE_DIR = "model_logs"
```

BUILDING MODEL WITH FUNCTIONAL API

```
# Build model with the Functional API

inputs = layers.Input(shape=(1,), dtype="string") # inputs are 1-dimensional strings
x = text_vectorizer(inputs) # turn the input text into numbers
x = embedding(x) # create an embedding of the numerized numbers
x = layers.GlobalAveragePooling1D()(x) # lower the dimensionality of the embedding (try running the model without this layer and see what happens)
outputs = layers.Dense(num_classes, activation="softmax")(x) # create the output layer, want binary outputs so use softmax activation
model_1 = tf.keras.Model(inputs, outputs, name="model_1_dense") # construct the model
```

```
# Compile model
model_1.compile(loss="categorical_crossentropy",
                optimizer=tf.keras.optimizers.Adam(),
                metrics=["accuracy"])
```

```
model_1.summary()
```

```
Model: "model_1_dense"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 1)]               0

 text_vectorization (TextVec  (None, None)             0
 torization)

 embedding_1 (Embedding)     (None, None, 128)         332544

 global_average_pooling1d (G  (None, 128)              0
 lobalAveragePooling1D)

 dense_10 (Dense)            (None, 5)                 645

=================================================================
Total params: 333,189
Trainable params: 333,189
Non-trainable params: 0
_________________________________________________________________
```

```
#Accuracy and loss of model_1
model1_loss, model1_accuracy = model_1.evaluate(X_test, Y_test, verbose=1)
print('Test loss:', model1_loss)
print('Test accuracy:', model1_accuracy)
```

```
2/2 [==============================] - 0s 3ms/step - loss: 0.8167 - accuracy: 0.7619
Test loss: 0.8166996836662292
Test accuracy: 0.761904776096344
```

LSTM

```
lstm_out = 128

model_2 = Sequential()
model_2.add(layers.Input(shape=(1,), dtype="string"))
model_2.add(text_vectorizer)
model_2.add(layers.Embedding(input_dim=max_vocab_length,
                                    output_dim=128,
                                    embeddings_initializer="uniform",
                                    input_length=max_length,
                                    name="embedding_2"))
model_2.add(layers.LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2,return_sequences=True))
model_2.add(layers.LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
model_2.add(layers.Dense(num_classes, activation="softmax")) # create the output layer, want binary outputs so use softmax activation
print(model_2.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization (TextVec  (None, None)             0
 torization)

 embedding_2 (Embedding)     (None, None, 128)         332544

 lstm (LSTM)                 (None, None, 128)         131584

 lstm_1 (LSTM)               (None, 128)               131584

 dense_11 (Dense)            (None, 5)                 645

=================================================================
Total params: 596,357
Trainable params: 596,357
Non-trainable params: 0
_________________________________________________________________
None
```

```
model_2.compile(loss="categorical_crossentropy",
                optimizer=tf.keras.optimizers.Adam(),
                metrics=["accuracy"])
```

```
model_2.compile(loss="categorical_crossentropy",
                optimizer=tf.keras.optimizers.Adam(),
                metrics=["accuracy"])
```

```
model_2.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization (TextVec  (None, None)             0
 torization)

 embedding_2 (Embedding)     (None, None, 128)         332544

 lstm (LSTM)                 (None, None, 128)         131584

 lstm_1 (LSTM)               (None, 128)               131584

 dense_11 (Dense)            (None, 5)                 645

=================================================================
Total params: 596,357
Trainable params: 596,357
Non-trainable params: 0
_________________________________________________________________
```

```
#Running the model
```

```
batch_size = 32
custom_early_stopping = EarlyStopping(monitor = 'val_loss', patience = 10, mode = 'min',
    restore_best_weights=True
)

# Fit model
model_2_history = model_2.fit(X_train,
                              Y_train,
                              epochs=50,
                              validation_data=(X_test, Y_test),
                              callbacks=[create_tensorboard_callback(SAVE_DIR,"LSTM"), custom_early_stopping])
```

```
Saving TensorBoard log files to: model_logs/LSTM/20220507-105237
Epoch 1/50
12/12 [==============================] - 6s 218ms/step - loss: 1.1610 - accuracy: 0.6809 - val_loss: 0.8939 - val_accuracy: 0.7619
Epoch 2/50
12/12 [==============================] - 2s 168ms/step - loss: 0.9246 - accuracy: 0.7367 - val_loss: 0.8631 - val_accuracy: 0.7619
Epoch 3/50
12/12 [==============================] - 2s 173ms/step - loss: 0.9101 - accuracy: 0.7367 - val_loss: 0.8656 - val_accuracy: 0.7619
Epoch 4/50
12/12 [==============================] - 2s 175ms/step - loss: 0.9042 - accuracy: 0.7367 - val_loss: 0.8287 - val_accuracy: 0.7619
Epoch 5/50
12/12 [==============================] - 2s 181ms/step - loss: 0.9027 - accuracy: 0.7394 - val_loss: 0.8167 - val_accuracy: 0.7619
Epoch 6/50
12/12 [==============================] - 2s 173ms/step - loss: 0.8508 - accuracy: 0.7367 - val_loss: 0.8203 - val_accuracy: 0.7619
Epoch 7/50
12/12 [==============================] - 2s 193ms/step - loss: 0.7692 - accuracy: 0.7473 - val_loss: 0.8426 - val_accuracy: 0.7619
Epoch 8/50
12/12 [==============================] - 3s 219ms/step - loss: 0.6975 - accuracy: 0.7713 - val_loss: 1.0731 - val_accuracy: 0.6190
Epoch 9/50
12/12 [==============================] - 3s 237ms/step - loss: 0.6203 - accuracy: 0.7793 - val_loss: 0.8589 - val_accuracy: 0.7619
Epoch 10/50
12/12 [==============================] - 3s 243ms/step - loss: 0.5130 - accuracy: 0.8032 - val_loss: 1.3898 - val_accuracy: 0.5714
Epoch 11/50
12/12 [==============================] - 2s 194ms/step - loss: 0.4814 - accuracy: 0.7793 - val_loss: 1.6652 - val_accuracy: 0.5714
Epoch 12/50
12/12 [==============================] - 2s 188ms/step - loss: 0.4421 - accuracy: 0.7899 - val_loss: 1.3940 - val_accuracy: 0.6429
Epoch 13/50
12/12 [==============================] - 2s 185ms/step - loss: 0.4206 - accuracy: 0.8191 - val_loss: 1.3856 - val_accuracy: 0.6429
Epoch 14/50
12/12 [==============================] - 3s 210ms/step - loss: 0.4957 - accuracy: 0.7553 - val_loss: 2.0305 - val_accuracy: 0.3810
Epoch 15/50
12/12 [==============================] - 3s 242ms/step - loss: 0.7633 - accuracy: 0.6755 - val_loss: 1.0410 - val_accuracy: 0.7619
```

```
#Evaluating the model_2
```

```
model_2.evaluate(X_test, Y_test)
```

```
2/2 [==============================] - 0s 16ms/step - loss: 0.8167 - accuracy: 0.7619
[0.8166620135307312, 0.761904776096344]
```

```
#Accuracy and loss of model_2
model2_loss, model2_accuracy = model_2.evaluate(X_test, Y_test, verbose=1)
print('Test loss:', model2_loss)
print('Test accuracy:', model2_accuracy)
```

```
2/2 [==============================] - 0s 14ms/step - loss: 0.8167 - accuracy: 0.7619
Test loss: 0.8166620135307312
Test accuracy: 0.761904776096344
```

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

```
Y_pred = model_2.predict(X_test)
Y_test_decoded = uniques[np.argmax(Y_test, 1)]
Y_pred_decoded = uniques[np.argmax(Y_pred, 1)]

# Classification Report
cr = classification_report(Y_test_decoded, Y_pred_decoded)
print("Classification Report:- ")
print(cr)
```

```
Classification Report:-
              precision    recall  f1-score   support

           0       0.78      1.00      0.88        32
           1       0.00      0.00      0.00         2
           2       0.00      0.00      0.00         2
           3       0.00      0.00      0.00         5
           4       0.00      0.00      0.00         1

    accuracy                           0.76        42
   macro avg       0.16      0.20      0.18        42
weighted avg       0.59      0.76      0.67        42
```

TUNING OF MODEL

```
#TUNING the model_2
#Changing the batch size and epochs
batch_size = 16
custom_early_stopping = EarlyStopping(monitor = 'val_loss', patience = 10, mode = 'min',
    restore_best_weights=True
)

# Fit model
modelt_2_history = model_2.fit(X_train,
                               Y_train,
                               epochs=50,
                               validation_data=(X_test, Y_test),
                               callbacks=[create_tensorboard_callback(SAVE_DIR,"LSTM"), custom_early_stopping])
```

```
Saving TensorBoard log files to: model_logs/LSTM/20220507-105459
Epoch 1/50
12/12 [==============================] - 3s 201ms/step - loss: 0.8693 - accuracy: 0.7420 - val_loss: 0.7905 - val_accuracy: 0.7857
Epoch 2/50
12/12 [==============================] - 2s 177ms/step - loss: 0.8565 - accuracy: 0.7500 - val_loss: 0.7911 - val_accuracy: 0.7857
Epoch 3/50
12/12 [==============================] - 2s 182ms/step - loss: 0.8102 - accuracy: 0.7553 - val_loss: 0.7483 - val_accuracy: 0.7619
Epoch 4/50
12/12 [==============================] - 2s 190ms/step - loss: 0.7535 - accuracy: 0.7553 - val_loss: 0.9219 - val_accuracy: 0.6667
Epoch 5/50
12/12 [==============================] - 2s 184ms/step - loss: 0.7875 - accuracy: 0.7287 - val_loss: 0.9045 - val_accuracy: 0.7619
Epoch 6/50
12/12 [==============================] - 2s 191ms/step - loss: 0.8273 - accuracy: 0.7234 - val_loss: 0.9535 - val_accuracy: 0.6667
Epoch 7/50
12/12 [==============================] - 2s 211ms/step - loss: 0.7630 - accuracy: 0.7154 - val_loss: 1.1423 - val_accuracy: 0.7619
Epoch 8/50
12/12 [==============================] - 2s 204ms/step - loss: 0.7694 - accuracy: 0.7367 - val_loss: 1.0938 - val_accuracy: 0.7619
Epoch 9/50
12/12 [==============================] - 3s 210ms/step - loss: 0.7950 - accuracy: 0.7367 - val_loss: 1.0475 - val_accuracy: 0.7619
Epoch 10/50
12/12 [==============================] - 2s 196ms/step - loss: 0.7937 - accuracy: 0.7367 - val_loss: 1.0246 - val_accuracy: 0.7619
Epoch 11/50
12/12 [==============================] - 2s 190ms/step - loss: 0.7713 - accuracy: 0.7367 - val_loss: 1.0057 - val_accuracy: 0.7619
Epoch 12/50
12/12 [==============================] - 2s 208ms/step - loss: 0.7726 - accuracy: 0.7367 - val_loss: 1.0463 - val_accuracy: 0.7619
Epoch 13/50
12/12 [==============================] - 2s 202ms/step - loss: 0.7611 - accuracy: 0.7367 - val_loss: 1.0693 - val_accuracy: 0.7619
```

```
model_2.evaluate(X_test, Y_test)
```

```
2/2 [==============================] - 0s 14ms/step - loss: 0.7483 - accuracy: 0.7619
[0.748318076133728, 0.761904776096344]
```

It can be noted that the accuracy of the model before and after tuning is the same.

BIDIRECTIONAL LSTM FOR TARGET VARIABLE ACCIDENT LEVEL

```
from keras.layers import LSTM
from keras.layers import Bidirectional
from tensorflow.python.keras.layers import GlobalMaxPool1D

lstm_out = 128

model_3 = Sequential()
model_3.add(layers.Input(shape=(1,), dtype="string"))
model_3.add(text_vectorizer)
model_3.add(layers.Embedding(input_dim=max_vocab_length,
                                    output_dim=128,
                                    embeddings_initializer="uniform",
                                    input_length=max_length,
                                    name="embedding_2"))
model_3.add(Bidirectional(LSTM(lstm_out, return_sequences = True)))
model_3.add(GlobalMaxPool1D())
model_3.add(Dropout(0.5, input_shape = (256,)))
model_3.add(Dense(128, activation = 'relu'))
model_3.add(Dropout(0.5, input_shape = (128,)))
model_3.add(Dense(64, activation = 'relu'))
model_3.add(Dropout(0.5, input_shape = (64,)))
model_3.add(layers.Dense(num_classes, activation="softmax")) # create the output layer, want binary outputs so use softmax activation
print(model_3.summary())
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization (TextVec  (None, None)             0
 torization)

 embedding_2 (Embedding)     (None, None, 128)         332544

 bidirectional (Bidirectiona  (None, None, 256)        263168
 l)

 module_wrapper (ModuleWrapp  (None, 256)              0
 er)

 dropout_8 (Dropout)         (None, 256)               0

 dense_12 (Dense)            (None, 128)               32896

 dropout_9 (Dropout)         (None, 128)               0

 dense_13 (Dense)            (None, 64)                8256

 dropout_10 (Dropout)        (None, 64)                0

 dense_14 (Dense)            (None, 5)                 325

=================================================================
Total params: 637,189
Trainable params: 637,189
Non-trainable params: 0
_________________________________________________________________
None
```

```
# Compile model
model_3.compile(loss="categorical_crossentropy",
                optimizer=tf.keras.optimizers.Adam(),
                metrics=["accuracy"])
```

```
batch_size = 32
custom_early_stopping = EarlyStopping(monitor = 'val_loss', patience = 10, mode = 'min',
    restore_best_weights=True
)

# Fit model
model_3_history = model_3.fit(X_train,
                              Y_train,
                              epochs=50,
                              validation_data=(X_test, Y_test),
                              callbacks=[create_tensorboard_callback(SAVE_DIR,"BI-LSTM"), custom_early_stopping])
```

```
Saving TensorBoard log files to: model_logs/BI-LSTM/20220507-105755
WARNING:tensorflow:Model failed to serialize as JSON. Ignoring...
Layer ModuleWrapper has arguments ['self', 'module', 'method_name']
in `__init__` and therefore must override `get_config()`.

Example:

class CustomLayer(keras.layers.Layer):
    def __init__(self, arg1, arg2):
        super().__init__()
        self.arg1 = arg1
        self.arg2 = arg2

    def get_config(self):
        config = super().get_config()
        config.update({
            "arg1": self.arg1,
            "arg2": self.arg2,
        })
        return config
Epoch 1/50
12/12 [==============================] - 5s 192ms/step - loss: 1.4655 - accuracy: 0.5559 - val_loss: 0.9183 - val_accuracy: 0.7619
Epoch 2/50
12/12 [==============================] - 1s 94ms/step - loss: 1.0669 - accuracy: 0.7340 - val_loss: 0.8605 - val_accuracy: 0.7619
Epoch 3/50
12/12 [==============================] - 1s 96ms/step - loss: 0.9919 - accuracy: 0.7314 - val_loss: 0.8918 - val_accuracy: 0.7619
Epoch 4/50
12/12 [==============================] - 1s 99ms/step - loss: 1.0221 - accuracy: 0.7367 - val_loss: 0.8908 - val_accuracy: 0.7619
Epoch 5/50
12/12 [==============================] - 1s 99ms/step - loss: 0.9949 - accuracy: 0.7367 - val_loss: 0.8807 - val_accuracy: 0.7619
Epoch 6/50
12/12 [==============================] - 1s 97ms/step - loss: 0.9770 - accuracy: 0.7367 - val_loss: 0.8688 - val_accuracy: 0.7619
Epoch 7/50
12/12 [==============================] - 1s 98ms/step - loss: 0.9756 - accuracy: 0.7367 - val_loss: 0.9199 - val_accuracy: 0.7619
Epoch 8/50
12/12 [==============================] - 1s 102ms/step - loss: 0.9536 - accuracy: 0.7367 - val_loss: 0.8739 - val_accuracy: 0.7619
Epoch 9/50
12/12 [==============================] - 1s 100ms/step - loss: 0.8619 - accuracy: 0.7367 - val_loss: 0.8624 - val_accuracy: 0.7619
Epoch 10/50
12/12 [==============================] - 1s 96ms/step - loss: 0.6471 - accuracy: 0.7367 - val_loss: 0.9333 - val_accuracy: 0.7619
Epoch 11/50
12/12 [==============================] - 1s 100ms/step - loss: 0.5223 - accuracy: 0.7367 - val_loss: 1.3871 - val_accuracy: 0.7619
Epoch 12/50
12/12 [==============================] - 1s 100ms/step - loss: 0.4524 - accuracy: 0.7553 - val_loss: 2.3493 - val_accuracy: 0.6905
```

```
model_3.evaluate(X_test, Y_test)
```

```
2/2 [==============================] - 0s 10ms/step - loss: 0.8605 - accuracy: 0.7619
[0.8605166673660278, 0.761904776096344]
```

```
#Accuracy and loss of model_3
model3_loss, model3_accuracy = model_3.evaluate(X_test, Y_test, verbose=1)
print('Test loss:', model3_loss)
print('Test accuracy:', model3_accuracy)
```

```
2/2 [==============================] - 0s 8ms/step - loss: 0.8605 - accuracy: 0.7619
Test loss: 0.8605166673660278
Test accuracy: 0.761904776096344
```

```
Y_pred = model_3.predict(X_test)
Y_test_decoded = uniques[np.argmax(Y_test, 1)]
Y_pred_decoded = uniques[np.argmax(Y_pred, 1)]

# Classification Report
cr = classification_report(Y_test_decoded, Y_pred_decoded)
print("Classification Report:- ")
print(cr)
```

```
Classification Report:-
              precision    recall  f1-score   support

           0       0.76      1.00      0.86        32
           1       0.00      0.00      0.00         2
           2       0.00      0.00      0.00         2
           3       0.00      0.00      0.00         5
           4       0.00      0.00      0.00         1

    accuracy                           0.76        42
   macro avg       0.15      0.20      0.17        42
weighted avg       0.58      0.76      0.66        42
```

LSTM with target variable potential accident level

```
from keras.utils import np_utils
X1 = data1['description_processed'].to_numpy()
uniques1, ids1 = np.unique(df1['Potential Accident Level'], return_inverse=True)

Y1 = np_utils.to_categorical(ids1, len(uniques1))

X1.shape, Y1.shape, ids1.shape
```

```
((418,), (418, 6), (418,))
```

```
from sklearn.model_selection import train_test_split
num_classes1 = 6

# Use train_test_split to split training data into training and validation sets
X1_train, X1_test, Y1_train, Y1_test = train_test_split(X1,
                                                        Y1,
                                                        test_size=0.1, # dedicate 10% of samples to validation set
                                                        random_state=42) # random state for reproducibility
```

```
Y1[0]
```

```
array([0., 0., 0., 1., 0., 0.], dtype=float32)
```

```
X1_train.shape, Y1_train.shape
```

```
((376,), (376, 6))
```

```
from tensorflow.keras.layers import TextVectorization
```

```
text_vectorizer1 = TextVectorization()
```

```
# Fit the text vectorizer to the training text
text_vectorizer1.adapt(X1_train)
```

```
import random

# Choose a random sentence from the training dataset and tokenize it

random_sentence1 = random.choice(X1_train)
print(f"Original text:\n{random_sentence1}\
      \n\nVectorized version:")
text_vectorizer1([random_sentence1])
```

```
Original text:
the technician magnetometric survey stepped thorn his reaction immediately retreat losing balance magnetometer antenna broke

Vectorized version:
<tf.Tensor: shape=(1, 15), dtype=int64, numpy=
array([[   2,   98,  816,  608,  291,  605, 1210,  139,  109, 1703, 1166,
         242, 1935, 2534, 1325]], dtype=int64)>
```

```
# Get the unique words in the vocabulary
words_in_vocab1 = text_vectorizer1.get_vocabulary()
top_5_words1 = words_in_vocab1[:5] # most common tokens (notice the [UNK] token for "unknown" words)
bottom_5_words1 = words_in_vocab1[-5:] # least common tokens
print(f"Number of words in vocab: {len(words_in_vocab1)}")
print(f"Top 5 most common words: {top_5_words1}")
print(f"Bottom 5 least common words: {bottom_5_words1}")

max_vocab_length1 = len(words_in_vocab1)
max_length1 = round(max([len(i.split()) for i in X1_train]))
```

```
Number of words in vocab: 2598
Top 5 most common words: ['', '[UNK]', 'the', 'hand', 'employee']
Bottom 5 least common words: ['absorbing', 'absorbent', 'abratech', 'abdomen', 'abb']
```

```
# Build model with the Functional API

inputs1 = layers.Input(shape=(1,), dtype="string") # inputs are 1-dimensional strings
x1 = text_vectorizer1(inputs1) # turn the input text into numbers
x1 = embedding(x1) # create an embedding of the numerized numbers
x1 = layers.GlobalAveragePooling1D()(x1) # lower the dimensionality of the embedding (try running the model without this layer and see what happens
outputs1 = layers.Dense(num_classes1, activation="softmax")(x1) # create the output layer, want binary outputs so use softmax activation
model1_1 = tf.keras.Model(inputs1, outputs1, name="model1_1_dense") # construct the model
```

```
# Compile model
modelp_1.compile(loss="categorical_crossentropy",
                 optimizer=tf.keras.optimizers.Adam(),
                 metrics=["accuracy"])
```

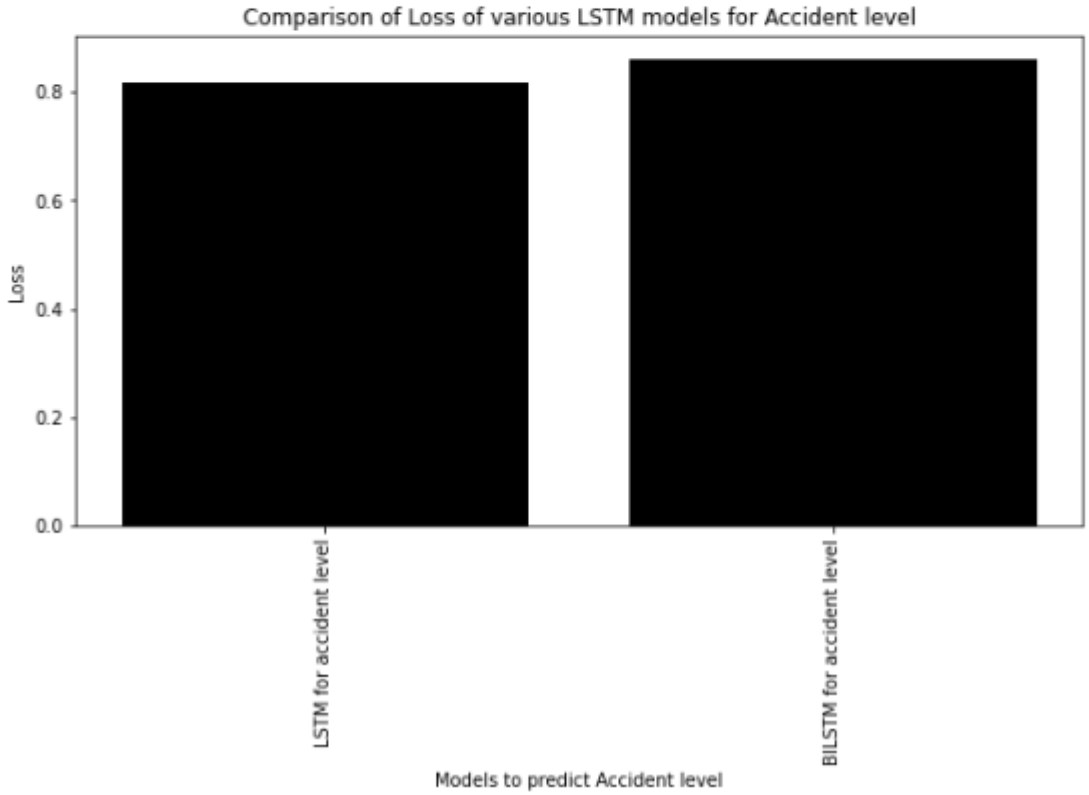
```
modelp_1.summary()
```

```
Model: "modelp_1_dense"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 1)]               0

 text_vectorization_1 (TextV  (None, None)             0
 ectorization)

 embedding_1 (Embedding)     (None, None, 128)         332544

 global_average_pooling1d_1   (None, 128)              0
 (GlobalAveragePooling1D)

 dense_15 (Dense)            (None, 6)                 774

=================================================================
Total params: 333,318
Trainable params: 333,318
Non-trainable params: 0
_________________________________________________________________
```

```
#Comparing all the BILSTM and LSTM models with target variable accident level
model_LSTM_accidentlevel_tf = ["LSTM for accident level","BILSTM for accident level"]

loss_LSTM_accidentlevel_tf = [model2_loss,model3_loss]
print(loss_LSTM_accidentlevel_tf)
```

```
[0.8166620135307312, 0.8605166673660278]
```

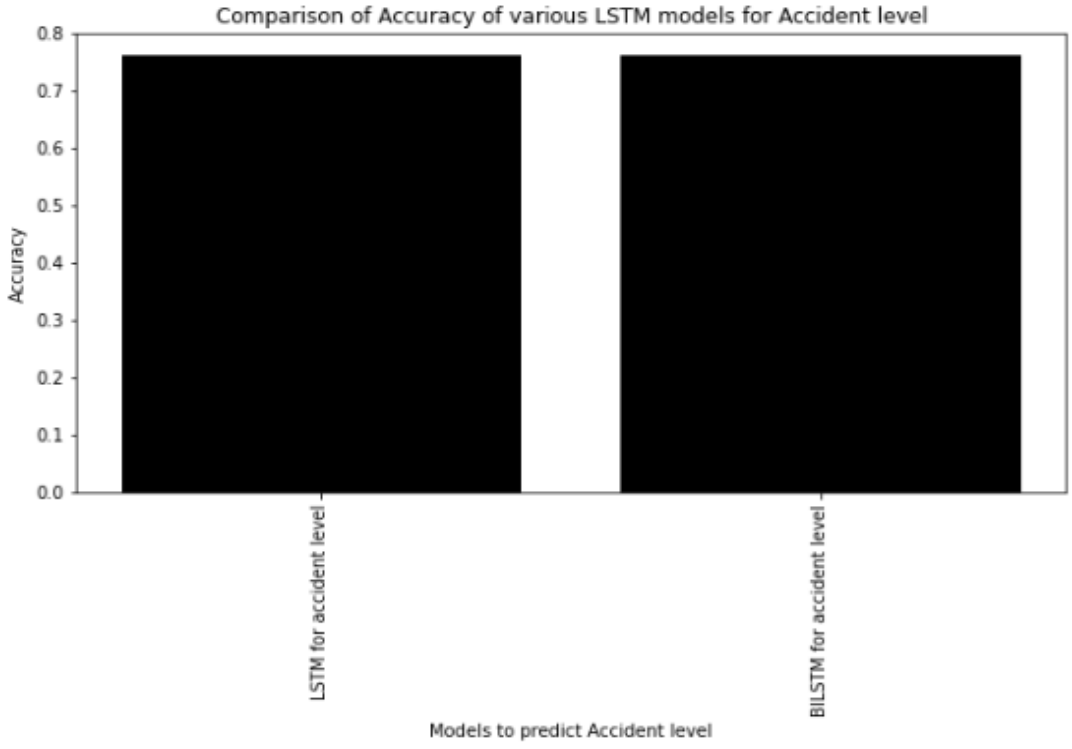
```
# creating the bar plot
fig = plt.figure(figsize = (10, 5))
plt.bar(model_LSTM_accidentlevel_tf,loss_LSTM_accidentlevel_tf, color ='black',width=0.8,)

plt.xlabel("Models to predict Accident level")
plt.xticks(rotation = 'vertical')
plt.ylabel("Loss")
plt.title("Comparison of Loss of various LSTM models for Accident level")
plt.show()
```



```
#Comparing accuracy
model_LSTM_a_accidentlevel_tf = ["LSTM for accident level","BILSTM for accident level"]

accuracy_LSTM_accidentlevel_tf = [model2_accuracy,model3_accuracy]
print(accuracy_LSTM_accidentlevel_tf)
```

```
[0.761904776096344, 0.761904776096344]
```

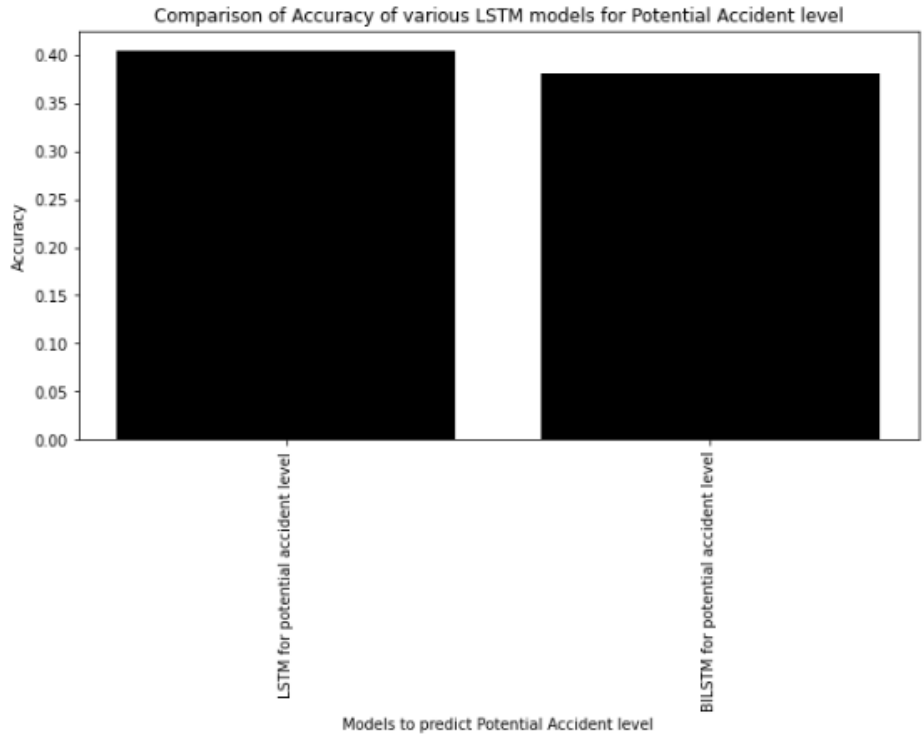
```
# creating the bar plot
fig = plt.figure(figsize = (10, 5))
plt.bar(model_LSTM_a_accidentlevel_tf,accuracy_LSTM_accidentlevel_tf, color ='black',width=0.8,)

plt.xlabel("Models to predict Accident level")
plt.xticks(rotation = 'vertical')
plt.ylabel("Accuracy")
plt.title("Comparison of Accuracy of various LSTM models for Accident level")
plt.show()
```



```
#Comparing accuracy
model_LSTM_a_paccidentlevel_tf = ["LSTM for potential accident level","BILSTM for potential accident level"]

accuracy_LSTM_paccidentlevel_tf = [model2t_accuracy,modelp_3_accuracy]
print(accuracy_LSTM_paccidentlevel_tf)
```

```
[0.4047619104385376, 0.380952388048172]
```

```
#Bar plot
fig = plt.figure(figsize = (10, 5))
plt.bar(model_LSTM_a_paccidentlevel_tf,accuracy_LSTM_paccidentlevel_tf, color ='black',width=0.8,)

plt.xlabel("Models to predict Potential Accident level")
plt.xticks(rotation = 'vertical')
plt.ylabel("Accuracy")
plt.title("Comparison of Accuracy of various LSTM models for Potential Accident level")
plt.show()
```



COMPARISON BENCHMARK

RESULT:

Among the above applied classifiers, the SVM model is performing the best for both the potential accident level and accident level as it is having high accuracy of about 77% for accident level and about 50% for potential accident level. The SVM model is performing better than the ANN classifier, LSTM and BILSTM classifiers. Hence, the SVM model is to be pickled for both potential accident level and accident level.

PICKLING OF MODEL

```
#Pickling the model
#For accident level
import pickle
filename_accidentlevel = 'finalized_accidentlevelmodel.sav'
pickle.dump(grid, open(filename_accidentlevel, 'wb'))
```

```
#For potential accidnet level
#Pickling
import pickle
filename_potentialaccidentlevel= 'finalized_potential_accidentlevelmodel.sav'
pickle.dump(grid_p, open(filename_potentialaccidentlevel, 'wb'))
```

Clickable UI to automate tasks performed under milestone 1 (EDA and importing the dataset).

Clickable UI for automating importing of the data-

The below code shown is executed with Tkinter and the below screenshot is the code for automating the process of importing the data.

CODE-

In [20]:
```
import tkinter as tk
from tkinter import ttk
import pandas as pd
import re
```

In [21]:
```
#App window
win=tk.Tk()
win.title('AUTOMATING TASKS OF MILESTONE 1')
```

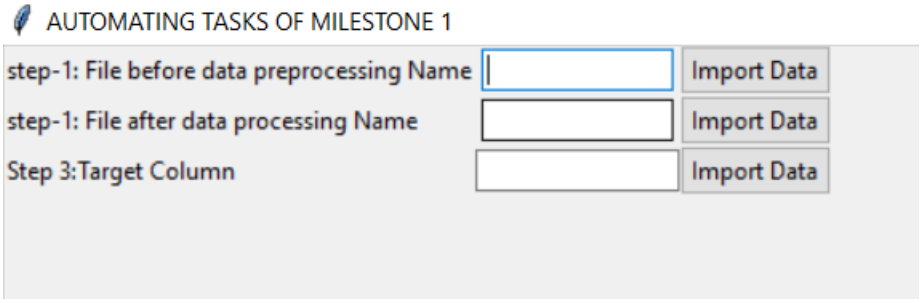
Out[21]:
```
''
```

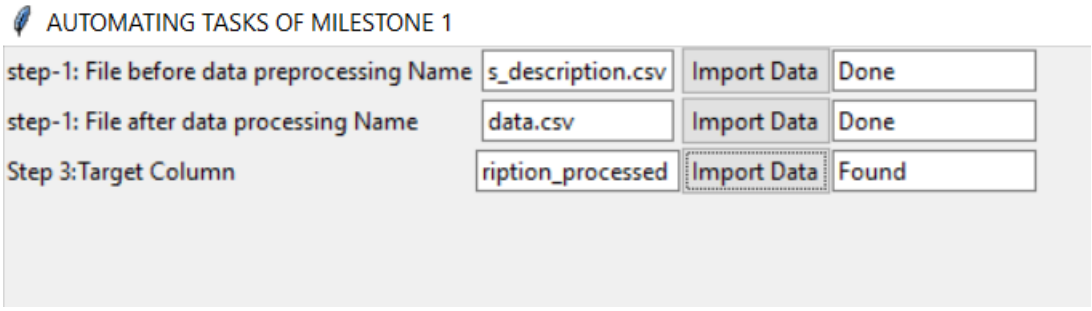
In [22]:
```
#Step 1 - Importing the data frame name
Name=ttk.Label(win,text="step-1: File before data preprocessing Name")
Name.grid(row=0,column=0,sticky=tk.W)
Name_var=tk.StringVar()
Name_entrybox=ttk.Entry(win,width=15,textvariable=Name_var)
Name_entrybox.grid(row=0,column=1)
def Import_Data():
    global DB
    DF_Name=Name_var.get()
    DB_extension=re.findall("\..*",DF_Name)
    if DB_extension==['.xlsx']:
        DB=pd.read_excel(DF_Name)
    elif DB_extension==['.csv']:
        DB=pd.read_csv(DF_Name)
    #Blank empty window to print confirmation
    confirm="Done"
    Confirm_entrybox=ttk.Entry(win,width=16)
    Confirm_entrybox.grid(row=0,column=3)
    Confirm_entrybox.insert(1,str(confirm))

Import_Data_Button=ttk.Button(win,text="Import Data",command=Import_Data)
Import_Data_Button.grid(row=0,column=2)
```

```
Name1=ttk.Label(win,text="step-1: File after data processing Name")
Name1.grid(row=1,column=0,sticky=tk.W)
Name1_var=tk.StringVar()
Name1_entrybox=ttk.Entry(win,width=15,textvariable=Name1_var)
Name1_entrybox.grid(row=1,column=1)
def Import_Data1():
    global DB1
    DF_Name1=Name1_var.get()
    DB_extension1=re.findall("\..*",DF_Name1)
    if DB_extension1==['.xlsx']:
        DB1=pd.read_excel(DF_Name1)
    elif DB_extension1==['.csv']:
        DB1=pd.read_csv(DF_Name1)
    #Blank empty window to print confirmation
    confirm="Done"
    Confirm_entrybox=ttk.Entry(win,width=16)
    Confirm_entrybox.grid(row=1,column=3)
    Confirm_entrybox.insert(1,str(confirm))

Import_Data1_Button=ttk.Button(win,text="Import Data",command=Import_Data1)
Import_Data1_Button.grid(row=1,column=2)
```

```
#Step 3- Target Data frame name
Target=ttk.Label(win,text="Step 3:Target Column")
Target.grid(row=2,column=0,sticky=tk.W)

Target_var=tk.StringVar()
Target_entrybox=ttk.Entry(win,width=16,textvariable=Target_var)
Target_entrybox.grid(row=2,column=1)

def Target_Data():
    global DB,Xc,y,Target_Name,Xc_train,y_train,Xc_test,y_test
    Target_Name=Target_var.get()

    Column_name=DB.columns
    Column_name
    found=0

    for i in range(len(Column_name)):
        if Column_name[i]==Target_Name:
            confirm="found"
            y=DB[Target_Name]
            Xc=DB.drop[Target_Name,axis==1]
            #Train and test split
            from sklearn.model_selection import train_test_split
            Xc_train,Xc_test,y_train,y_test=train_test_split(Xc,y,test_size=0.20,random_state=0,shuffle=True)
        else:
            confirm="Not Found"


    confirm="Found"
    Confirm_entrybox=ttk.Entry(win,width=16)
    Confirm_entrybox.grid(row=2,column=3)
    Confirm_entrybox.insert(1,str(confirm))

Target_Data_Button=ttk.Button(win,text="Import Data",command=Target_Data)
Target_Data_Button.grid(row=2,column=2)
win.mainloop()
```

OUTPUT-





Clickable UI for automating EDA-

From the below-shown code, it is evident that the automation of the univariate and bivariate analysis is performed with the clickable UI Tkinter.

CODE SCREENSHOTS-

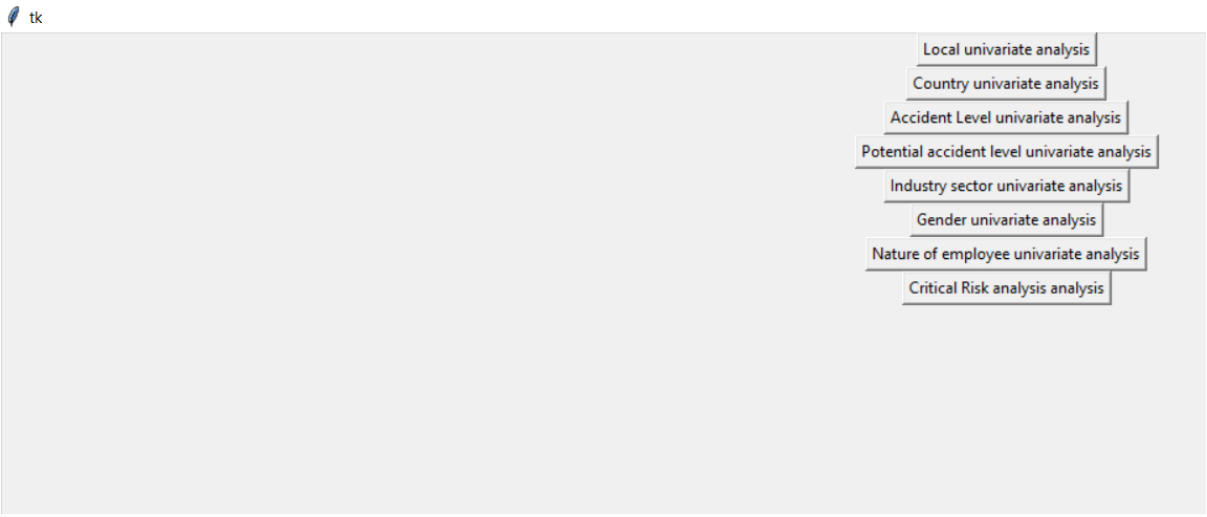
In [23]:
```
from tkinter import *
#Creating a win
win1 = Tk()
#Giving a Function To The Button
def univariate_analysis_categorical(dataset,feature):
    print("\n")
    print("=============================================================================================")
    print("Data Analysis of feature: ",feature)
    print("=============================================================================================\n")

    print("\n")
    print("-----------------")
    print("Countplot  for feature: ",feature)
    print("-----------------")

    plt.figure(figsize=(10,6))
    sns.countplot(dataset[feature],order = dataset[feature].value_counts().index)
    plt.xticks(rotation = 'vertical')
    plt.show()

    print("-----------------")
    print("Pie Chart for feature: ",feature)
    print("-----------------")

    labels=dataset[feature].unique()
    plt.figure(figsize=(10,6))
    dataset[feature].value_counts().plot.pie(autopct="%.1f%%")
    plt.show()
```

```
    print("\n")
    print("-----------------")
    print("Value Counts for feature: ",feature)
    print("--------------------")

    print(dataset[feature].value_counts().sort_values(ascending=False))
    print('')
def btn():
     univariate_analysis_categorical(data,'Local')



  #Creating The Button
button1 =  Button(win1, text="Local univariate analysis", command=btn)
#put on screen
button1.pack()
def btn1():
     univariate_analysis_categorical(data,'Country')



  #Creating The Button
button2 =  Button(win1, text="Country univariate analysis", command=btn1)
#put on screen
button2.pack()
def btn2():
     univariate_analysis_categorical(data,'Accident Level')



  #Creating The Button
button3 =  Button(win1, text="Accident Level univariate analysis", command=btn2)
#put on screen
button3.pack()
```

```
def btn3():
     univariate_analysis_categorical(data,'Potential Accident Level')



  #Creating The Button
button4 =  Button(win1, text="Potential accident level univariate analysis", command=btn3)
#put on screen
button4.pack()

button3.pack()
def btn4():
     univariate_analysis_categorical(data,'Industry Sector')



  #Creating The Button
button5 =  Button(win1, text="Industry sector univariate analysis", command=btn4)
#put on screen
button5.pack()
def btn5():
     univariate_analysis_categorical(data,'Gender')



  #Creating The Button
button6 =  Button(win1, text="Gender univariate analysis", command=btn5)
#put on screen
button6.pack()
def btn6():
     univariate_analysis_categorical(data,'Natureofemployee')
```

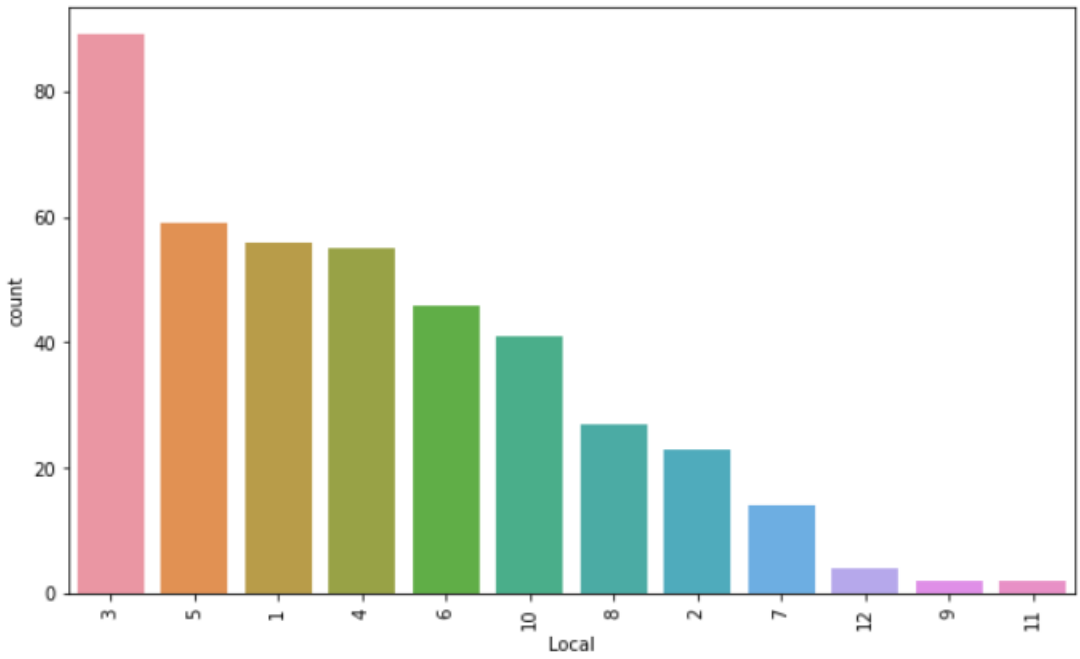
```
  #Creating The Button
button7 =  Button(win1, text="Nature of employee univariate analysis", command=btn6)
#put on screen
button7.pack()
def btn7():
     univariate_analysis_categorical(data,'Critical Risk')



  #Creating The Button
button8 =  Button(win1, text="Critical Risk analysis analysis", command=btn7)
#put on screen
button8.pack()




win1.mainloop()
```

OUTPUT SCREENSHOTS-

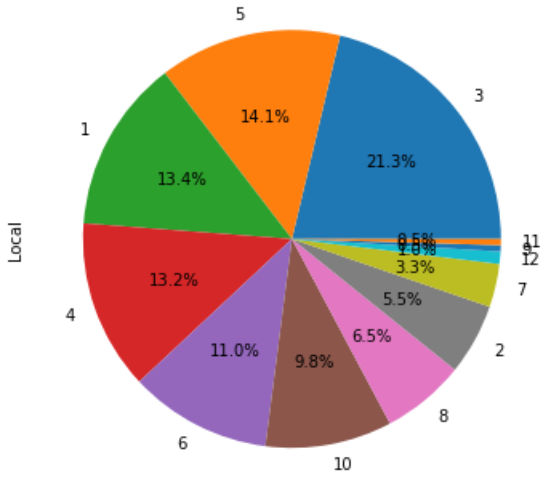


```
================================================================================
Data Analysis of feature:  Local
================================================================================


-----------------
Countplot  for feature:  Local
-----------------
```

```
D:\anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass th
ersion 0.12, the only valid positional argument will be `data`, and passing oth
sult in an error or misinterpretation.
  warnings.warn(
```



```
-----------------
Pie Chart for feature:  Local
------------------
```




```
-----------------
Value Counts for feature:  Local
-------------------
3     89
5     59
1     56
4     55
6     46
10    41
8     27
2     23
7     14
12     4
9      2
11     2
Name: Local, dtype: int64
```

UI For bivariate analysis:

CODE-

```
from tkinter import *
#Creating a win
win2 = Tk()
#Giving a Function To The Button
def bivariate_analysis_categorical(data,feature1,feature2):


            if feature1 != feature2:


                print("\n")
                print("=============================================================================================")
                print("Data Analysis of features: ",feature1,' and  ', feature2)
                print("=============================================================================================")


                bivariate_analysis_df = pd.crosstab(index=data[feature1],
                                            columns=data[feature2])

                print("\n")
                print("-------------------------------------------")
                print("Cross table Analysis of features: ",feature1,' and  ', feature2)
                print("-------------------------------------------")

                display(bivariate_analysis_df)

                print("\n")
                print("-------------------------------------------")
                print("Count plot Analysis of features: ",feature1,' and  ', feature2)
                print("-------------------------------------------")

                plt.figure(figsize=(12,6))
                sns.countplot(x=feature1, hue=feature2, data=data)
                plt.show()
```

```
def b():
     bivariate_analysis_categorical(data,'Country','Accident Level')



  #Creating The Button
b1 =  Button(win2, text="Country vs Accident Level bivariate analysis", command=b)
#put on screen
b1.pack()

def b1():
     bivariate_analysis_categorical(data,'Country','Potential Accident Level')



  #Creating The Button
b2 =  Button(win2, text="Country vs Potential Accident Level bivariate analysis", command=b1)
#put on screen
b2.pack()

def b2():
     bivariate_analysis_categorical(data,'Country','Local')



  #Creating The Button
b3 =  Button(win2, text="Country vs Local bivariate analysis", command=b2)
#put on screen
b3.pack()

def b3():
     bivariate_analysis_categorical(data,'Country','Industry Sector')



  #Creating The Button
b4 =  Button(win2, text="Country vs Accident Level bivariate analysis", command=b3)
#put on screen
b4.pack()

def b4():
     bivariate_analysis_categorical(data,'Country','Critical Risk')
```

```
  #Creating The Button
b5 =  Button(win2, text="Country vs Critical Risk bivariate analysis", command=b4)
#put on screen
b5.pack()

def b5():
     bivariate_analysis_categorical(data,'Country','Gender')



  #Creating The Button
b6 =  Button(win2, text="Country vs Gender bivariate analysis", command=b5)
#put on screen
b6.pack()

def b6():
     bivariate_analysis_categorical(data,'Industry Sector','Gender')



  #Creating The Button
b7 =  Button(win2, text="Industry Sector vs Gender bivariate analysis", command=b6)
#put on screen
b7.pack()

def b8():
     bivariate_analysis_categorical(data,'Industry Sector','Accident Level')



  #Creating The Button
b8 =  Button(win2, text="Industry Sector vs Accident Level bivariate analysis", command=b7)
#put on screen
b8.pack()

def b9():
     bivariate_analysis_categorical(data,'Industry Sector',' Potential Accident Level')



  #Creating The Button
b9 =  Button(win2, text="Industry Sector vs  Potential Accident Level bivariate analysis", command=b8)
#put on screen
b9.pack()
```

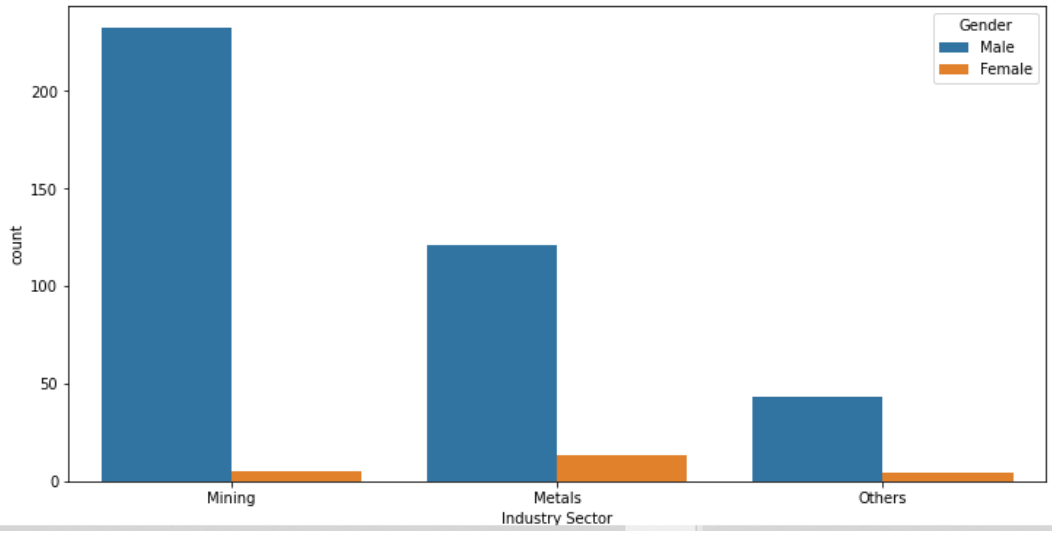
```
def b9():
     bivariate_analysis_categorical(data,'Industry Sector','Critical Risk')



  #Creating The Button
b10 =  Button(win2, text="Industry Sector vs Critical Risk bivariate analysis", command=b9)
#put on screen
b10.pack()

def b10():
     bivariate_analysis_categorical(data,'Industry Sector','Local')



  #Creating The Button
b11 =  Button(win2, text="Industry Sector vs Local bivariate analysis", command=b10)
#put on screen
b11.pack()


win2.mainloop()
```

OUTPUT SCREENSHOTS-

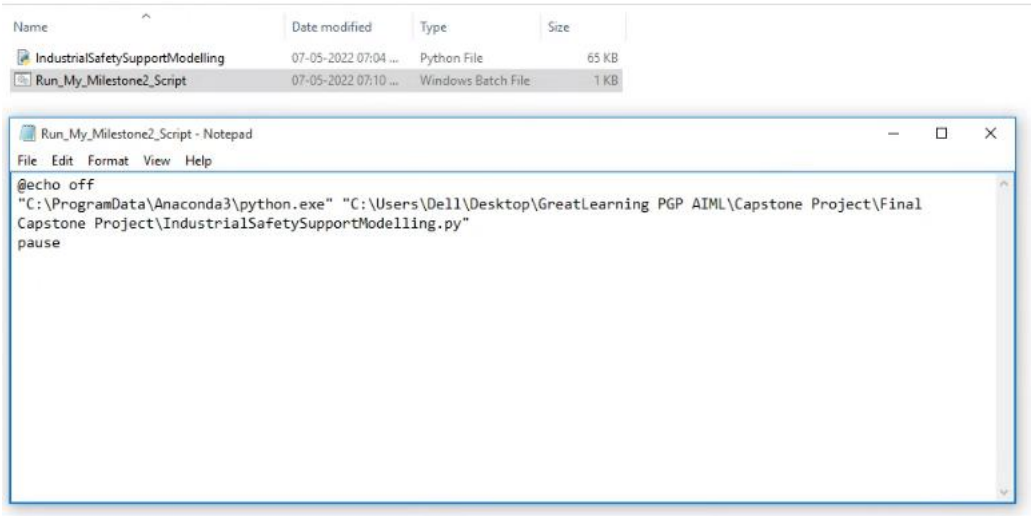

```
==========================================================================================
Data Analysis of features:  Industry Sector  and   Gender
==========================================================================================


------------------------------------------
Cross table Analysis of features:  Industry Sector  and   Gender
------------------------------------------
```

Gender	Female	Male
Industry Sector		
Metals	13	121
Mining	5	232
Others	4	43

```
------------------------------------------
Count plot Analysis of features:  Industry Sector  and   Gender
------------------------------------------
```

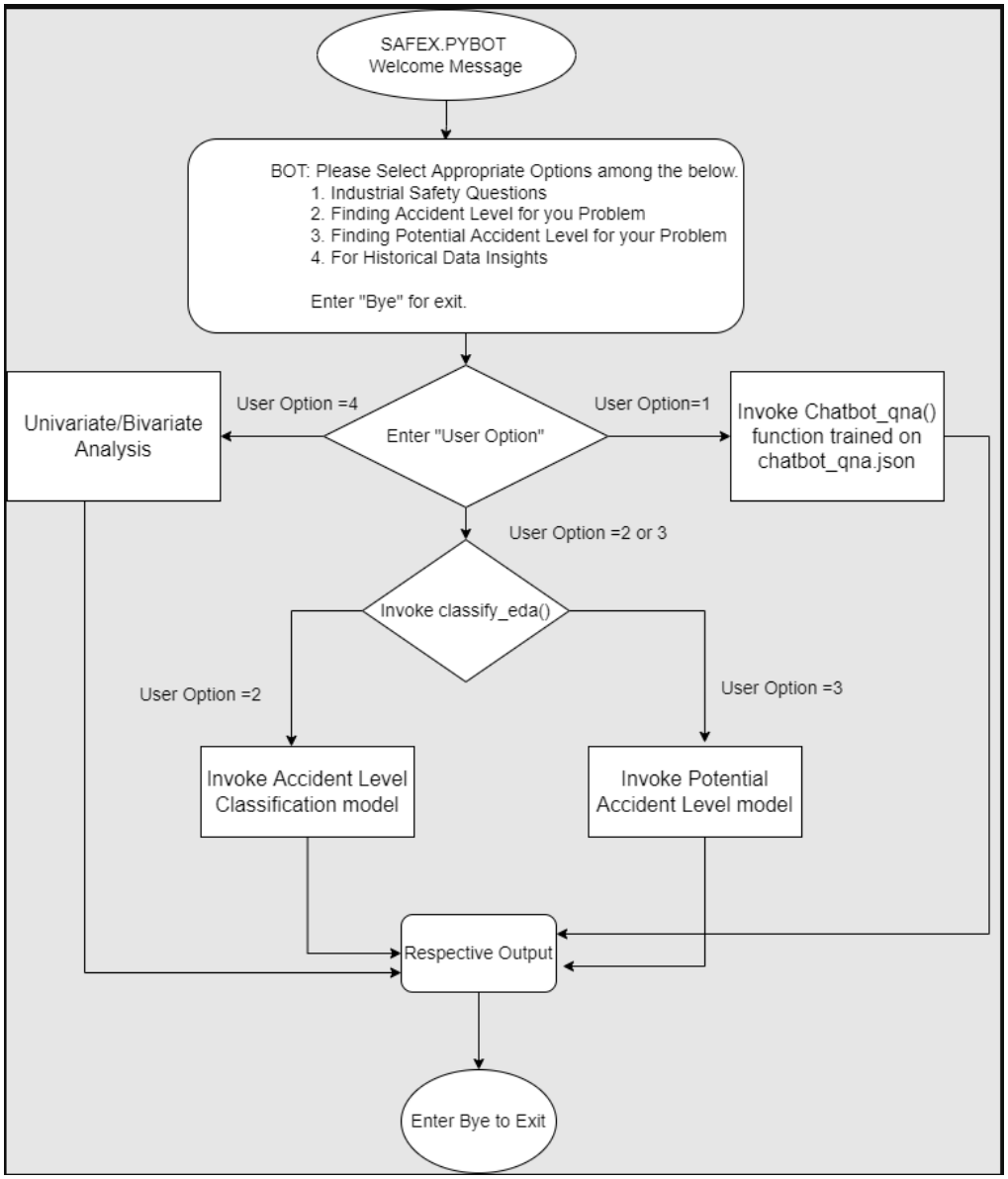


Clickable batch file to automatically run the model file (milestone 2)



CHATBOT

Flow diagram of Chatbot functioning



```
#Importing the data and files
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from scipy import stats; from scipy.stats import zscore, norm, randint

import warnings
warnings.filterwarnings("ignore")
```

```
# import the libraries

import nltk
from nltk.stem.lancaster import LancasterStemmer
stemmer = LancasterStemmer()

# things we need for Tensorflow

import tensorflow as tf
import tflearn
import random
```

```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\compat\v2_compat.py:111: disable_resource_variables (from tensorf
low.python.ops.variable_scope) is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
curses is not supported on this machine (please install/reinstall curses for an optimal experience)
```

```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Dell\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

Modelling and functions for Question and Answers

Importing our chat-bot intents file

```
# import our chat-bot intents file
import json
with open('GLSafexInfo.json') as json_data:
    intents = json.load(json_data)
```

Initializing data fields for our file, stemming and removing of duplicates

```
words = []
classes = []
documents = []
ignore_words = ['?']
# loop through each sentence in our intents patterns
for intent in intents['intents']:
    for pattern in intent['patterns']:
        # tokenize each word in the sentence
        w = nltk.word_tokenize(pattern)
        # add to our words list
        words.extend(w)
        # add to documents in our corpus
        documents.append((w, intent['tag']))
        # add to our classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])

# stem and lower each word and remove duplicates
words = [stemmer.stem(w.lower()) for w in words if w not in ignore_words]
words = sorted(list(set(words)))

# remove duplicates
classes = sorted(list(set(classes)))
```

Training Data

```
# create our training data
training = []
output = []
# create an empty array for our output
output_empty = [0] * len(classes)

# training set, bag of words for each sentence
for doc in documents:
    # initialize our bag of words
    bag = []
    # list of tokenized words for the pattern
    pattern_words = doc[0]
    # stem each word
    pattern_words = [stemmer.stem(word.lower()) for word in pattern_words]
    # create our bag of words array
    for w in words:
        bag.append(1) if w in pattern_words else bag.append(0)

    # output is a '0' for each tag and '1' for current tag
    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1

    training.append([bag, output_row])

# shuffle our features and turn into np.array
random.shuffle(training)
training = np.array(training)

# create train and test lists
X_train = list(training[:,0])
y_train = list(training[:,1])
```

```
# reset underlying graph data
from tensorflow.python.framework import ops
ops.reset_default_graph()

# Build neural network
net_qna = tflearn.input_data(shape=[None, len(X_train[0])])
net_qna = tflearn.fully_connected(net_qna, 16)
net_qna = tflearn.fully_connected(net_qna, 16)
net_qna = tflearn.fully_connected(net_qna, len(y_train[0]), activation='softmax')
net_qna = tflearn.regression(net_qna)

# Define model and setup tensorboard
model_qna = tflearn.DNN(net_qna, tensorboard_dir='tflearn_logs')

# Start training (apply gradient descent algorithm)
model_qna.fit(X_train, y_train, n_epoch=1000, batch_size=8, show_metric=True)
model_qna.save('model_qna.tflearn')
```

```
Training Step: 12999  | total loss: 0.10005 | time: 0.038s
| Adam | epoch: 1000 | loss: 0.10005 - acc: 0.9912 -- iter: 096/101
Training Step: 13000  | total loss: 0.09260 | time: 0.038s
| Adam | epoch: 1000 | loss: 0.09260 - acc: 0.9921 -- iter: 101/101
--
INFO:tensorflow:C:\Users\Dell\Desktop\GreatLearning PGP AIML\Capstone Project\Milestone 2\Apoorv Working\model_qna.tflearn is not in all_model_checkpo
int_paths. Manually adding it.
```

```
import pickle
pickle.dump( {'words':words, 'classes':classes, 'X_train':X_train, 'y_train':y_train}, open( "qna_training_data", "wb" ) )
```

```
# restore all of our data structures
import pickle
data_qna = pickle.load( open( "qna_training_data", "rb" ) )
words_qna = data_qna['words']
classes_qna = data_qna['classes']
X_train_qna = data_qna['X_train']
y_train_qna = data_qna['y_train']

# import our chat-bot intents file
import json
with open('GLSafexInfo.json') as json_data:
    intents_qna = json.load(json_data)
```

```
# load our saved model
model_qna.load('./model_qna.tflearn')
```

```
INFO:tensorflow:Restoring parameters from C:\Users\Dell\Desktop\GreatLearning PGP AIML\Capstone Project\Milestone 2\Apoorv Working\model_qna.tflearn
```

Functions for cleansing the sentences and to get the bag of words

```
# functions for cleansing the sentenses and to get the bag of words

def clean_up_sentence(sentence):
    # tokenize the pattern
    sentence_words = nltk.word_tokenize(sentence)
    # stem each word
    sentence_words = [stemmer.stem(word.lower()) for word in sentence_words]
    return sentence_words

# return bag of words array: 0 or 1 for each word in the bag that exists in the sentence
def bow(sentence, words, show_details=False):
    # tokenize the pattern
    sentence_words = clean_up_sentence(sentence)
    # bag of words
    bag = [0]*len(words)
    for s in sentence_words:
        for i,w in enumerate(words):
            if w == s:
                bag[i] = 1
                if show_details:
                    print ("found in bag: %s" % w)

    return(np.array(bag))
```

Functions to classify the sentences and for respective responses
Creating a data structure to hold user context

```
# functions to classify the sentences and for respective responses
# create a data structure to hold user context

context = {}

ERROR_THRESHOLD = 0.25
def classify(sentence):
    # generate probabilities from the model
    results = model_qna.predict([bow(sentence, words_qna)])[0]
    # filter out predictions below a threshold
    results = [[i,r] for i,r in enumerate(results) if r>ERROR_THRESHOLD]
    # sort by strength of probability
    results.sort(key=lambda x: x[1], reverse=True)
    return_list = []
    for r in results:
        return_list.append((classes_qna[r[0]], r[1]))
    # return tuple of intent and probability
    return return_list


def response(sentence, show_details=False):
    results = classify(sentence)
    # if we have a classification then find the matching intent tag
    if results:
        # loop as long as there are matches to process
        while results:
            for i in intents_qna['intents']:
                # find a tag matching the first result
                if i['tag'] == results[0][0]:
                    # a random response from the intent
                    return print('BOT: ', random.choice(i['responses']))

            results.pop(0)
```

```
def show_eda(response_result):
    eda_flag = True
    result = list(response_result.split(","))
    result_items = len(result)

    for item in result:
        if item not in dataset_features:
            eda_flag = False

    if eda_flag == True:
        if result_items == 1:
            univariate_analysis_categorical(dataset,result[0])
        elif result_items == 2:

            bivariate_analysis_categorical(dataset,result[0],result[1])
        else:
            print("Sorry, not able to find appropriate answer. Please ask teh question again")
    else:
        print(response_result)
```

Chatbot: Historical Data Insights

```
dataset= pd.read_csv('data.csv')
print("Shape of the dataset is :",dataset.shape)
dataset.head()
```

```
Shape of the dataset is : (418, 14)
```

	Date	Country	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	Natureofemployee	Critical Risk	Description	Year	Month	Weekday	Season
0	2016-01-01	Country_01	1	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...	2016	1	Friday	Summer
1	2016-01-02	Country_02	2	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...	2016	1	Saturday	Summer
2	2016-01-06	Country_01	3	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...	2016	1	Wednesday	Summer
3	2016-01-08	Country_01	4	Mining	I	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 C...	2016	1	Friday	Summer
4	2016-01-10	Country_01	4	Mining	IV	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances t...	2016	1	Sunday	Summer

Creating a list of possible features allowed for showing the data insights

```
dataset_features=['Country', 'Local', 'Industry Sector', 'Accident Level',
       'Potential Accident Level', 'Gender', 'Natureofemployee','Year']
```

Univariate Analysis

```
def univariate_analysis_categorical(dataset,feature):
    print("\n")
    print("=============================================================================================")
    print("Data Analysis of feature: ",feature)
    print("=============================================================================================\n")

    print("\n")
    print("-----------------")
    print("Countplot  for feature: ",feature)
    print("-----------------")

    plt.figure(figsize=(10,6))
    sns.countplot(dataset[feature],order = dataset[feature].value_counts().index)
    plt.xticks(rotation = 'vertical')
    plt.show()

    print("-----------------")
    print("Pie Chart for feature: ",feature)
    print("------------------")

    labels=dataset[feature].unique()
    plt.figure(figsize=(10,6))
    dataset[feature].value_counts().plot.pie(autopct="%.1f%%")
    plt.show()


    print("\n")
    print("-----------------")
    print("Value Counts for feature: ",feature)
    print("-------------------")

    print(dataset[feature].value_counts().sort_values(ascending=False))
    print('')
```

Bivariate Analysis

```
def bivariate_analysis_categorical(dataset,feature1,feature2):


            if feature1 != feature2:


                print("\n")
                print("=============================================================================================")
                print("Data Analysis of features: ",feature1,' and  ', feature2)
                print("=============================================================================================")


                bivariate_analysis_df = pd.crosstab(index=dataset[feature1],
                                          columns=dataset[feature2])

                print("\n")
                print("-------------------------------------------")
                print("Cross table Analysis of features: ",feature1,' and  ', feature2)
                print("-------------------------------------------")

                display(bivariate_analysis_df)

                print("\n")
                print("-------------------------------------------")
                print("Count plot Analysis of features: ",feature1,' and  ', feature2)
                print("-------------------------------------------")

                plt.figure(figsize=(12,6))
                sns.countplot(x=feature1, hue=feature2, data=dataset)
                plt.show()
```

Modelling and Functions for Historical Data Insights through chatbot

```
# import our chat-bot intents file
import json
with open('GLSafexInfo_eda.json') as json_data:
    intents_eda = json.load(json_data)

# Initialize data fields for our file

words = []
classes = []
documents = []
ignore_words = ['?']
# loop through each sentence in our intents patterns
for intent in intents_eda['intents']:
    for pattern in intent['patterns']:
        # tokenize each word in the sentence
        w = nltk.word_tokenize(pattern)
        # add to our words list
        words.extend(w)
        # add to documents in our corpus
        documents.append((w, intent['tag']))
        # add to our classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])

# stem and lower each word and remove duplicates
words = [stemmer.stem(w.lower()) for w in words if w not in ignore_words]
words = sorted(list(set(words)))

# remove duplicates
classes = sorted(list(set(classes)))

print (len(documents), "documents")
print (len(classes), "classes", classes)
print (len(words), "unique stemmed words", words)


# create our training data
training = []
output = []
# create an empty array for our output
output_empty = [0] * len(classes)
```

```
# training set, bag of words for each sentence
for doc in documents:
    # initialize our bag of words
    bag = []
    # list of tokenized words for the pattern
    pattern_words = doc[0]
    # stem each word
    pattern_words = [stemmer.stem(word.lower()) for word in pattern_words]
    # create our bag of words array
    for w in words:
        bag.append(1) if w in pattern_words else bag.append(0)

    # output is a '0' for each tag and '1' for current tag
    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1

    training.append([bag, output_row])

# shuffle our features and turn into np.array
random.shuffle(training)
training = np.array(training)

# create train and test lists
X_train = list(training[:,0])
y_train = list(training[:,1])


# reset underlying graph data
from tensorflow.python.framework import ops
ops.reset_default_graph()

# Build neural network
net_eda = tflearn.input_data(shape=[None, len(X_train[0])])
net_eda = tflearn.fully_connected(net_eda, 16)
net_eda = tflearn.fully_connected(net_eda, 16)
net_eda = tflearn.fully_connected(net_eda, len(y_train[0]), activation='softmax')
net_eda = tflearn.regression(net_eda)

# Define model and setup tensorboard
model_eda = tflearn.DNN(net_eda, tensorboard_dir='tflearn_logs')

# Start training (apply gradient descent algorithm)
model_eda.fit(X_train, y_train, n_epoch=1000, batch_size=8, show_metric=True)
model_eda.save('model_eda.tflearn')


import pickle
pickle.dump( {'words':words, 'classes':classes, 'X_train':X_train, 'y_train':y_train}, open( "eda_training_data", "wb" ) )
```

```
Training Step: 12999  | total loss: 0.12146 | time: 0.056s
| Adam | epoch: 1000 | loss: 0.12146 - acc: 0.9921 -- iter: 096/101
Training Step: 13000  | total loss: 0.11350 | time: 0.064s
| Adam | epoch: 1000 | loss: 0.11350 - acc: 0.9929 -- iter: 101/101
--
INFO:tensorflow:C:\Users\Dell\Desktop\GreatLearning PGP AIML\Capstone Project\Milestone 2\Apoorv Working\model_eda.tflearn is not in all_model_checkpo
int_paths. Manually adding it.
```

```
# restore all of our data structures
import pickle
data_eda = pickle.load( open( "eda_training_data", "rb" ) )
words_eda = data_eda['words']
classes_eda = data_eda['classes']
X_train_eda = data_eda['X_train']
y_train_eda = data_eda['y_train']
```

```
# import our chat-bot intents file
import json
with open('GLSafexInfo_eda.json') as json_data:
    intents_eda = json.load(json_data)


# load our saved model
model_eda.load('./model_eda.tflearn')
```

```
INFO:tensorflow:Restoring parameters from C:\Users\Dell\Desktop\GreatLearning PGP AIML\Capstone Project\Milestone 2\Apoorv Working\model_eda.tflearn
```

```
# functions to classify the sentences and for respective responses
# create a data structure to hold user context

context = {}

ERROR_THRESHOLD = 0.25
def classify_eda(sentence):
    # generate probabilities from the model
    results = model_eda.predict([bow(sentence, words_eda)])[0]
    # filter out predictions below a threshold
    results = [[i,r] for i,r in enumerate(results) if r>ERROR_THRESHOLD]
    # sort by strength of probability
    results.sort(key=lambda x: x[1], reverse=True)
    return_list = []
    for r in results:
        return_list.append((classes_eda[r[0]], r[1]))
    # return tuple of intent and probability
    return return_list


def response_eda(sentence, show_details=False):

    results = classify_eda(sentence)

    # if we have a classification then find the matching intent tag
    if results:
        # loop as long as there are matches to process
        while results:
            for i in intents_eda['intents']:

                # find a tag matching the first result
                if i['tag'] == results[0][0]:

                    #return print(random.choice(i['responses']))
                    response_result = random.choice(i['responses'])
                    show_eda(response_result)


            results.pop(0)
```

Chatbot: Classification Model

```
dataset_clf = dataset.copy()
```

```
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer,PorterStemmer
from nltk.tokenize import word_tokenize
#from wordcloud import WordCloud, STOPWORDS
import nltk
nltk.download('punkt')
nltk.download('wordnet')
import string
nltk.download('stopwords')
stop=set(stopwords.words('english'))

def preprocess_text(text):
      corpus=[]
      #stem=PorterStemmer()
      lem=WordNetLemmatizer()
      for news in text:
          words=[w for w in word_tokenize(news) if (w not in stop)]

          words=[lem.lemmatize(w) for w in words if len(w)>2]
          words = [''.join(c for c in s if c not in string.punctuation) for s in words if s]
          words = [word.lower() for word in words]
          words = [word for word in words if word.isalpha()]
          corpus.append(words)

      return corpus

dataset_clf['processed_text']= preprocess_text(dataset_clf['Description'])

desc_processed = []
for i in range(len(dataset_clf['processed_text'])):
   desc_processed.append(' '.join(wrd for wrd in dataset_clf.iloc[:,14][i]))

dataset_clf['description_processed'] = desc_processed
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Dell\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Dell\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Dell\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Modelling and functions for Accident Level

```
#Count vectorization
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer

X = dataset_clf['description_processed']
y = dataset_clf['Accident Level']

vectorizer = CountVectorizer(binary=True, ngram_range=(1, 2))
Xc = vectorizer.fit_transform(X).toarray()
Xc_train, Xc_test, yc_train, yc_test = train_test_split(Xc, y, test_size=0.15, random_state=42)
```

```
#SVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.svm import LinearSVC

svc = LinearSVC( max_iter=2500)
svc.fit(Xc_train, yc_train)
yc_pred_SVC = svc.predict(Xc_test)
```

```
acc_svc = accuracy_score(yc_test, yc_pred_SVC)
acc_svc_tr = svc.score(Xc_train, yc_train)
print("Train accuracy of the SVC model : {:.2f}".format(acc_svc_tr*100))
print("Test accuracy of the SVC model : {:.2f}".format(acc_svc*100))
```

```
Train accuracy of the SVC model : 99.44
Test accuracy of the SVC model : 79.37
```

```
print('Classification report:',classification_report(yc_test, yc_pred_SVC))
```

```
Classification report:               precision    recall  f1-score   support

           I       0.80      1.00      0.89        48
          II       0.00      0.00      0.00         5
         III       1.00      0.25      0.40         4
          IV       1.00      0.20      0.33         5
           V       0.00      0.00      0.00         1

    accuracy                           0.79        63
   macro avg       0.56      0.29      0.32        63
weighted avg       0.75      0.79      0.73        63
```

```
import pickle
filename = 'svc_model.sav'
pickle.dump(svc, open(filename, 'wb'))
```

```
loaded_model_acclevel = pickle.load(open('./svc_model.sav', 'rb'))
```

```
import pickle
filename = 'svc_model.sav'
pickle.dump(svc, open(filename, 'wb'))
```

```
loaded_model_acclevel = pickle.load(open('./svc_model.sav', 'rb'))
```

```
def response_acclevel(sentence):

    df_user_input=pd.DataFrame()
    df=pd.DataFrame()
    user_input = {'Description': sentence}
    df_user_input = df_user_input.append(user_input, ignore_index = True)
    df_user_input['processed_text']= preprocess_text(df_user_input['Description'])
    df['processed_text']=df_user_input['processed_text']
    df.reset_index(inplace=True)
    desc_processed = []
    for i in range(len(df['processed_text'])):
        desc_processed.append(' '.join(wrd for wrd in df.iloc[:,1][i]))
    test = vectorizer.transform(desc_processed)
    result = loaded_model_acclevel.predict(test)
    return result
```

Modelling and functions for Potential Accident Level

```
#Count vectorization
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer

X = dataset_clf['description_processed']
y = dataset_clf['Potential Accident Level']

vectorizer = CountVectorizer(binary=True, ngram_range=(1, 2))
Xc = vectorizer.fit_transform(X).toarray()
Xc_train, Xc_test, yc_train, yc_test = train_test_split(Xc, y, test_size=0.15, random_state=42)
```

```
#SVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.svm import LinearSVC

svc_potacclevel = LinearSVC( max_iter=2500)
svc_potacclevel.fit(Xc_train, yc_train)
yc_pred_SVC_potacclevel = svc_potacclevel.predict(Xc_test)
```

```
acc_svc_potacclevel = accuracy_score(yc_test, yc_pred_SVC_potacclevel)
acc_svc_tr_potacclevel = svc.score(Xc_train, yc_train)
print("Train accuracy of the SVC model : {:.2f}".format(acc_svc_tr_potacclevel*100))
print("Test accuracy of the SVC model : {:.2f}".format(acc_svc_potacclevel*100))
```

```
Train accuracy of the SVC model : 19.44
Test accuracy of the SVC model : 46.03
```

```
print('Classification report:',classification_report(yc_test, yc_pred_SVC_potacclevel))
```

```
Classification report:                precision    recall  f1-score   support

           I       1.00      0.14      0.25         7
          II       0.43      0.55      0.48        11
         III       0.35      0.32      0.33        19
          IV       0.52      0.70      0.59        23
           V       0.00      0.00      0.00         3

    accuracy                           0.46        63
   macro avg       0.46      0.34      0.33        63
weighted avg       0.48      0.46      0.43        63
```

```
import pickle
filename = 'svc_potacclevel_model.sav'
pickle.dump(svc_potacclevel, open(filename, 'wb'))
```

```
loaded_model_potacclevel = pickle.load(open('./svc_potacclevel_model.sav', 'rb'))
```

```
def response_potacclevel(sentence):

    df_user_input=pd.DataFrame()
    df=pd.DataFrame()
    user_input = {'Description': sentence}
    df_user_input = df_user_input.append(user_input, ignore_index = True)
    df_user_input['processed_text']= preprocess_text(df_user_input['Description'])
    df['processed_text']=df_user_input['processed_text']
    df.reset_index(inplace=True)

    desc_processed = []
    for i in range(len(df['processed_text'])):
        desc_processed.append(' '.join(wrd for wrd in df.iloc[:,1][i]))
    test = vectorizer.transform(desc_processed)
    result = loaded_model_potacclevel.predict(test)
    return result
```

Chatbot with Conditions
Chatbot function for Question and Answers

```
def chatbot_qna():
    flag=True
    print('\n')
    print("BOT: I am your Virtual assistant. I will try to answer your questions on Accidents Data! \
    \n \t If you want to exit any time, just type Bye!")
    while(flag==True):
        print('\n')
        user_response = input("You: ")
        user_response=user_response.lower()
        if(user_response!='bye') and (user_response!='exit') and (user_response!='quit'):
            if(user_response=='thanks' or user_response=='thank you' ):
                flag=False
                print('\n')
                print("BOT: You are welcome..")
                print("BOT: Exited from Question and Answer Module")

            else:
                print('\n')
                response(user_response)
                print('\n')
        else:
            flag=False
            print('\n')
            print("BOT: Exited from Question and Answer Module")
            #print("BOT: Goodbye! Take care  ")
            print('\n')
            print('-----------------------------------------------------')
            print('\n')
```

Chatbot function for Accident Level Classification

```
def chatbot_acclevel():
    flag=True
    print('\n')
    print("BOT: I am your Virtual assistant. I will try to Predict Accident Level for your description \
    \n \t If you want to exit any time, just type Bye!")
    while(flag==True):
        print('\n')
        user_response = input("You: ")
        user_response=user_response.lower()
        if(user_response!='bye') and (user_response!='exit') and (user_response!='quit'):
            if(user_response=='thanks' or user_response=='thank you' ):
                flag=False
                print('\n')
                print("BOT: You are welcome..")
                print("BOT: Exited from Accident Level Predictor")

            else:
                print('\n')
                response = response_acclevel(user_response)

                if response:
                    print ("Bot: Accident Level is --> ",response[0])
                print('\n')
        else:
            flag=False
            print('\n')
            print("BOT: Exited from Accident Level Predictor Module")
            #print("BOT: Goodbye! Take care  ")
            print('\n')
            print('-----------------------------------------------------')
            print('\n')
```

Chatbot function for Potential Accident Level Classification

```
def chatbot_potacclevel():
    flag=True
    print('\n')
    print("BOT: I am your Virtual assistant. I will try to predict Potential Accident Level for your description \
    \n \t If you want to exit any time, just type Bye!")
    while(flag==True):
        print('\n')
        user_response = input("You: ")
        user_response=user_response.lower()
        if(user_response!='bye') and (user_response!='exit') and (user_response!='quit'):
            if(user_response=='thanks' or user_response=='thank you' ):
                flag=False
                print('\n')
                print("BOT: You are welcome..")
                print("BOT: Exited from Potential Accident Level Predictor")

            else:
                print('\n')
                response = response_potacclevel(user_response)

                if response:
                    print ("Bot: Potential Accident Level is --> ",response[0])
                print('\n')
        else:
            flag=False
            print('\n')
            print("BOT: Exited from Potential Accident Level Predictor Module")
            #print("BOT: Goodbye! Take care  ")
            print('\n')
            print('-----------------------------------------------------')
            print('\n')
```

Chatbot function for showing Historical Data Insights

```
def chatbot_eda():
    flag=True
    print('\n')
    print("BOT: I am your Virtual assistant. I will help you with Insights of Accident Historical Data! \
          \n \t If you want to exit any time, just type Bye!")
    while(flag==True):

        print('\n')
        user_response = input("You: ")
        user_response=user_response.lower()
        if(user_response!='bye') and (user_response!='exit') and (user_response!='quit'):
            if(user_response=='thanks' or user_response=='thank you' ):
                flag=False
                print('\n')
                print("BOT: You are welcome..")
                print("BOT: Exited from Data Insights Module")

            else:
                    print('\n')
                    response_eda(user_response)
                    print('\n')
        else:
            flag=False
            print('\n')
            #print("BOT: Goodbye! Take care  ")
            print("BOT: Exited from Data Insights Module")
            print('\n')
            print('-----------------------------------------------------')
            print('\n')
```

Chatbot: Home Page

```
while(mainflag==True):
    if flag == False:
        print('')
        print("BOT: Please Select appropriate option: \
          \n \t  \
          \n \t 1 --> For Question and Answers \
          \n \t 2 --> For finding the Accident level for your problem \
          \n \t 3 --> For finding the Potential Accident level for your problem \
          \n \t 4 --> For Historical Data Insights \
          \n \t  \
          \n \t if you want to exit any time, just type Bye!")

    print('\n')
    user_response = input("You: ")
    user_response=user_response.lower()
    if(user_response!='bye') and (user_response!='exit') and (user_response!='quit'):
        if(user_response=='thanks' or user_response=='thank you' ):
            mainflag=False
            print('\n')
            print("BOT: You are welcome..")

        elif (user_response=='1'):
                chatbot_qna()
        elif (user_response=='2'):
                chatbot_acclevel()
        elif (user_response=='3'):
                chatbot_potacclevel()
        elif (user_response=='4'):
                chatbot_eda()

    else:
        mainflag=False
        print('\n')
        print("BOT: Goodbye! Take care  ")
        print('\n')
        print('===============================================================')
        print('\n')
```

```
BOT: Please Select appropriate option:

        1 --> For Question and Answers
        2 --> For finding the Accident level for your problem
        3 --> For finding the Potential Accident level for your problem
        4 --> For Historical Data Insights

        if you want to exit any time, just type Bye!


You: 1


BOT: I am your Virtual assistant. I will try to answer your questions on Accidents Data!
        If you want to exit any time, just type Bye!


You: tell me details on issue realted to zinc


BOT:  Based on the historical facts it has been observed  that Employees in the Metal Industry have met with accidents related to volumetric balloon,
cyclone duct, conveyor, Zinc which have result in small burns on the body parts including face as well. So it is always recommended to take proper pre
cautions and use face and body covers while working on such areas
```

You: quit

BOT: Exited from Question and Answer Module

BOT: Please Select appropriate option:

1 --> For Question and Answers
2 --> For finding the Accident level for your problem
3 --> For finding the Potential Accident level for your problem
4 --> For Historical Data Insights

if you want to exit any time, just type Bye!

You: 2

BOT: I am your Virtual assistant. I will try to Predict Accident Level for your description
If you want to exit any time, just type Bye!

You: When the plant operator was semi-kneeling when lifting the lid or gate (15 kg) of the distributor box of the secondary mill No. 4 and No. 5, his right knee slips due to the presence of debris spilled on the platform or floor (Grating) - which gave him an extra effort in his left leg, generating a muscle contracture.

Bot: Accident Level is --> I

You: bye

BOT: Exited from Accident Level Predictor Module

BOT: Please Select appropriate option:

1 --> For Question and Answers
2 --> For finding the Accident level for your problem
3 --> For finding the Potential Accident level for your problem
4 --> For Historical Data Insights

if you want to exit any time, just type Bye!

You: 3

BOT: I am your Virtual assistant. I will try to predict Potential Accident Level for your description
If you want to exit any time, just type Bye!

You: When the plant operator was semi-kneeling when lifting the lid or gate (15 kg) of the distributor box of the secondary mill No. 4 and No. 5, his right knee slips due to the presence of debris spilled on the platform or floor (Grating) - which gave him an extra effort in his left leg, generating a muscle contracture.

Bot: Potential Accident Level is --> III

You: bye

BOT: Exited from Potential Accident Level Predictor Module

BOT: Please Select appropriate option:

1 --> For Question and Answers
2 --> For finding the Accident level for your problem
3 --> For finding the Potential Accident level for your problem
4 --> For Historical Data Insights

if you want to exit any time, just type Bye!

You: give me details about country and industry

BOT: Please Select appropriate option:

1 --> For Question and Answers
2 --> For finding the Accident level for your problem
3 --> For finding the Potential Accident level for your problem
4 --> For Historical Data Insights

if you want to exit any time, just type Bye!

You: 4

BOT: I am your Virtual assistant. I will help you with Insights of Accident Historical Data!
         If you want to exit any time, just type Bye!

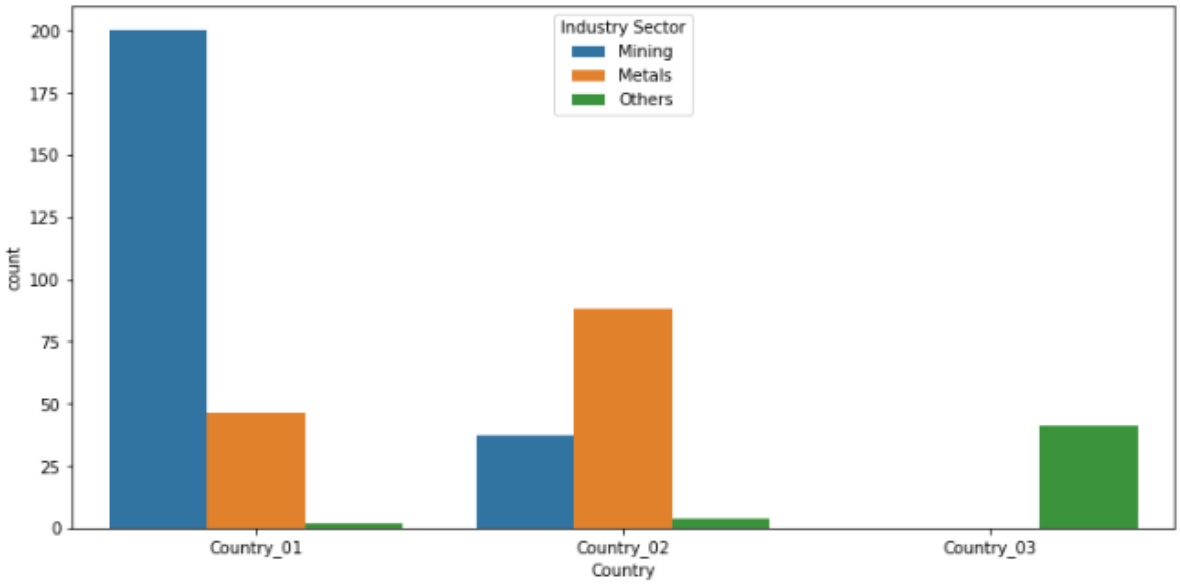
You: You: give me details about country and industry

==========
Data Analysis of features: Country and Industry Sector
==========

Cross table Analysis of features: Country and Industry Sector
---

Industry Sector	Metals	Mining	Others
Country			
Country_01	46	200	2
Country_02	88	37	4
Country_03	0	0	41

Count plot Analysis of features: Country and Industry Sector



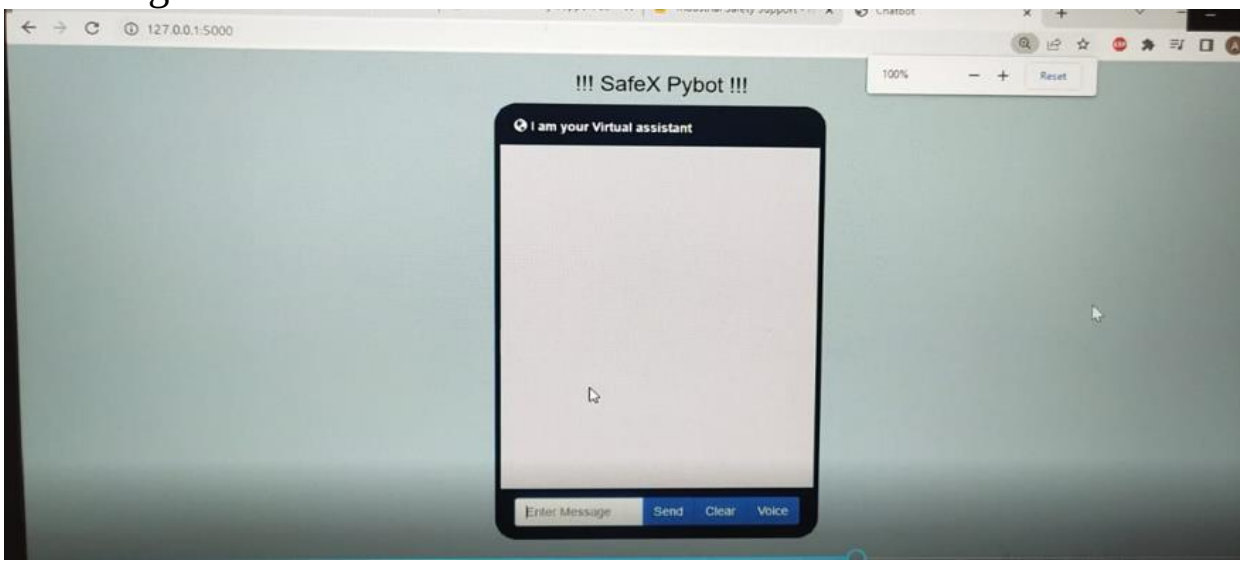
You: exit

BOT: Exited from Data Insights Module

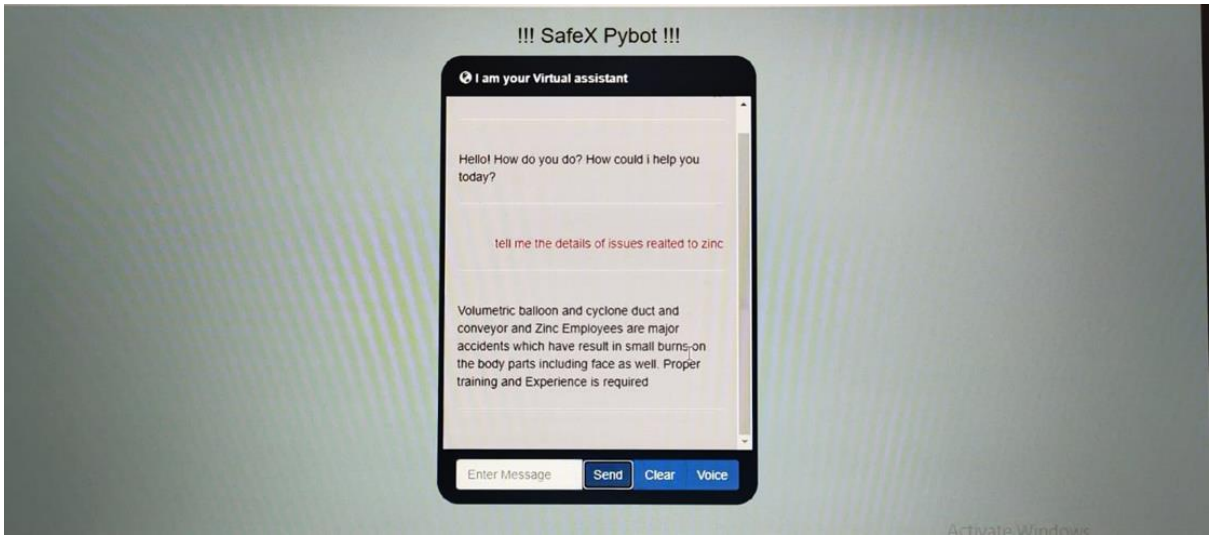
```
BOT: Please Select appropriate option:

         1 --> For Question and Answers
         2 --> For finding the Accident level for your problem
         3 --> For finding the Potential Accident level for your problem
         4 --> For Historical Data Insights

         if you want to exit any time, just type Bye!


You: exit


BOT: Goodbye! Take care


======================================================================
```

UI Screenshots:

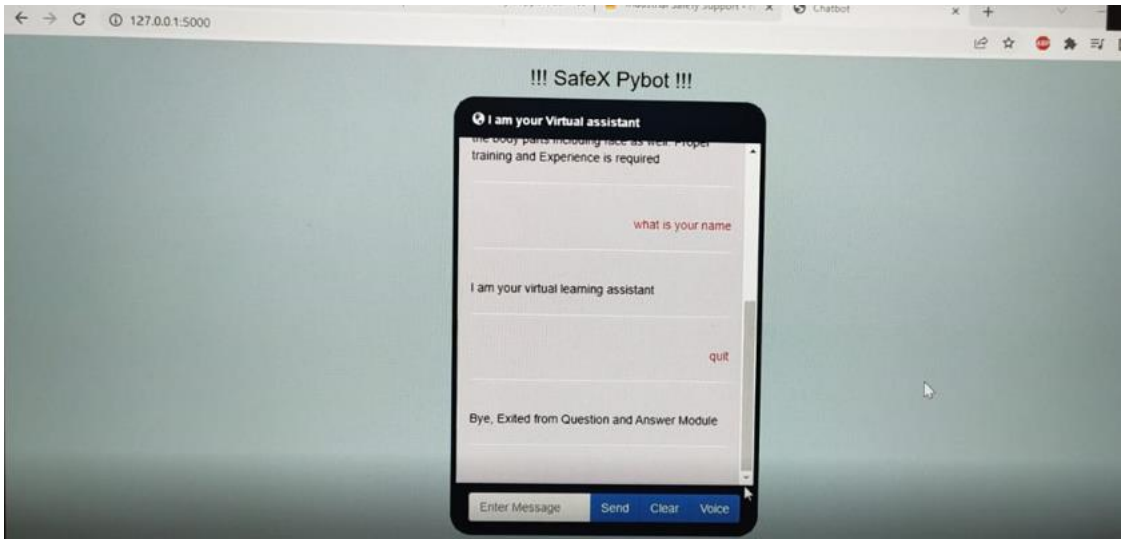
UI Design



Question & Answer

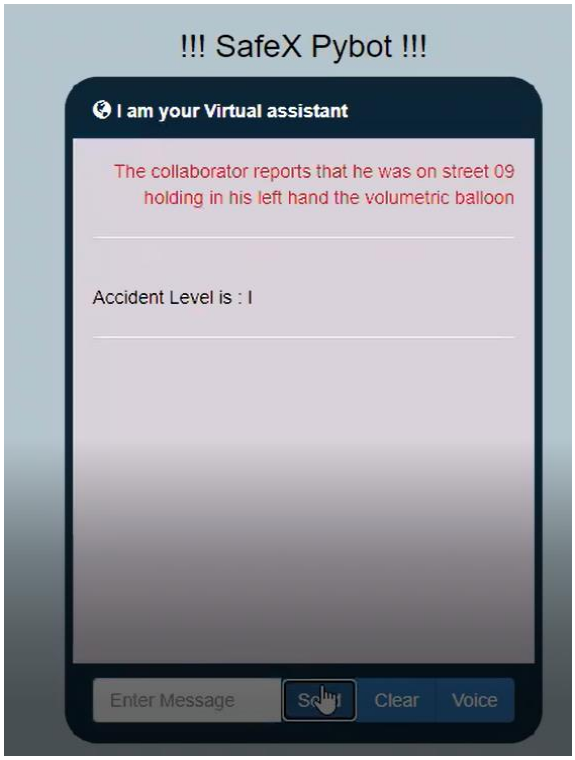


Our Safex Chatbot also recognises the typoerrors and provide the relevant answers to questions asked by the users.



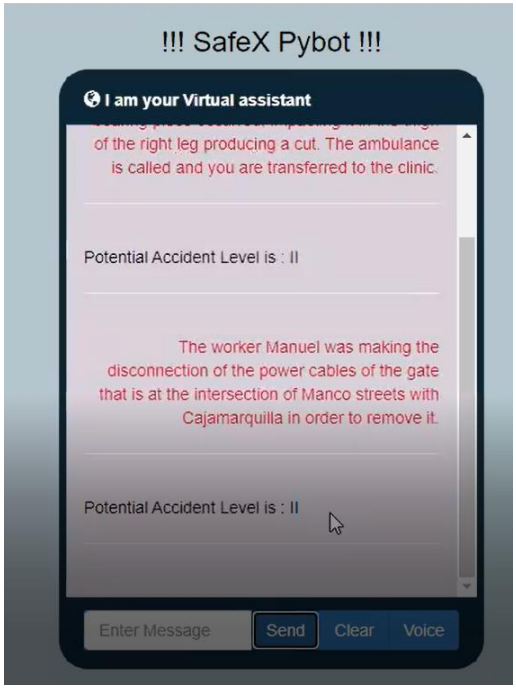
Accident Level

The below UI is used for determining Accident Level according to the description provided by the user to the Bot



Potential Accident Level

The below UI is used for determining Potential Accident Level according to the description provided by the user to the Bot



Implications:

6.a.1) Before using Chatbot, users were not able to get the required information about the potential accidents, precautions and measures to such accidents occurred in industries

6.a.2) Users were not able to identify the criticality of accidents occurring in industries

6.a.3) By using chatbot, users can able to get the proper information such that they can take necessary precautions while working in industries.

6.a.4) The chatbot created by us is using the SVM model to determine the accident level, and potential accident level according to the accident details provided by the user to the bot. Hence, this would be useful to prevent the same type of accidents that occurred previously.

6.a.5) The Question and Answer module of our chatbot is providing the users about the accidents that took place previously in different industrial sectors to different types of employees such as third party, employee, and others. Hence, this would be helpful for the people/ users to understand the risks and causes of the accidents. It also provides the precautionary measures to be taken to prevent accidents.

6.a.6) Using chatbots in industries aids in lowering support cost, increasing conversation rates, boosting customer loyalty, accelerating sales.

6.a.7) Chatbots means faster replies, better customer service, easier support and sales flow.

Limitations and Closing Reflections:

1) The provided dataset is of historical dataset, so the efficiency and usefulness can be increased if we are provided with the latest dataset. Some of the Industrial sectors and its criticality might differ over the time. Hence working on the latest data would be better to understand the risks and accidents caused by the industries.
2) Chat application is not deployed on cloud. Hence, as a part of future enhancement we can deploy the application on cloud.
3) Chat application is not supporting multi-languages. As a part of future enhancement, we can able to design and deploy which supports multi- languages.
4) Feedback support feature can be added as future enhancement.
5) Can do more study on domain and frame different types of questions to provide more detailed analysis of the causes and risks of the accidents occurred in different sectors of industries.