CHATBOT INTERFACE USING NLP

POST GRADUATE PROGRAM IN

ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

TEXAS McCombs The University of Texas at Austin McCombs School of Business | GREAT LAKES EXECUTIVE LEARNING | Great Learning POWER AHEAD

May 2023 **Capstone Project for PGP AIML**

Project Team JUNE B Group:3 (NLP)

ALAHARI SUNIL

N S PRASAD

ISHANT KUNDRA

KOORAPATI VAMSHI

GOPI DESETTY

JEREMY KUJUR

NLP BASED CHATBOT INTERFACE FOR METALS & MINING INDUSTRY

Design Advisor
Dr. Aamit Ajit Lakhani

Abstract

The following document contains the report for the capstone project in progress. This report is used to explain the steps come across and insights at place of need.

The purposes of an abstract are:

1. To define the Problem Statement.

2. To give a clear indication of the objective, scope and key results of the project so that readers may understand the overview.

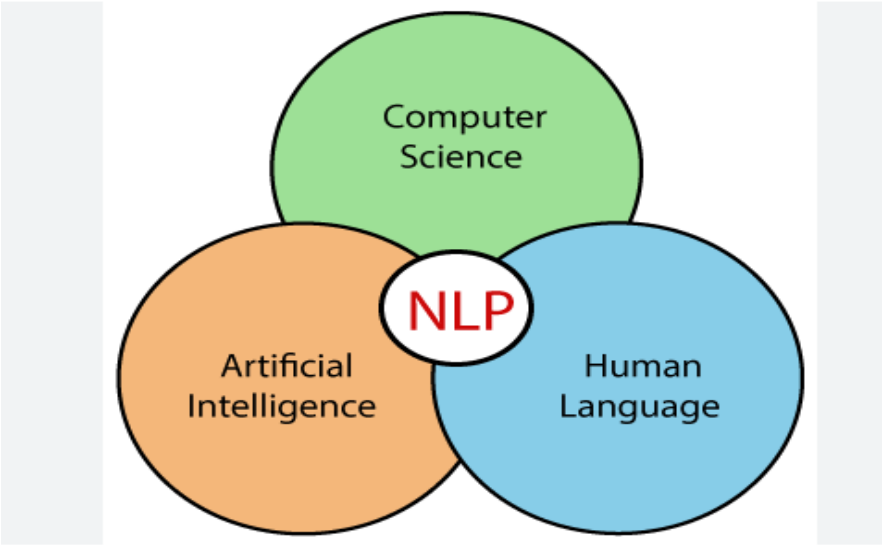
3. To provide the details of Libraries / module or code used, outline the approach taken and also to present the tasks remaining hereafter.

Table of Contents

Introduction to NLP

Natural language processing (NLP) refers to the branch of Artificial Intelligence concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.

NLP combines computational linguistics rule-based modeling of human language with statistical, machine learning and deep learning models. Together, these technologies enable computers to process human language in the form text or voice data and to understand its full meaning, complete with the speaker or writer's intent and sentiment.



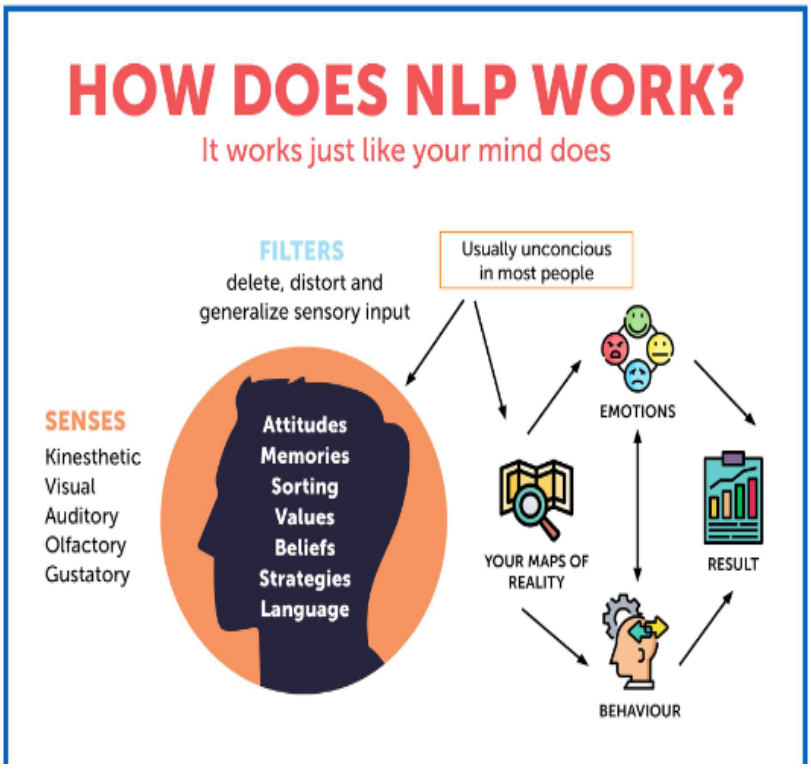
How does NLP work:

NLP drives computer programs that translate text from one language to another, respond to spoken commands and summarize large volumes of text rapidly even in real time.

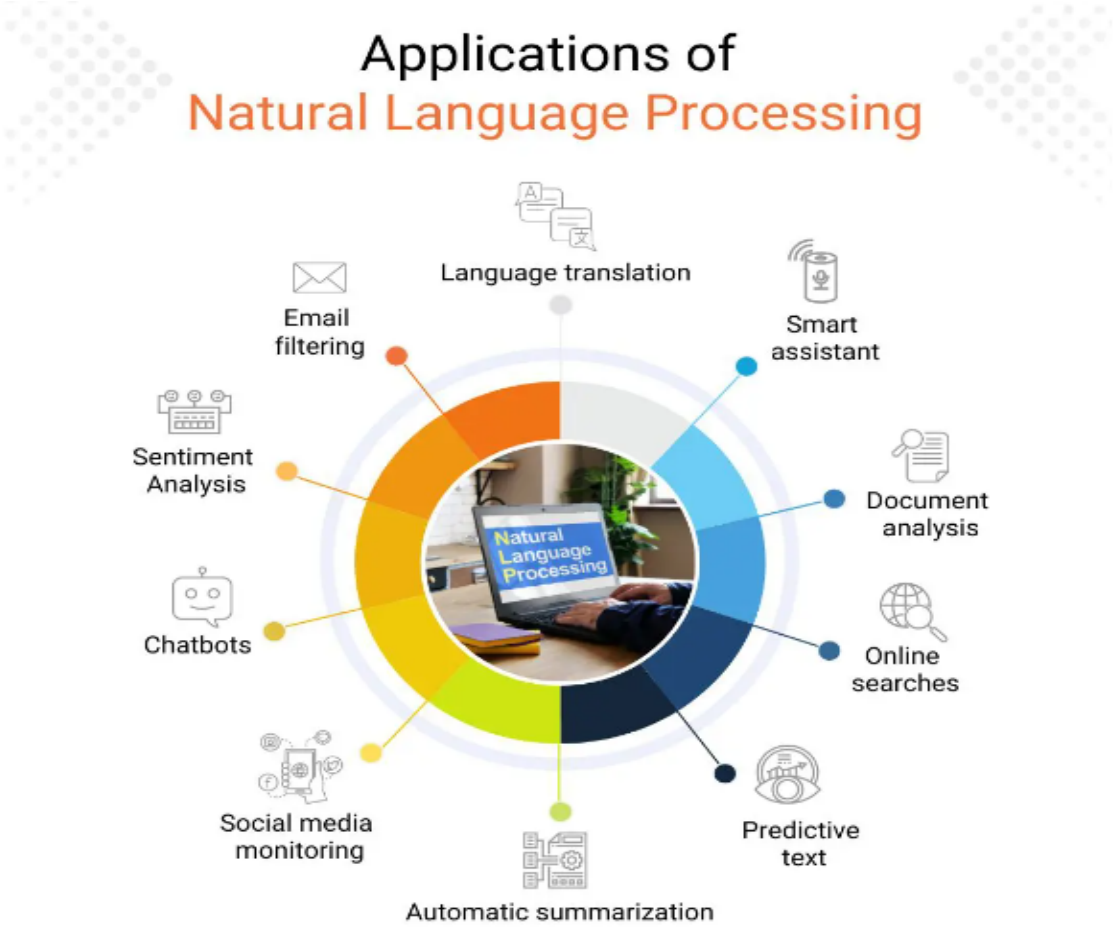
NLP follows different methodologies to break down human text and voice data in ways that help the computer make sense of what its ingesting. Some of them are as follows:

- **Speech Recognition** – or Speech to text, which reliably converts voice data into text data. It's very challenging since there are different ways people talk – quickly, slurring words together, with variation in tones, accents, and sometime even incorrect grammar.

- **Part of speech tagging** – or grammatical tagging, is the process of determining the part of speech of a particular word or piece of text based on its use and context. Part of speech identifies sentence with verbs.
- **Word sense disambiguation** is the selection of the meaning of a word with multiple meanings through a process of semantic analysis that determine the word that makes the most sense in given context.
- **Named entity recognition** identifies words or phrases as useful entities.
- **Co-reference resolution** is the task of identifying if and when two words refer to the same entity. It determines the person or object to which a certain pronoun refers, it involves metaphor or an idiom in the text.
- **Sentiment analysis** attempts to extracts subjective qualities – attitudes, emotions, sarcasm, confusion, suspicion from text.
- **Natural language generation** is sometimes described as the opposite of speech recognition or speech to text.



Applications of NLP:



NLP Use Cases:

- **Spam Detection**: NLP's text classification capabilities to scan emails for language that often indicates spam or phishing. These indicators can include overuse of financial terms, characteristics bad grammar, threatening language, inappropriate urgency, misspelled company names and more.

- **Machine translation**: Google translate is an example of widely available NLP technology at work. Truly useful machine translation involves more than replacing words in one language with words of another. Effective translation has to capture accurately the meaning and tone of the input language and translate it to text with the same meaning and desired impact in the output language.

- **Virtual agents and chatbots:** Virtual agents such as Apple's Siri and Amazon's Alexa use speech recognition to recognize patterns in voice commands and natural language generation to respond with appropriate action or helpful

comments. Chatbots perform the same magic in response to typed text entries. The best of these also learn to recognize contextual clues about human requests and use them to provide even better responses or options over time. Next enhancement for these applications is question answering, the ability to respond to our questions, anticipated or not with relevant and helpful answers in their own words.

- **Social Media sentiment analysis:** NLP has become an essential business tool for uncovering hidden data insights from social media channels. Sentiment analysis can analyze language used in social media posts, responses, reviews and more to extract attitudes and emotions in response to products, promotions, events information companies can use in product designs, advertising campaigns.

- **Text summarization:** Text summarization uses NLP techniques to digest huge volumes of digital text and create summaries and synopses for indexes, research databases or busy readers who don't have time to read full text. The best text summarization applications use semantic reasoning and natural language generation to add useful context and conclusions to summaries.

RNN (Recurrent Neural Networks):

What is RNN?

Recurrent neural networks (RNNs) are a type of artificial neural network that is well-suited for processing sequential data such as text, audio, or video. RNNs have a recurrent connection between the hidden neurons in adjacent layers, which allows them to retain information about the previous input while processing the current input.

This makes RNNs particularly useful for tasks such as language translation or speech recognition, where understanding the context is essential. A long short term memory neural network is designed to overcome the vanishing gradient problem, which can occur when training traditional RNNs on long sequences of data. LSTMs have been shown to be effective for a variety of tasks, including machine translation and image captioning.

Standard Feedforward Neural Networks are only suitable for independent data points. To include the dependencies between these data points, we must change the neural network if the data are organized in a sequence where each data point depends on the one before. A unique kind of deep learning network called RNN full form Recurrent Neural Network is designed to cope with time series data or data that contains sequences.

The idea of "memory" in RNNs enables them to store the states or details of earlier inputs to produce the subsequent output in the sequence. Different types of RNNs are based on the number of inputs and outputs. They are:

- One-to-one
- One-to-many
- Many-to-one
- Many-to-many

Advantages and Disadvantages of Recurrent Neural Networks

The following are a few advantages that Recurrent Neural Networks offer:

- The processing of sequential data.
- Ability to remember and preserve primary outcomes.
- When calculating new results, consider the most recent and previous results.
- The model size does not change as the input size does. Over time, it distributes weights to other components.

Some of the disadvantages of RNN are:

- Problems with vanishing and gradient explosion.
- The challenge of training an RNN is quite challenging.
- Using Tanh or Relu as an activation feature prevents it from processing long sequences.

Different RNN Architectures

Short-term memory is a problem for recurrent neural networks. They will need more time to transfer information from earlier time steps to later ones if a sequence is lengthy. RNNs may exclude crucial details from the start if you're trying to process a paragraph of text to make predictions.

GRUs and LSTMs were developed to answer short-term memory problems. They possess inbuilt components known as gates that can control the information flow. These gates can learn which data in a sequence should be kept or ignored. To create predictions, it can convey relevant data along the extensive chain of sequences by doing this.

Speech synthesis, speech recognition, and text production all use this architecture. You can even use them to create video captions.

Applications of RNN

- It helps resolve time-series issues like stock market forecasts.
- It assists in resolving problems with sentiment analysis and text mining.
- The development of NLP technology, machine translation, speech recognition, language modeling, etc., uses RNNs.
- Besides other OCR uses, it aids in picture captioning, video tagging, text summarization, image identification, and facial recognition.

Conclusion

- The Recurrent Neural Networks ended with examples of RNN neural network applications and their use in processing sequential input.
- RNNs can solve time-series and data sequence problems more effectively than conventional feedforward algorithms.
- You have learned much about RNN in this article, including its types, uses, and architecture. You have also known about its applications.

LSTM (Long Short-Term Memory):

Long Short Term Memory networks are a type of recurrent neural network designed to model complex, sequential data. Unlike traditional RNNs, which are limited by the vanishing gradient problem, LSTMs can learn long-term dependencies by using a method known as gated recurrent units (GRUs). GRUs contain a "forget" gate, which allows them to selectively forget information from the previous timestep, and an "update" gate, which allows them to control how much information from the current timestep is passed on to the next time step.

This makes LSTMs well-suited for tasks such as machine translation, where it is important to be able to remember and interpret information from long sequences. In addition, LSTMs can be trained using a variety of different methods, including backpropagation, through time and reinforcement learning.

What is Long Short-Term Memory (LSTM)? This question has been asked lately with the release of several new smartphones and devices that include LSTM technology. While LSTM hasn't quite hit the mainstream yet, it's something you should be familiar with. In this post, we'll take a detailed look at what LSTM is and how it works. We'll also explore some benefits of using LSTM and discuss possible applications for this innovative technology.

What is Long Short-Term Memory (LSTM)?

Long short-term memory (LSTM) is the artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard RNNs, LSTM has "memory cells" that can remember information for long periods of time. It also has three gates that control the flow of information into and out of the memory cells: the input gate, the forget gate, and the output gate.

LSTM networks have been used on a variety of tasks, including speech recognition, language modelling, and machine translation. In recent years, they have also been used for more general sequence learning tasks such as activity recognition and music transcription.

Ideology Behind LSTMs and How Does It Work?

So, as we are now through with the basic question, “what is long short term memory” let us move on to the ideology behind Long short term memory networks. Humans can remember memories from the distant past, as well as recent events, and we can also easily recall sequences of events. LSTMs are designed to mimic this ability, and they have been shown to be successful in a variety of tasks, such as machine translation, image captioning, and even handwriting recognition. Does this bring an easy undertaking to “What is the long-term memory”?

LSTM vs. RNN

Long Short Term Memory neural networks are types of recurrent neural networks (RNN) that are well-suited for modeling sequence data. In contrast to RNNs, which tend to struggle with long-term dependencies, LSTMs can remember information for extended periods of time. This makes them ideal for tasks such as language modeling, where it is important to be able to capture the context of a sentence to predict the next word. LSTMs are also commonly used in machine translation and speech recognition applications.

There are a number of advantages that LSTMs have over traditional RNNs.

- First, they are much better at handling long-term dependencies. This is due to their ability to remember information for extended periods of time.
- Second, LSTMs are much less susceptible to the vanishing gradient problem. This is because they use a different kind of activation function, known as an LSTM cell, which helps to preserve information over long sequences.
- Finally, LSTMs are very efficient at modeling complex sequential data. This is because they can learn high-level representations that capture the structure of the data.

Despite these advantages, LSTMs do have some drawbacks.

- First, they are more complicated than traditional RNNs and require more training data in order to learn effectively.
- Second, they are not well-suited for online learning tasks, such as prediction or classification tasks where the input data is not a sequence. Third, LSTMs can be slow to train on large datasets. This is due to the fact that they must learn the parameters of the LSTM cells, which can be computationally intensive.
- Finally, LSTMs may not be appropriate for all types of data. For example, they may not work well with highly nonlinear data or data with a lot of noise.

What are Bidirectional LSTMs?

Bidirectional LSTMs are a type of recurrent neural network that is often used for natural language processing tasks. Unlike traditional LSTMs, which read input sequentially from left to right, bidirectional LSTMs are able to read input in both directions, allowing them to capture context from both the past and the future.

This makes them well-suited for tasks such as named entity recognition, where it is important to be able to identify entities based on their surrounding context. Bidirectional LSTMs are also sometimes used for machine translation, where they can help to improve the accuracy of the translation by taking into account words that appear later in the sentence.

For a comprehensive look into the world of LSTM, it is advisable to get enrolled in a MongoDB Certification course and learn everything you need to know about these neural networks.

LSTM Applications

LSTM has been used to achieve state-of-the-art results in a wide range of tasks such as language modelling, machine translation, image captioning, and more.

1. Language Modeling

One of the most common applications of LSTM is language modelling. Language modelling is the task of assigning a probability to a sequence of words. In order to do this, LSTM must learn the statistical properties of language so that it can predict the next word in a sentence.

2. Machine Translation

Another common application of LSTM is a machine translation. Machine translation is the process of translating one natural language into another. LSTM has been shown to be effective for this task because it can learn the long-term dependencies that are required for accurate translations.

3. Handwriting Recognition

Handwriting recognition is the task of automatically recognizing handwritten text from images or scanned documents. This is a difficult task because handwritten text can vary greatly in terms of style and quality, and there are often multiple ways to write the same word. However, because LSTMs can remember long-term dependencies between strokes, they have been shown to be effective for handwriting recognition tasks.

4. Image Captioning

LSTM can also be used for image captioning. Image captioning is the task of generating a textual description of an image. This is a difficult task because it requires understanding both the visual content of an image and the linguistic rules for describing images. However, LSTM works well at image captioning by learning how to interpret images and generate appropriate descriptions.

5. Image Generation using Attention Models

Attention models are a type of neural network that can learn to focus on relevant parts of an input when generating an output. This is especially useful for tasks like image generation, where the model needs to focus on different parts of the image at different times. LSTMs can be used together with attention models to generate images from textual descriptions.

6. Question Answering

LSTMs can also be used for question-answering tasks. Given a question and a set of documents, an LSTM can learn to select passages from the documents that are relevant to the question and use them to generate an answer. This task is known as reading comprehension and is an important testbed for artificial intelligence systems.

Recently, Google released the SQUAD dataset, which contains 100,000+ questions answered by crowd workers on a set of Wikipedia articles. A number of

different neural networks have been proposed for tackling this challenge, and many of them use LSTMs in some way or another.

7. Video-to-Text Conversion

Video-to-text conversion is the task of converting videos into transcripts or summaries in natural language text. This is a difficult task because it requires understanding both the audio and visual components of the video in order to generate accurate text descriptions. LSTMs have been used to develop successful video-to-text conversion systems.

8. Polymorphic Music Modeling

Polyphonic music presents a particular challenge for music generation systems because each note must be generated independently while still sounding harmonious with all the other notes being played simultaneously. One way to tackle this problem is to use an LSTM network trained on polyphonic music data. This approach has been shown to generate convincing polyphonic music samples that sound similar to human performances.

9. Speech Synthesis

Speech synthesis systems typically use some form of acoustic modeling in order to generate speech waveforms from text input. Recurrent neural networks are well suited for this task due to their ability to model sequential data such as speech signals effectively.

10. Protein Secondary Structure Prediction

Protein secondary structure prediction is another important application of machine learning in biology. Proteins are often described by their primary structure (the sequence of amino acids) and their secondary structure (the three-dimensional shape).

Secondary structure prediction can be viewed as a sequence labelling task, where each residue in the protein sequence is assigned one of three labels (helix, strand, or coil). Long Short Term Memory networks have been shown to be effective at protein secondary structure prediction, both when used alone and when used in combination with other methods such as support vector machines.

Limitations of LSTM

LSTMs are not perfect, however, and there are certain limitations to their abilities. Here, we'll explore some of those limitations and what they mean for the future of artificial intelligence.

1. Temporal Dependencies

One of the biggest limitations of LSTMs is their inability to handle temporal dependencies that are longer than a few steps. This was demonstrated in a paper published by Google Brain researchers in 2016. The researchers found that when they trained an LSTM on a dataset with long-term dependencies (e.g., 100 steps), the network struggled to learn the task and generalize to new examples.

This limitation arises because LSTMs use a forget gate to control what information is kept in the cell state and what is forgotten. However, this gate can only forget information that is a few steps back; anything further back is forgotten completely. As a result, LSTMs struggle to remember dependencies that are many steps removed from the current input.

There are two possible ways to address this limitation: either train a larger LSTM with more cells (which requires more data) or use a different type of neural network altogether. Researchers from DeepMind recently proposed a new type of recurrent neural network called the Neural Stack Machine, which they claim can learn temporal dependencies of arbitrary length.

However, it remains to be seen whether this model will be able to scale to large datasets and complex tasks like machine translation and automatic question answering.

2. Limited Context Window Size

Another limitation of LSTMs is their limited context window size. A context window is the set of inputs that the network uses to predict the next output; for instance, in a language model, an input might be a sequence of words while the output is the next word in the sentence. The size of the context window is determined by the number of recurrent units in the LSTM; typically, this number is between 2 and 4.

This means that an LSTM can only consider a limited number of inputs when making predictions; anything outside of the context window is ignored completely. This can be problematic for tasks like machine translation, where it's important to consider the entire input sentence (not just the last few words) in order to produce an accurate translation.

There are two possible ways to address this limitation as well: either train a larger LSTM with more cells (which requires more data) or use Attention-based models instead, which have been shown to be better at handling long input sequences. However, both of these methods come with their own trade-offs and challenges (e.g., Attention models usually require more training data). In case you feel these limitations are still in your way, then get in touch with the experts of KnowledgeHut's Database Courses and solve all your problems with their professional expertise.

Conclusion:

So far, we've learned that LSTM is a powerful memory tool that can be used to improve your studying and learning habits. We've looked at some of the ways you can use LSTM to its full potential in order to make the most of your memory. Finally, we've discussed how you can apply these same techniques to other areas of your life in order to achieve success. Are you ready to start using LSTM?

Introduction to Chatbot

Chatbot is a computer application that simulates and processes human conversation (either written or spoken) allowing humans to interact with digital devices as if they are communicating with a real person.

In a fast-moving world, it is essential to have Chatbot for every service across we deal with. So, invention of Chatbot is helping every individual to interact with the service provider seamlessly with mostly with FAQs.

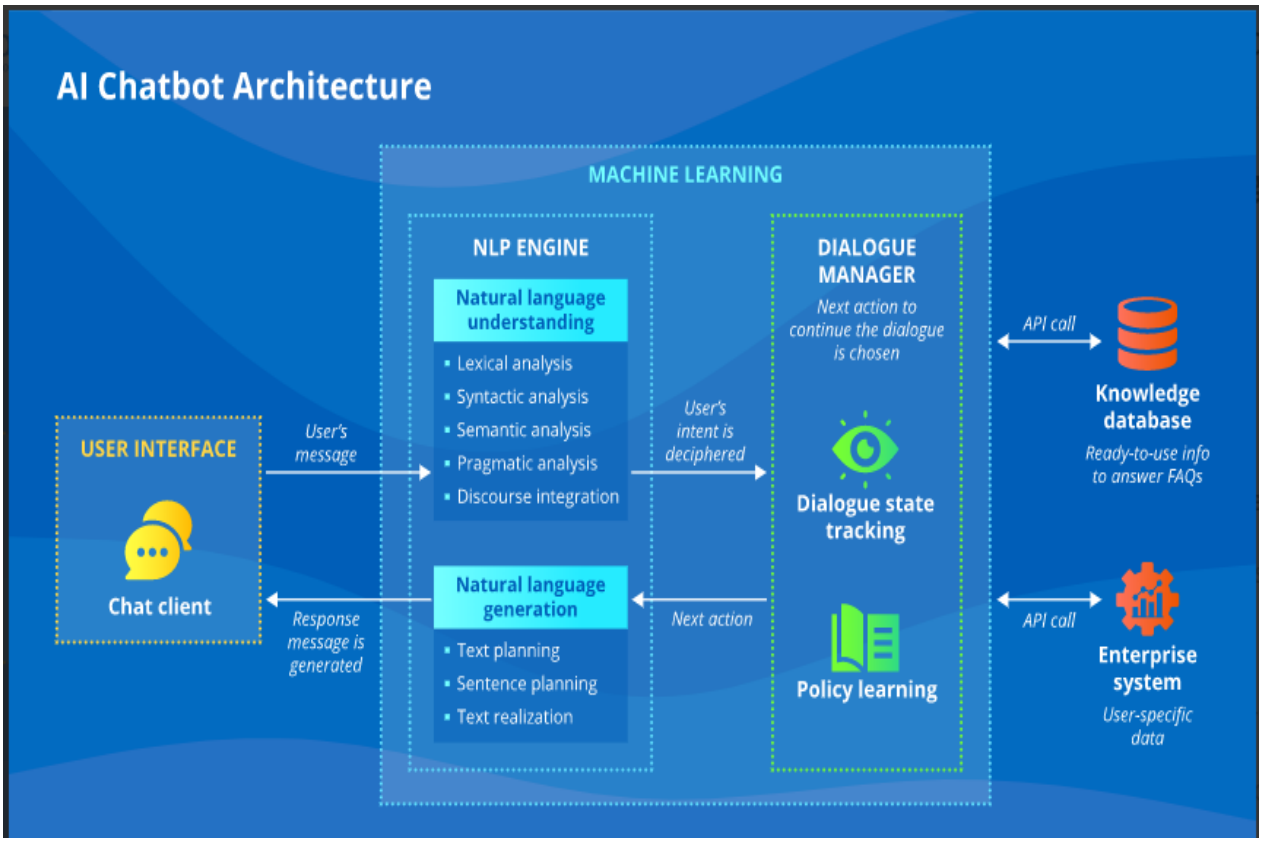
What is an AI Based Chatbot?

All the Chatbots use Artificial Intelligence (AI) and Natural Language Processing (NLP) to understand user questions and automate responses to them, simulating human conversation.

Chatbots are designed to respond by understanding user questions and request through Text or an Audio or Both. NLP helps in picking the right word or phrase or a sentence to prepare & process internally a most relevant answer for the question being asked.

Over time, Chatbots have integrated more rules and thanks to NLP, so users can experience them in a conversational way. In fact, the latest Chatbots are contextually aware and able to learn as they are exposed to more and more human language. In order to facilitate use of NLP, current day technologies rely on Machine Learning (ML) and Deep Learning (DL) which are essential elements of AI to develop an increasingly granular knowledge base of questions and responses that are based on user interaction. This evolution has increased their ability to predict user needs accurately and respond in a best way.

Typical Architecture of an AI based Chatbot:



Any Chatbot architecture should consist of the following:

- Chat window / session / front end application interface
- Deep learning model for NLP
- Corpus or Training data for training the NLP model
- Application Database for processing actions to be performed by the chatbot

Corpus or Training Data:

Corpus means the data that could be used to train the NLP model to understand the human language as text or speech and reply using same medium of interaction. So it is usually huge data with many human interactions which can be designed by using one of the following methods:

- Manual
- Accumulated data over time in an organized manner

Algorithm for text based Chatbot:

- Input the corpus
- Design NTLK responses and converse based chat utility as a function to interact with the user
- Run the chat utility function

Designing a Chatbot Window:

Chatbot to be designed with a function that enables the user to interact with bot using text. The function keeps the chat window alive unless it is asked to break or quit. The algorithm as follows:

- The text bot introduces itself to the user
- Chatbot asks the user to type in the chat window using the NTLK converse function

- Chatbot understands what the user has typed in the chat utility window using NTLK chat pairs and reflections function.

Evaluate or test the chatbot

Following are steps followed during NLP based Evaluation.

1. Data Pre-processing

I. Text case (Upper or Lower) handling:

To maintain common text input converting to either Upper or Lower case is essential. This will avoid misinterpretation of text / words if spelled under different cases.

II. Tokenization

Converts a sentence into single words.

III. Stemming

This process is to find similarities between words with the same root words. This will help us to reduce the bag of words by associating similar words with their corresponding root words.

IV. Generate BOW (Bag of Words)

Process of converting words into numbers by generating vector embeddings from the tokens generated above. This is given as input to the neural network model for understanding the written text. This is followed by Tagging & One hot Encoding of Tag.

V. Text Classification

Design a classifier model which can be trained on the corpus with respect to the target variable, i.e., tag the input from corpus. Following are the list of classifiers used:

- Multinomial Naive Bayes
- Support Vector Machines

- Neural Network Classifier

Project Objective

To Design a ML / CL based Chatbot utility which can help professionals to highlight the safety risk as per the incident description. For the Data analytics, extensive Database created from one of the biggest Metals & Mining Industry in Brazil which has various plants across World.

Description of Problem

During the day-to-day operations, since the employee deal with harsh environment, heavy machines, unsafe work conditions safety hazards are inevitable. This raises a paramount concern to Company Management who own such industry for safety & wellbeing of their employees.

So, it is imperative to know the reasons of safety issues to why such accidents take place affecting their employees with minor /major injuries.

Data Description

The Column of database contains following data:

- Data: timestamp or time / date information – Used to know the instance of accident
- Countries: which country the accident occurred -- Used for geographical analysis
- Local: the city where the manufacturing plant is located – Used for geographical analysis
- Industry Sector: which sector the plant belongs to – Used to understand the Plant where accident occurred
- Accident Level: from I to VI, it registers how severe was the accident – Used to categorize the severity of accident to understand the nature.
- Potential Accident Level: Depending on the accident level, the database also registers how severe the accident could have been – Used to understand other factors to predict the potential accident that could have happened.
- Genre: if the person is male or female – Used to categorize the gender involved in accident

- Employee or Third Party: if the injured person is an employee or third party – Used to understand the nature of job / category of people involved.
- Critical Risk: some description of the risk involved in the accident – Used to identify the risk
- Description: Detailed description of how the accident happened – Used to get clarity on the accident details for further detailed analysis

Project Task

Milestone 1:

Process:

Step 1: Import the data
Step 2: Data Cleansing
Step 3: Data Preprocessing (NLP Preprocessing techniques)
Step 4: Data Preparation – Cleansed data in.xlsx or .csv file
Step 5: Design train and text basic machine learning classifiers
Submission: Interim report

Milestone 2:

Process:

Step 1: Design, train and test Neural networks classifiers
Step 2: Design, train and test RNN or LSTM classifiers
Step 3: Choose the best performing classifier and pickle it
Step 4: Final Report
Submission: Final report, Jupyter Notebook with all the steps in Milestone-1 and Milestone-2

Milestone 3: [Optional]

Process:

Step 1: Design a clickable UI based chatbot interface
Submission: Final report, Jupyter Notebook with the addition of clickable UI based interface

Presentation and Report

- You should start preparing the final report at least 2 weeks prior to the project completion date.
- Teams should send a draft of last of the project before the last session to the mentor and get the necessary inputs for submission.
- The expectations for the final report will be included in your Capstone course page.

Details of Steps involved in Milestone-1:

As mentioned above the Project Milestone1 involves major 6 steps to arrive at the required interim results using Machine Learning techniques.

Libraries Used:

Notebook uses various library functions for further process & analysis of data.

1. Numpy – Used for reading large multidimensional Arrays and Matrices.
2. Pandas – Used for Data Manipulation & Analysis
3. Matplotlib.pyplot – Used for creating 2D graphics & plots
4. Seaborn – Used for printing the Graphs
5. Matplotlib.inline – Used for Probability Distributions and various Statistical functions

Library Functions Used:

1. Scipy – Scientific Python – Open-source library which provides more utility functions for Optimization, Statistics and signal processing. This has high level commands and classes for Visualizing and Manipulating Data.
2. Sklearn – Scikit Learn – Open-source ML library which features Various Classification, Regression and Clustering Algorithms such as SV media. It has Algorithmic Decision-making methods which can give Predictive Data Analysis.
3. Liner Regression, Logistic Regression, Decision Tree models, Random Forest Regression, Gradient Boosting Regression & Classification, K- Nearest Neighbors, Support Vector, Naive Bayes etc.
4. XGBoost – Extreme Gradient Boosting – is a scalable, distributed gradient boosted decision Tree ML Library. It provides Parallel tree boosting and is the leading library for Regression, Classification and Ranking problems.
5. HoloViews – is also an open-source ML library designed to make data analysis and Visualization seamless & simple. It is possible to express with few lines of code so lets us to focus on what we are trying to explore & convey, not on the process of plotting. Data Visualization is the essence of Data Science which captures the user's attention by giving Graphical representation to the Problem / solution.
6. TensorFlow – Allows to create a graphical representation of Dataflow which describe how Data moves through the graph. This platform helps in implementing best practices for Data Automation, Model tracking, Performance monitoring and Model retraining.
7. Nltk – Natural Language Toolkit is a platform for building Python programs that work with Human language data for applying in statistical NLP. It contains text processing libraries for Tokenization, Parsing, Classification, Stemming, Tagging and Semantic reasoning.

Import the Data

In general, data from the client are made available in .csv format. So, the Data import to be done from .csv file into the Jupyter notebook using relevant syntax.

In this step the comprehensive data being read from the path where it is located. Reading the data following can be basically known:

1. Information of the file and description of the content

```
Printing the Information of the Content of the CSV File

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 425 entries, 0 to 424
Data columns (total 11 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Unnamed: 0                425 non-null    int64
 1   Data                      425 non-null    object
 2   Countries                 425 non-null    object
 3   Local                     425 non-null    object
 4   Industry Sector           425 non-null    object
 5   Accident Level            425 non-null    object
 6   Potential Accident Level  425 non-null    object
 7   Genre                     425 non-null    object
 8   Employee or Third Party   425 non-null    object
 9   Critical Risk             425 non-null    object
 10  Description               425 non-null    object
dtypes: int64(1), object(10)
memory usage: 36.6+ KB
None
```

```
Printing the Description of the Content (Include all) in the CSV File

        Unnamed: 0                 Data   Countries     Local Industry Sector  \
count   425.000000                  425         425       425              425
unique         NaN                  287           3        12                3
top            NaN  2017-02-08 00:00:00  Country_01  Local_03          Mining
freq           NaN                    6         251        90              241
mean    224.084706                  NaN         NaN       NaN              NaN
std     125.526786                  NaN         NaN       NaN              NaN
min       0.000000                  NaN         NaN       NaN              NaN
25%     118.000000                  NaN         NaN       NaN              NaN
50%     226.000000                  NaN         NaN       NaN              NaN
75%     332.000000                  NaN         NaN       NaN              NaN
max     438.000000                  NaN         NaN       NaN              NaN

       Accident Level Potential Accident Level Genre Employee or Third Party  \
count             425                      425   425                      425
unique              5                        6     2                        3
top                 I                       IV  Male              Third Party
freq              316                      143   403                      189
mean              NaN                      NaN   NaN                      NaN
std               NaN                      NaN   NaN                      NaN
min               NaN                      NaN   NaN                      NaN
25%               NaN                      NaN   NaN                      NaN
50%               NaN                      NaN   NaN                      NaN
75%               NaN                      NaN   NaN                      NaN
max               NaN                      NaN   NaN                      NaN
```

2. First and Last Five Rows of CSV file

```
Printing the First 5 Rows of CSV File

   Unnamed: 0                 Data   Countries     Local Industry Sector  \
0           0  2016-01-01 00:00:00  Country_01  Local_01          Mining
1           1  2016-01-02 00:00:00  Country_02  Local_02          Mining
2           2  2016-01-06 00:00:00  Country_01  Local_03          Mining
3           3  2016-01-08 00:00:00  Country_01  Local_04          Mining
4           4  2016-01-10 00:00:00  Country_01  Local_04          Mining

  Accident Level Potential Accident Level Genre Employee or Third Party  \
0              I                       IV  Male             Third Party
1              I                       IV  Male                Employee
2              I                      III  Male    Third Party (Remote)
3              I                        I  Male             Third Party
4             IV                       IV  Male             Third Party

         Critical Risk                                        Description
0              Pressed  While removing the drill rod of the Jumbo 08 f...
1  Pressurized Systems  During the activation of a sodium sulphide pum...
2         Manual Tools  In the sub-station MILPO located at level +170...
3               Others  Being 9:45 am. approximately in the Nv. 1880 C...
4               Others  Approximately at 11:45 a.m. in circumstances t...



Printing the Last 5 Rows of CSV File

     Unnamed: 0                 Data   Countries     Local Industry Sector  \
420         434  2017-07-04 00:00:00  Country_01  Local_04          Mining
421         435  2017-07-04 00:00:00  Country_01  Local_03          Mining
422         436  2017-07-05 00:00:00  Country_02  Local_09          Metals
423         437  2017-07-06 00:00:00  Country_02  Local_05          Metals
424         438  2017-07-09 00:00:00  Country_01  Local_04          Mining

    Accident Level Potential Accident Level   Genre Employee or Third Party  \
420              I                      III    Male             Third Party
421              I                       II  Female                Employee
422              I                       II    Male                Employee
423              I                       II    Male                Employee
424              I                       II  Female             Third Party

                    Critical Risk  \
420                         Others
421                         Others
422              Venomous Animals
423                           Cut
424  Fall prevention (same level)

                                      Description
420  Being approximately 5:00 a.m. approximately, w...
421  The collaborator moved from the infrastructure...
422  During the environmental monitoring activity i...
423  The Employee performed the activity of strippi...
424  At 10:00 a.m., when the assistant cleaned the ...
++++++++++++++============================================================+++++++++++++++
```

3. The description of the content in Transpose format

```
Printing the Description of the Content in the CSV File in Transpose Format(Including all)

                         count unique  \
Unnamed: 0               425.0    NaN
Data                       425    287
Countries                  425      3
Local                      425     12
Industry Sector            425      3
Accident Level             425      5
Potential Accident Level   425      6
Genre                      425      2
Employee or Third Party    425      3
Critical Risk              425     33
Description                425    411

                                                                  top  \
Unnamed: 0                                                        NaN
Data                                                  2017-02-08 00:00:00
Countries                                                    Country_01
Local                                                          Local_03
Industry Sector                                                  Mining
Accident Level                                                        I
Potential Accident Level                                             IV
Genre                                                              Male
Employee or Third Party                                     Third Party
Critical Risk                                                    Others
Description              During the activity of chuteo of ore in hopper...


                          freq        mean         std  min    25%    50%  \
Unnamed: 0                 NaN  224.084706  125.526786  0.0  118.0  226.0
Data                         6         NaN         NaN  NaN    NaN    NaN
Countries                  251         NaN         NaN  NaN    NaN    NaN
Local                       90         NaN         NaN  NaN    NaN    NaN
Industry Sector            241         NaN         NaN  NaN    NaN    NaN
Accident Level             316         NaN         NaN  NaN    NaN    NaN
Potential Accident Level   143         NaN         NaN  NaN    NaN    NaN
Genre                      403         NaN         NaN  NaN    NaN    NaN
Employee or Third Party    189         NaN         NaN  NaN    NaN    NaN
Critical Risk              232         NaN         NaN  NaN    NaN    NaN
Description                  3         NaN         NaN  NaN    NaN    NaN

                           75%    max
Unnamed: 0               332.0  438.0
Data                       NaN    NaN
Countries                  NaN    NaN
Local                      NaN    NaN
Industry Sector            NaN    NaN
Accident Level             NaN    NaN
Potential Accident Level   NaN    NaN
Genre                      NaN    NaN
Employee or Third Party    NaN    NaN
Critical Risk              NaN    NaN
Description                NaN    NaN
```

4. Shape of the CSV file

```
Printing the SHAPE of the CSV File

(425, 11)
++++++++++++++============================================================+++++++++++++++
```

5. Columns of CSV file

```
Printing the Columns of the CSV File

Index(['Unnamed: 0', 'Data', 'Countries', 'Local', 'Industry Sector',
       'Accident Level', 'Potential Accident Level', 'Genre',
       'Employee or Third Party', 'Critical Risk', 'Description'],
      dtype='object')
++++++++++++++============================================================+++++++++++++++
```

6. Data types of columns in the CSV file

```
Printing the DataTypes of All Columns in the CSV File

Unnamed: 0                 int64
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
++++++++++++++============================================================+++++++++++++++
```

Data Cleansing

Data Cleansing is an essential process since, regardless how sophisticated the ML algorithm is, there is no possibility to obtain good results with bad data. Having clean data will ultimately increase overall productivity and allow for the highest quality information in decision making. Properly formatted data can help removal of errors or having fewer errors are the main purpose of Cleansing the data.

1. It starts in this case from merging the two files IndSafty.csv and IndSfty.xls. After Concatenating the files all the above insights to be checked to ensure the proper merging.
2. Merged data to be checked for Null and Duplicate values.
3. Changing the Data column datatype to datetime64[ns] solved in finding the duplicates in the data frame.
4. After changing the data column, we found the 2 files csv and xlsx files are same and contains similar data
5. Drop the Null and Duplicate values to make the dataset more accurate for further process.
6. We have dropped unnecessary columns like Unnamed: 0 which further helped in finding the duplicates.
7. After that we dropped duplicates from the data frame we have checked for Null Values in the data.
8. We have changed the names of columns to some meaning full names like data to Date and etc.

Final merged data looks as follows:

```
In [18]:
# Final Dataframe after removing duplicates

df_IndSafety_Final
```

Out[18]:

	Date	Country	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Description
0	2016-01-01	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...
1	2016-01-02	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...
2	2016-01-06	Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...
3	2016-01-08	Country_01	Local_04	Mining	I	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 C...
4	2016-01-10	Country_01	Local_04	Mining	IV	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances t...
...	...	...	...	...	...	...	...	...	...	...
413	2017-07-04	Country_01	Local_04	Mining	I	III	Male	Third Party	Others	Being approximately 5:00 a.m. approximately, w...
414	2017-07-04	Country_01	Local_03	Mining	I	II	Female	Employee	Others	The collaborator moved from the infrastructure...
415	2017-07-05	Country_02	Local_09	Metals	I	II	Male	Employee	Venomous Animals	During the environmental monitoring activity i...
416	2017-07-06	Country_02	Local_05	Metals	I	II	Male	Employee	Cut	The Employee performed the activity of strippi...
417	2017-07-09	Country_01	Local_04	Mining	I	II	Female	Third Party	Fall prevention (same level)	At 10:00 a.m., when the assistant cleaned the ...

418 rows × 10 columns

Following are the observation with above data

1. There are records of accidents in every month from 1st Jan 2016 to 9th July. So there are no Outliers in ‘Date’ column.
2. Only three country types so no ourliers in ‘Country’ column.
3. There are 12 local cities where manufacturing plant is located whose types are in sequence. Hence no outliers in ‘Local’ column.
4. There are only three industry sector types which are in sequence, hence no outliers in ‘Industry Sector’ column.
5. There are only five Accident Level types which are in sequence so there are no outliers in 'Accident Level' column.
6. There are only six Potential Accident Level types which are in sequence so there are no outliers in 'Potential Accident Level' column.
7. There are only two Gender types in the provided data so there are no outliers in 'Gender' column.
8. There are only three Employee types in the provided data so there are no outliers in 'Gender' column.
9. There are quite a lot of Critical risk descriptions and we don't see any outliers but with the help of SME we can decide whether this column has outliers or not.

Summary after Data Cleansing:

1. Removed 'Unnamed: 0' column and renamed - 'Data', 'Countries', 'Genre', 'Employee or Third Party' columns in the dataset.
2. We had 7 duplicate instances in the dataset and dropped those duplicates.
3. There are no outliers in the dataset.
4. No missing values in dataset.
5. We are left with 418 rows and 10 columns after data cleansing.

In this step, to better understand the data, we are extracting the Year, Month_Year, Day_Name, Week_Of_Year, Quarter from Date column and creating new features such as weekday, week of year.

In [25]:
```
# Adding 5 more columns "Year", "Month_Year", "Day Name", "Week Of Year", "Quarter" fetching from Date Column for I

df_IndSafety_Final['Year'] = df_IndSafety_Final['Date'].dt.year
df_IndSafety_Final['Month_Year'] = df_IndSafety_Final['Date'].dt.to_period('M')
df_IndSafety_Final['Day_Name'] = df_IndSafety_Final['Date'].dt.day_name()
df_IndSafety_Final['Week_Of_Year'] = df_IndSafety_Final['Date'].dt.weekofyear
df_IndSafety_Final['Quarter'] = df_IndSafety_Final['Date'].dt.quarter

```

Results are as follows:

Out[27]:

	Date	Country	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Description	Year	Month_Year	Day_Name	Week_Of_Year	Quarter
0	2016-01-01	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...	2016	2016-01	Friday	53	1
1	2016-01-02	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...	2016	2016-01	Saturday	53	1
2	2016-01-06	Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...	2016	2016-01	Wednesday	1	1

It is essential also to check for Brazilian National holidays for whole year. This gives an insight to arrive the working days and non-working days. Even if it is non-working days plants may operate with minimum strength to run the production activities and hence to determine the number of operations, people working etc.

```
--------------------------------------------------------------------------------
List of Brazil holidays in 2016
--------------------------------------------------------------------------------
(datetime.date(2016, 1, 1), 'Confraternização Universal')
(datetime.date(2016, 3, 25), 'Sexta-feira Santa')
(datetime.date(2016, 4, 21), 'Tiradentes')
(datetime.date(2016, 5, 1), 'Dia do Trabalhador')
(datetime.date(2016, 9, 7), 'Independência do Brasil')
(datetime.date(2016, 10, 12), 'Nossa Senhora Aparecida')
(datetime.date(2016, 11, 2), 'Finados')
(datetime.date(2016, 11, 15), 'Proclamação da República')
(datetime.date(2016, 12, 25), 'Natal')
(datetime.date(2016, 2, 8), 'Carnaval')
(datetime.date(2016, 2, 9), 'Carnaval')
(datetime.date(2016, 2, 10), 'Início da Quaresma')
(datetime.date(2016, 5, 26), 'Corpus Christi')
(datetime.date(2016, 10, 28), 'Dia do Servidor Público')
(datetime.date(2016, 12, 24), 'Véspera de Natal')
(datetime.date(2016, 12, 31), 'Véspera de Ano-Novo')
--------------------------------------------------------------------------------
List of Brazil holidays in 2017
--------------------------------------------------------------------------------
(datetime.date(2017, 1, 1), 'Confraternização Universal')
(datetime.date(2017, 4, 14), 'Sexta-feira Santa')
(datetime.date(2017, 4, 21), 'Tiradentes')
(datetime.date(2017, 5, 1), 'Dia do Trabalhador')
(datetime.date(2017, 9, 7), 'Independência do Brasil')
(datetime.date(2017, 10, 12), 'Nossa Senhora Aparecida')
(datetime.date(2017, 11, 2), 'Finados')
(datetime.date(2017, 11, 15), 'Proclamação da República')
(datetime.date(2017, 12, 25), 'Natal')
(datetime.date(2017, 2, 27), 'Carnaval')
(datetime.date(2017, 2, 28), 'Carnaval')
(datetime.date(2017, 3, 1), 'Início da Quaresma')
(datetime.date(2017, 6, 15), 'Corpus Christi')
(datetime.date(2017, 10, 28), 'Dia do Servidor Público')
(datetime.date(2017, 12, 24), 'Véspera de Natal')
(datetime.date(2017, 12, 31), 'Véspera de Ano-Novo')
```

After Adding the column, the data frame looks like below:

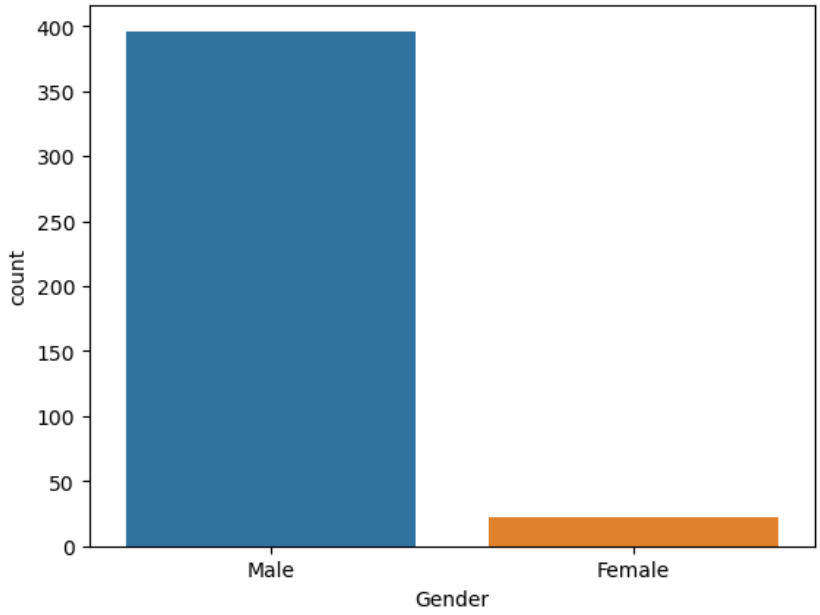
```
In [27]:
df_IndSafety_Final['Is_Holiday'] = [1 if str(val).split()[0] in brazil_holidays else 0 for val in df_IndSafety_Fina
df_IndSafety_Final.head(3)
```

Out[27]:

Country	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Description	Year	Month_Year	Day_Name	Week_Of_Year	Quarter	Is_Holiday
Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...	2016	2016-01	Friday	53	1	1
Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...	2016	2016-01	Saturday	53	1	0
Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...	2016	2016-01	Wednesday	1	1	0

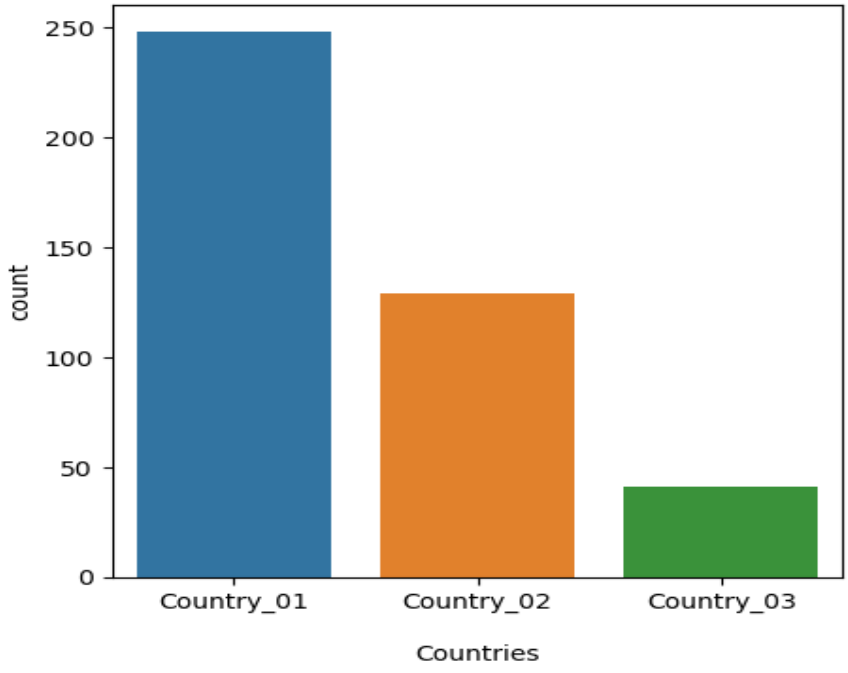
Pictographical Analysis of the Dataset

I. To find the Gender distribution across the industry in dataset

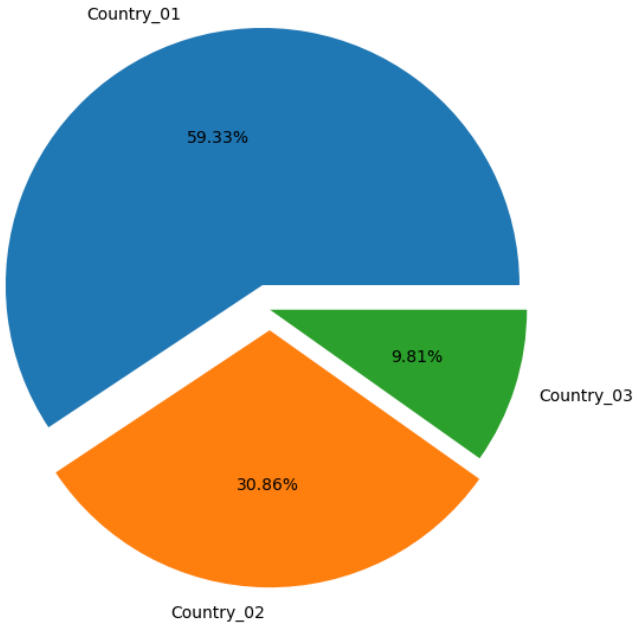


As per this data there are about 390 Male and 25 Female employees.

II. Following bar chart will show the distribution of country-wise accidents



III. Same data can be displayed in pie chart for more clarity

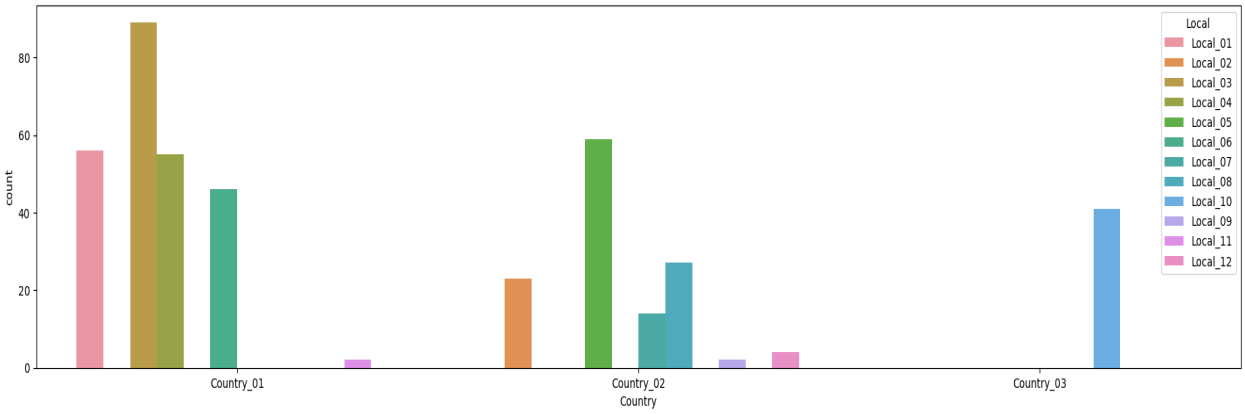


From the above two figures following can be inferred:

a. Country 01 has more number accidents with 59.33%
b. Country 02 has 30.86% and Country 03 has least of 9.81%
c. It shows number of operations are more and high in volume in Country 01 as compared to Country 02 & 03.
d. Needs more focus on Country 01 & Country 02 for their operations.
e. If there are any good practices in Country 03, to be adapted in other countries.

IV. Inside every country there are few operating plants.

Their distribution can be drawn as follows:

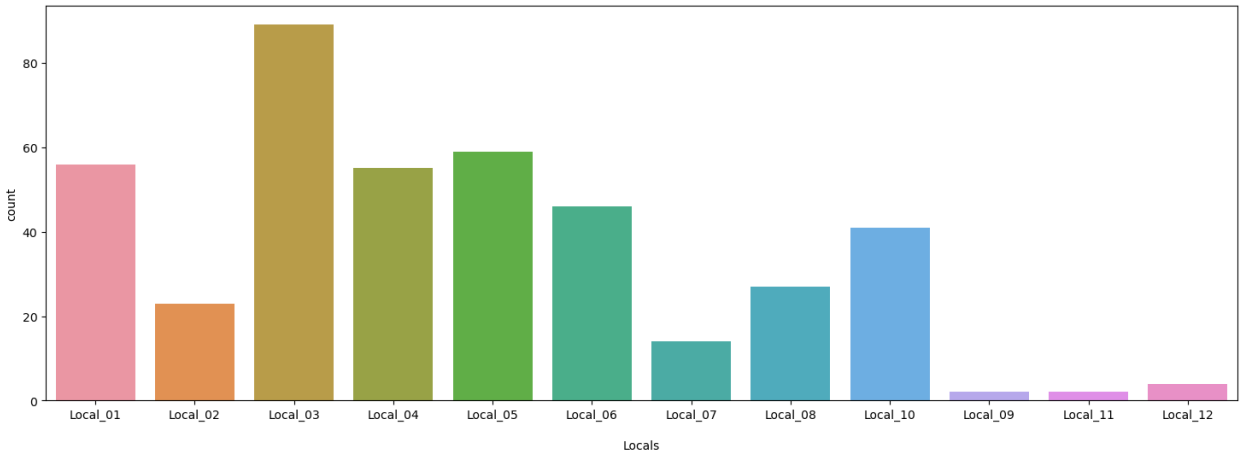


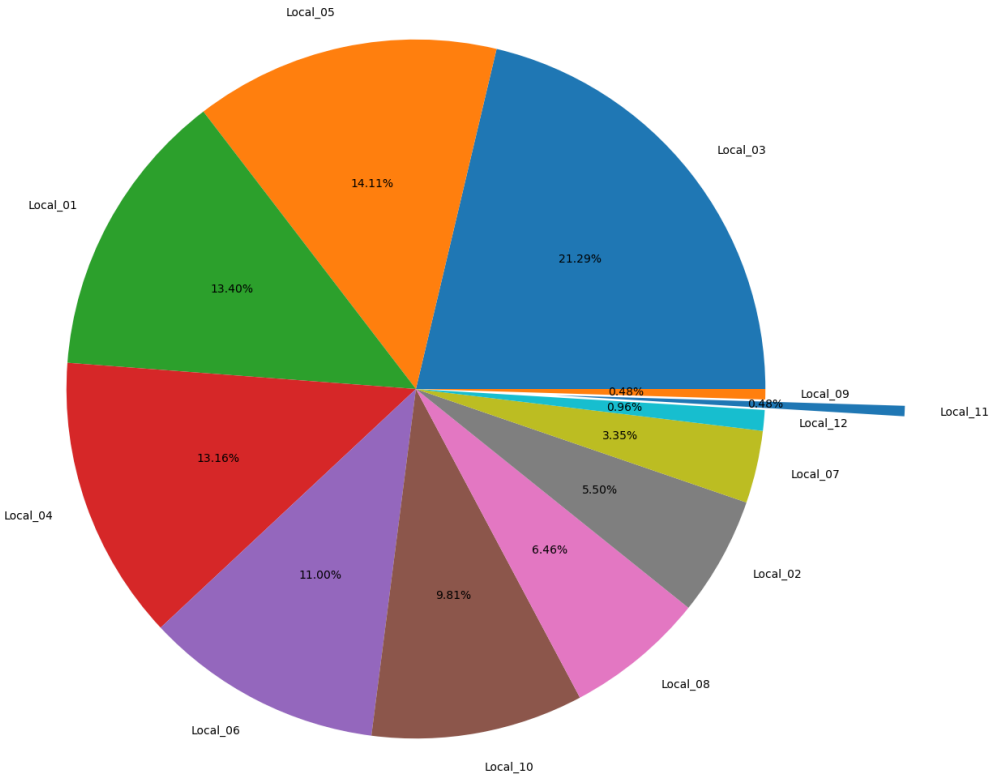
Following can be further inferred:

a. Country 02 has more number of plants – Local 06 – 11
b. Country 01 has more number of accidents – Local 01 – 05
c. Country 03 has only one plant in operation – Local 12

V. Further the for all the countries the Plant (location) wise data

This is plotted as follows:





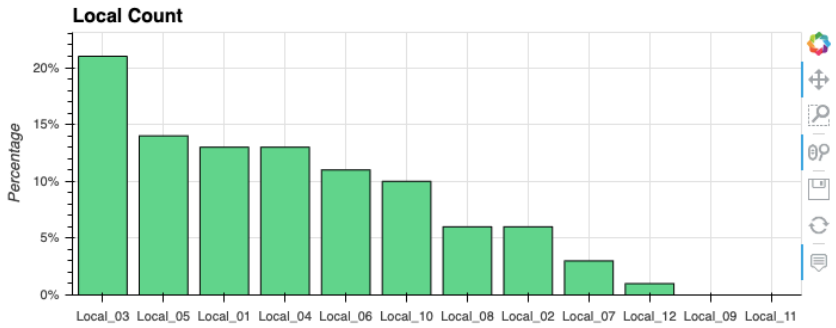
The above figures gives us the following insights:

a. Local 03 which belongs to Country 01 has highest number of accidents 21.29%
b. Followed by Local 05, 01, 04
c. Focus need to be on Local 03 to the volume and operation to reduce or eliminate number of accidents
d. Though Country 02 has more number of plants the number of accidents are relatively lesser.
e. If Country 02 has any good practices to be shared with Country 01 also.
f. In Country 02 only Local 06 has 11% and rest all are lesser.

In [39]:
```
# Displaying Some interractive Pictorial Representations.

local_cnt = np.round(df_merged_final['Local'].value_counts(normalize=True) * 100)

hv.extension('bokeh')
hv.Bars(local_cnt).opts(title="Local Count", color="#58D68D", xlabel="Locals", ylabel="Percentage", yformatter='%
                .opts(opts.Bars(width=700, height=300,tools=['hover'],show_grid=True))

```

executed in 7.98s, finished 15:55:41 2023-06-04

Out[39]:

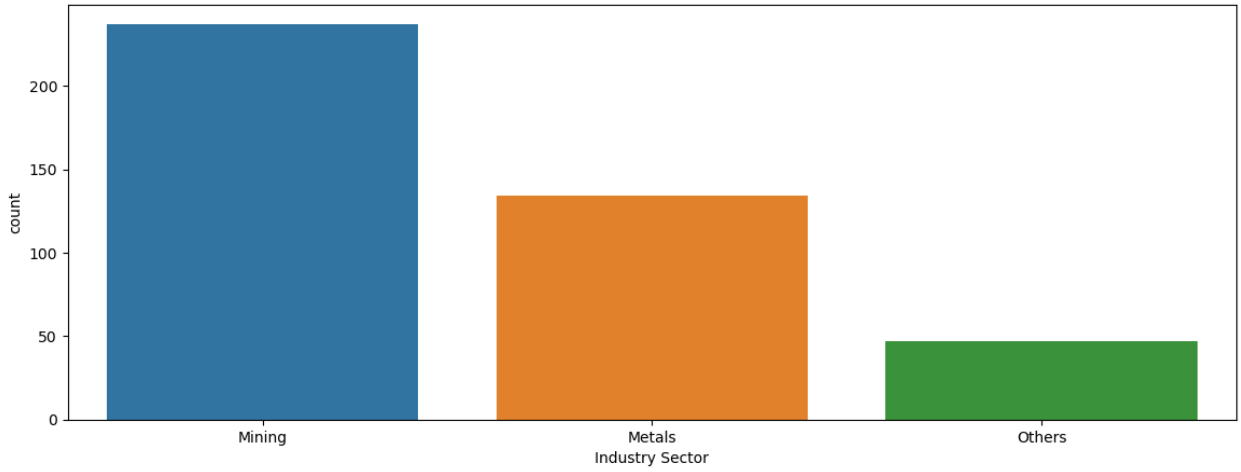


The above figure is an ***interactive way of indicating*** the same distribution as Pie Chart for number of Accidents in percentage versus Local Plants irrespctive of country.

Speciality of this plot is that it allows the User to expand / collapse the view for whichever section someone wants to see.

That allows the User to have more closer look at the data which can even show any perticular section of data to be analyzed more.

VI. The Industry sector-wise data can be plotted as follows:

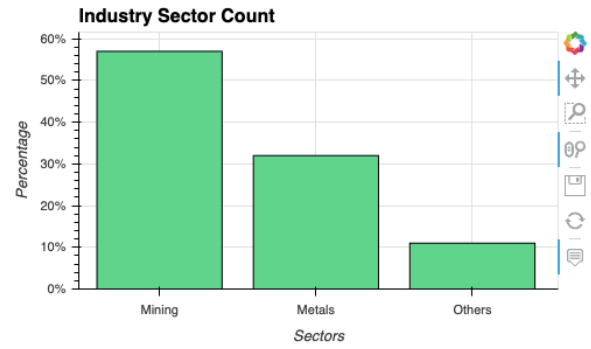


Above figure shows following pattern:

a. The accidents are spread across three sectors viz., Mining, Metals and Others.
b. Mining has more number of accidents followed by Metals sector.
c. Mining accounts to more than 55% of the Accidents
d. Metals Sector is prone to more than 30% of the Accidents.
e. Others are least but can't be ignored.

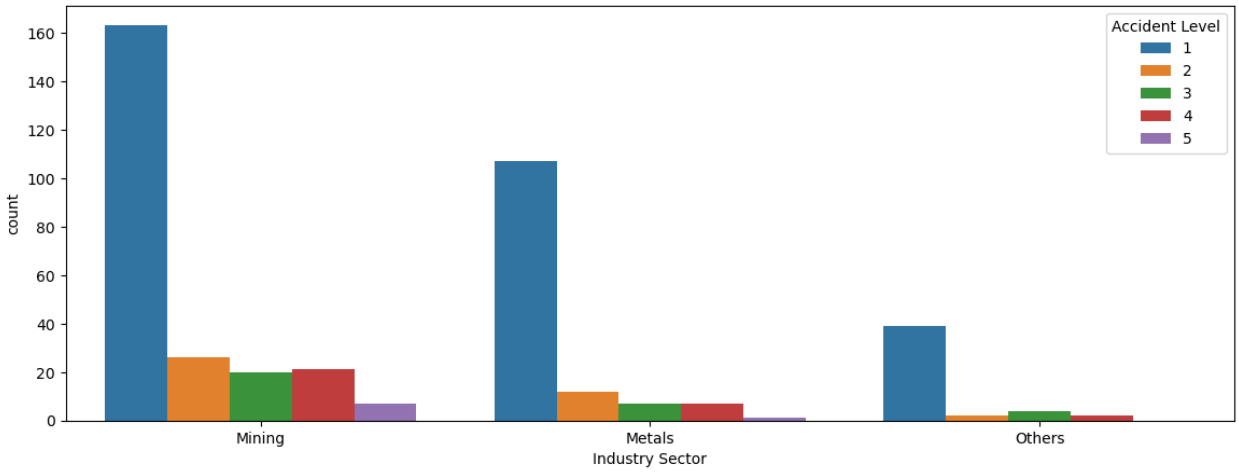
```
------------------------------------------------------------
Value Counts for `Industry Sector` label
------------------------------------------------------------
Mining count: 237 i.e. 57.0%
Metals count: 134 i.e. 32.0%
Others count: 47 i.e. 11.0%
------------------------------------------------------------
Distributon of `Industry Sector` label
------------------------------------------------------------
```

Out[40]:



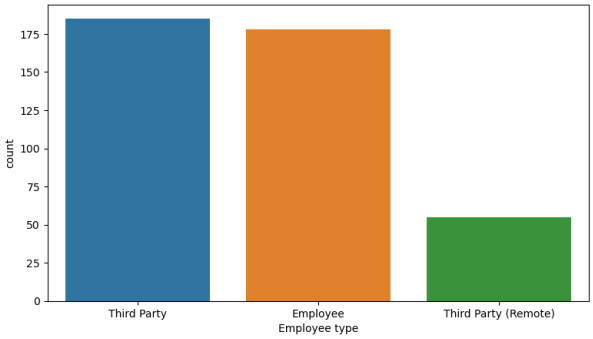
Similar to the previous display, even this shows the Industry sector-wise versus number of Accidents in percentage. Even this being interactive plot allows user to navigate through the sector data more closely.

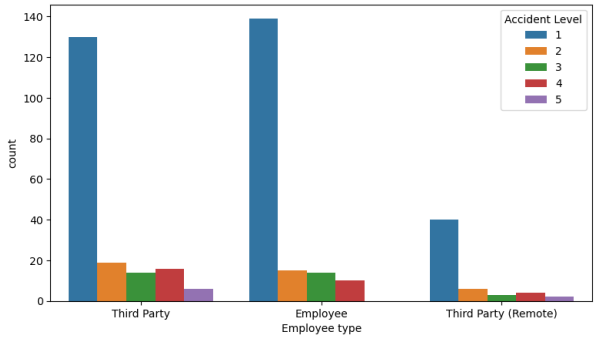
VII. Following Plot shows Sector-wise number & levels of accidents



a. Irrespective of Sector Accident level -1 are more in number which is alarming and to be addressed on priority.
b. Mining sector has more number of Level 2,3,4 also

VIII. Following shows the Employee type wise distribution





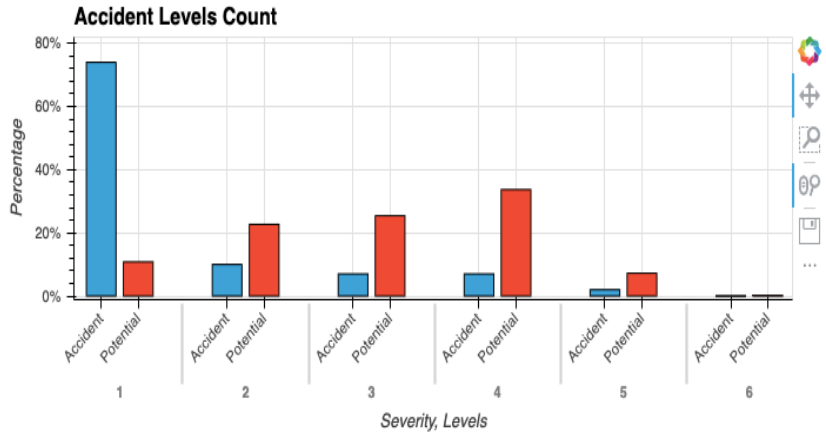
Observations are as follows:

a. The number of third party employees are more in number
b. Third party remote operation are lesser in number compared to plant employees.
c. As seen sector wise distribution even here the number of Accident Level 1 are more in all the employees irrespective of their types.
d. More focus needed on Accident Level 1 to understand the type of accidents.
e. Root cause analysis to be carried for each Accident Level 1 followed by other levels of Accident.
f. More plant employees come across accidents than remote operations. So more emphasize on Automation than on the personal involvement to minimize accidents.

IX. Plotting Accidents versus Potential Accident levels

```
------------------------------------------------------------
Value Counts for `Accident Level` label
------------------------------------------------------------------------------
Accident Level - I count: 309 i.e. 74.0%
Accident Level - II count: 40 i.e. 10.0%
Accident Level - III count: 31 i.e. 7.0%
Accident Level - IV count: 30 i.e. 7.0%
Accident Level - V count: 8 i.e. 2.0%
Accident Level - VI count: 0 i.e. 0.0%
------------------------------------------------------------
Value Counts for `Potential Accident Level
------------------------------------------------------------------------------
Potential Accident Level - I count: 45 i.e. 11.0%
Potential Accident Level - II count: 95 i.e. 23.0%
Potential Accident Level - III count: 106 i.e. 25.0%
Potential Accident Level - IV count: 141 i.e. 34.0%
Potential Accident Level - V count: 30 i.e. 7.0%
Potential Accident Level - VI count: 1 i.e. 0.0%
------------------------------------------------------------
Distributon of `Accident Level` & `Potential Accident Level` label
------------------------------------------------------------------------------
```

Out[41]:



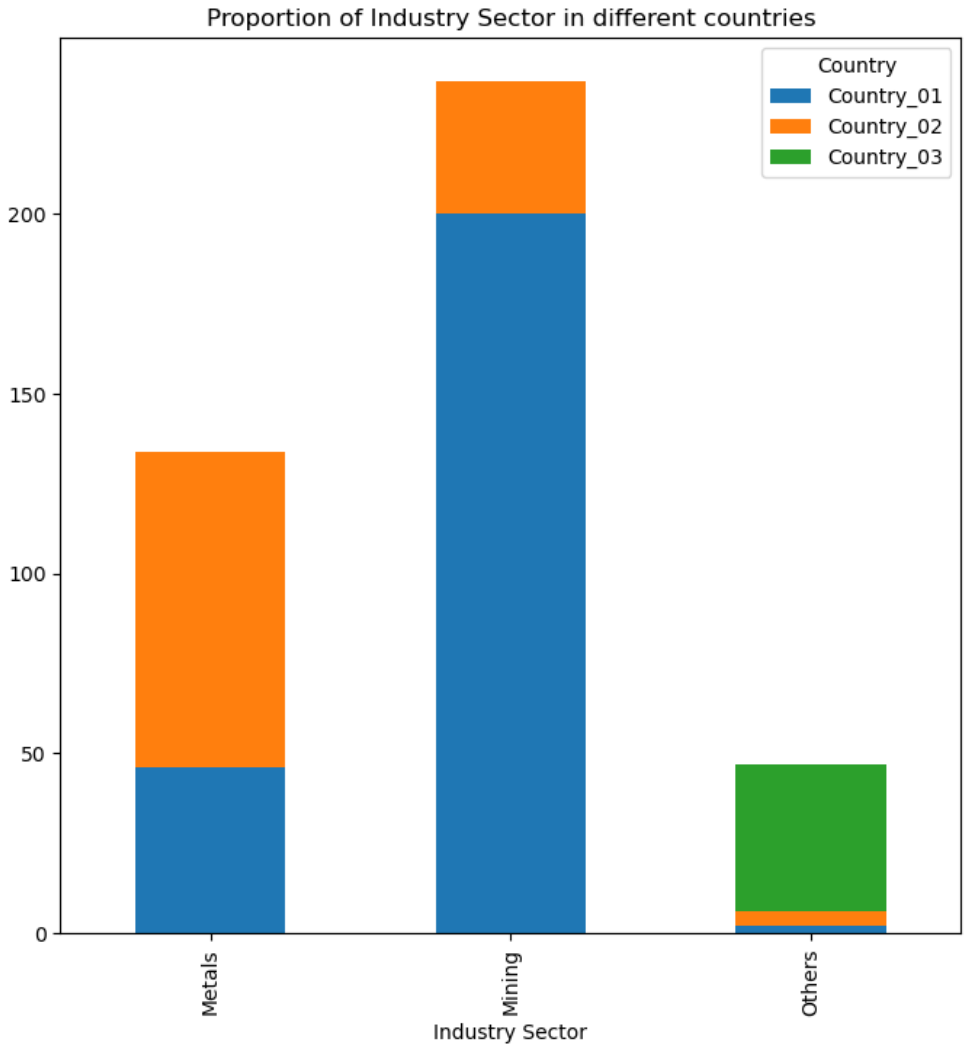
This Plot gives a nice representation of data distribution with a prediction of Potential Accident for each level of Accidents. Following can be inferred:

a. Though number of Accident Level -1 is high in number 74% against that the Potential is only 11%. This indicates there is sufficient ways to curtail Level-1 Accidents and they are controllable in nature.

b. Though number of Accident Level-2, 3, 4 & 5 are less, Potential accident levels 2, 3, 4 & 5 are high. This shows they are either have less control or poor planing leading to risk.

c. So it is required to take adequate measures to curtail the Level 2,3,4 & 5. Else they prove to be at risk.

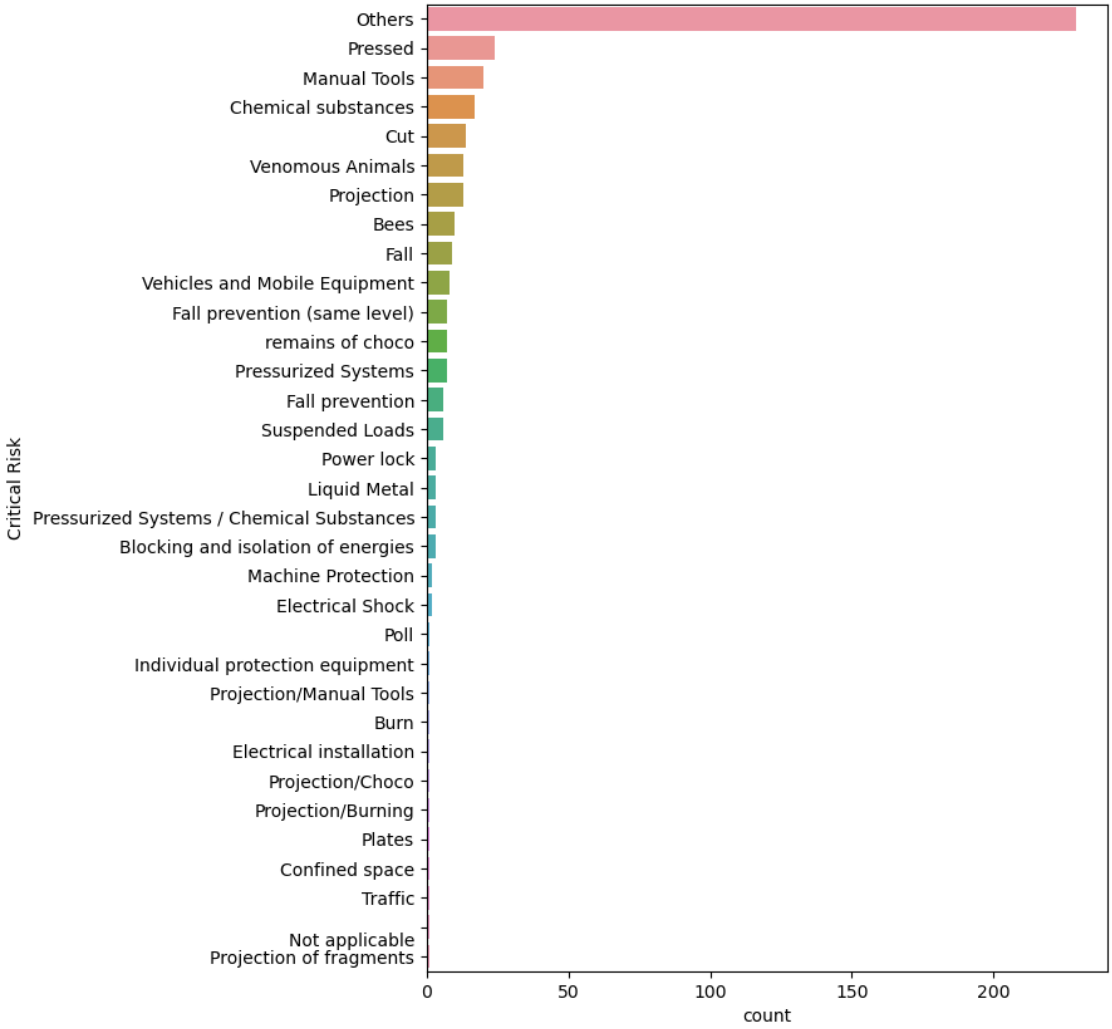
X. Distribution of Industry sector across different Countries



This plot gives another view point for data analysis.

a. Metals and Mining are spread across Country 01 & 02 only
b. Other sector are only in Country 03.
c. So the best practices can be shared between Country 01 & 02 to maintain the safer workplace.
d. Based on Accident level in Country-03 best practices can be adopted from Country 01 & 02.

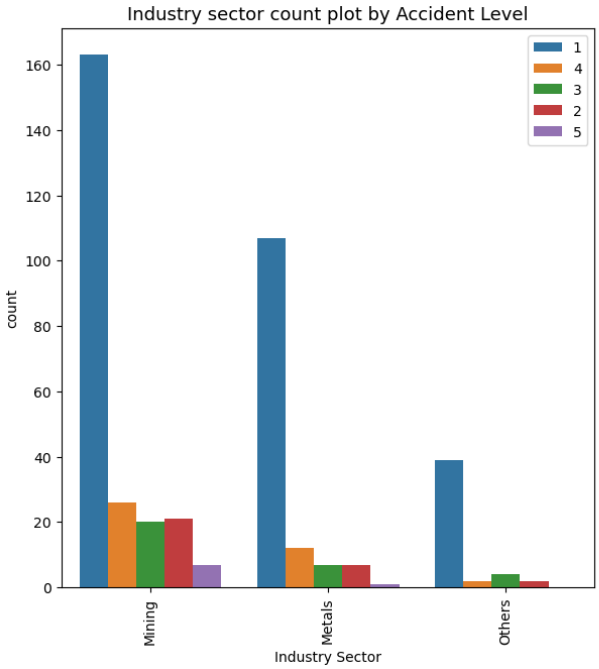
XI. Distribution of Accident Counts versus Critical Risk events

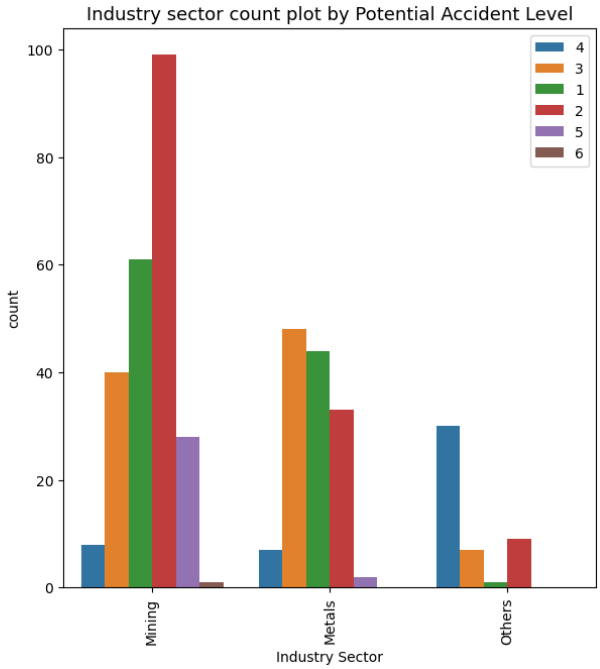


The above distribution gives an insight to the cause of various accidents taken place.

a. There are about 33 categories of causes for Accidents arrived from the Data provided.
b. Among them 14 categories are least in number.
c. 12 categories take next higher number.
d. 7 categories are next higher number. These include the environmental hazards like venomous animals & bees, manual failures like usage of wrong tools, projection, ignorance in chemical substances, cut etc.
e. There is many ignorance towards safety habit which has caused vehicle movement not taken care, fall prevention, projection & suspended loads not managed.
f. Category “others” are highest in number. As they are listed as without any specific area of risk and only mentioned as others, it loses the focus of any specific risk.
g. User should capture or segregate the others type into more specific category for better accuracy of analysis.

XII. Distribution of Industry sector-wise Accident Level as well as Potential Accident Level





Few insights are as follows:

a. Mining Sector has highest number of Accidents of Accident Level 1 and Potential Accident Level 2.

b. Category Others to be evaluated with respect to Accident Level 1 and Potential Accident Level 2 as this can give clarity on type of critical risk which can be rectified.

c. Metals Sector has highest number of Accidents of Accident Level 1 and Potential Accident Level 3.

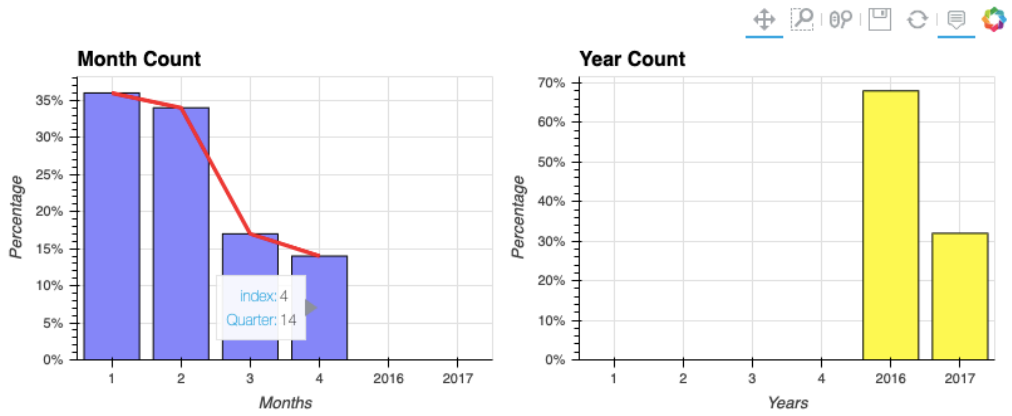
d. Category Others to be evaluated with respect to Accident Level 1 and Potential Accident Level 3 as this can give clarity on type of critical risk which can be rectified.

e. In Other Sector, Accidents of Accident Level 1 and Potential Accident Level 4 are higher.

f. Irrespective of Industry sector Accident Level 1 is higher in counts. This need to be verified.

XIII. Year Month wise Percentage View:

Out[48]:

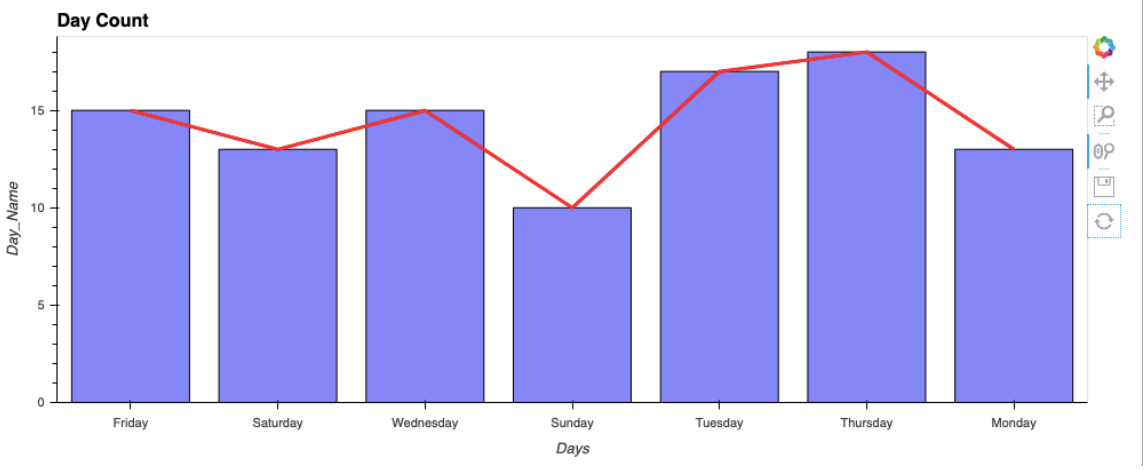


XIV. Day Count wise

In [49]:

```
1 day_cnt = np.round(df_IndSafety_Final['Day_Name'].value_counts(normalize=True,sort=False) * 100)
2 hv.Bars(day_cnt).opts(title="Day Count", color="#8888ff", xlabel="Days") * hv.Curve(day_cnt).opts(width=980, height
3
```

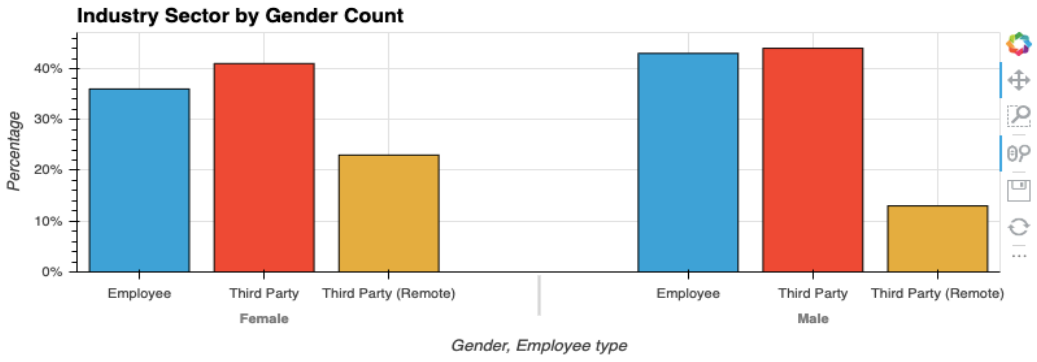
Out[49]:



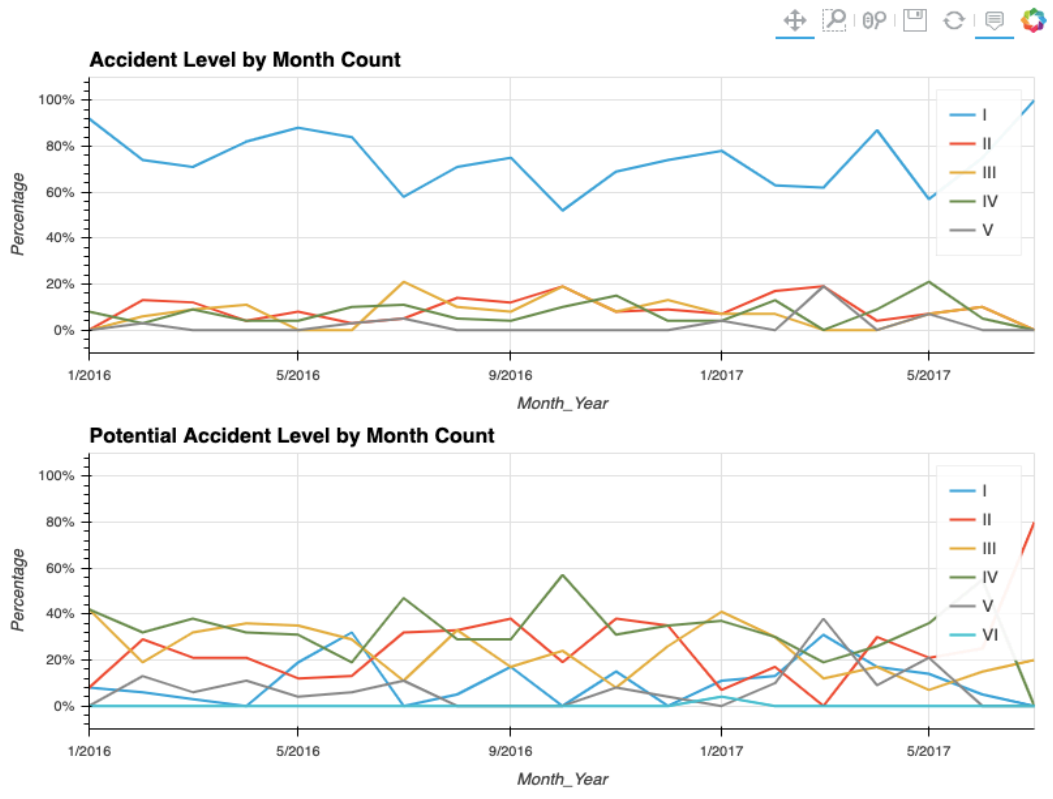
XV. Some other Analyses:

Employee type	Employee	Third Party	Third Party (Remote)
Gender			
Female	36.0	41.0	23.0
Male	43.0	44.0	13.0

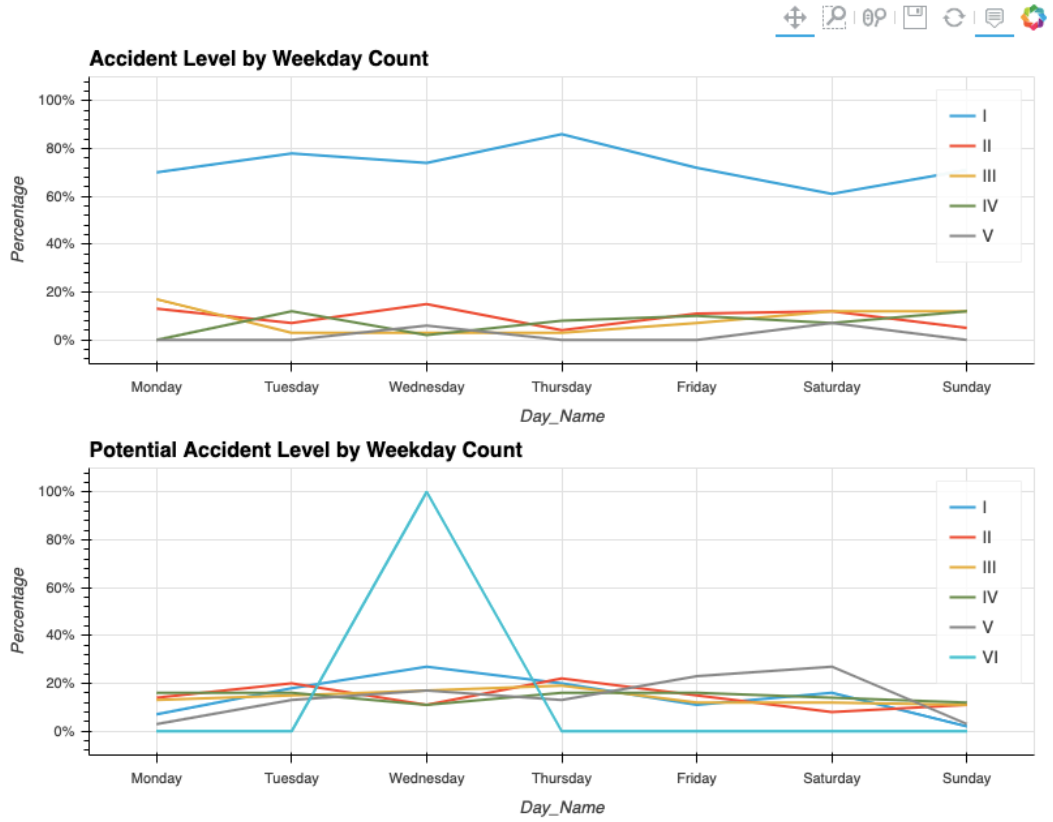
Out[52]:



Out[57]:

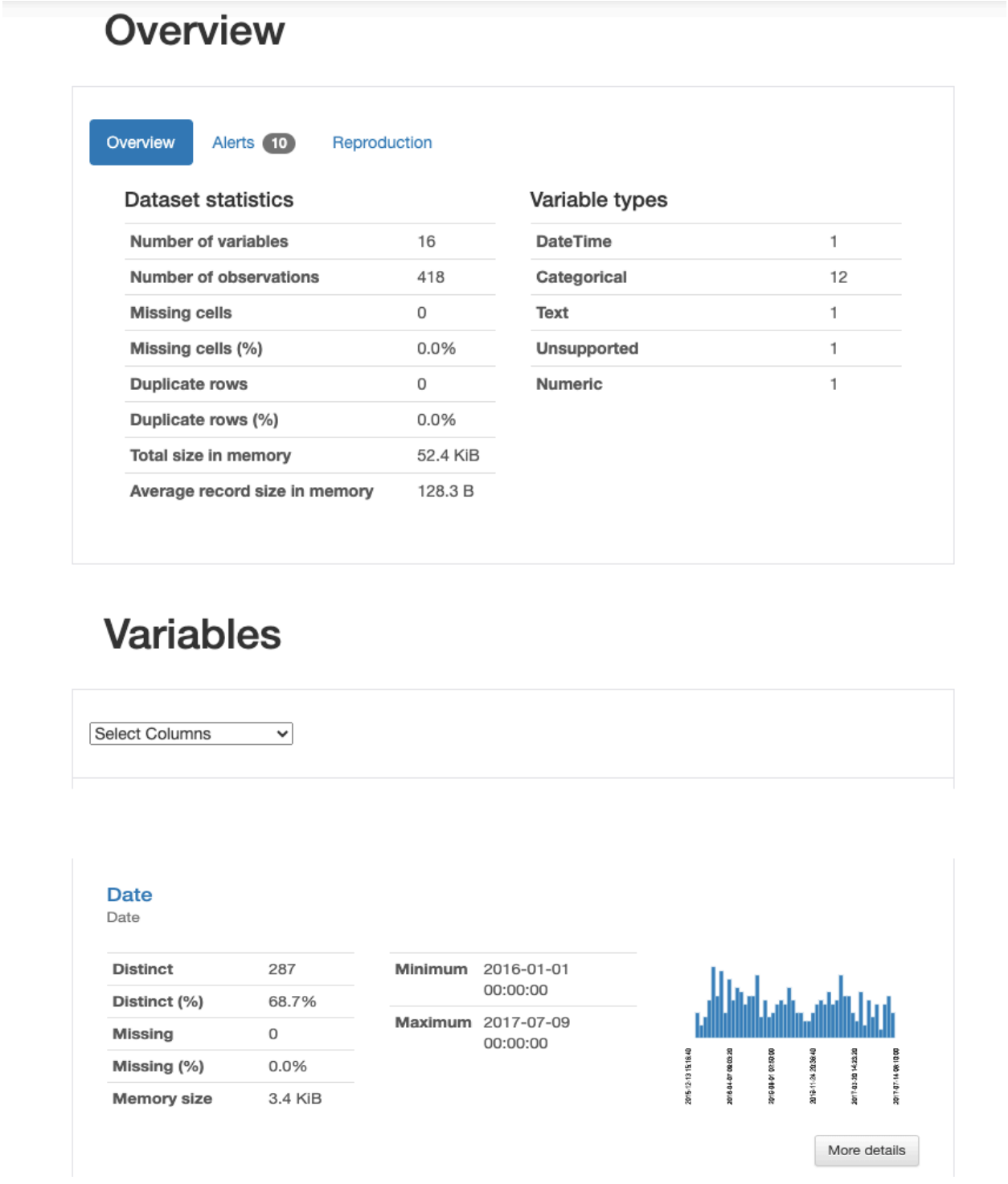

Accident Level by Month Count
Percentage
100%
80%
60%
40%
20%
0%
1/2016
5/2016
9/2016
1/2017
5/2017
Month_Year
I
II
III
IV
V
Potential Accident Level by Month Count
Percentage
100%
80%
60%
40%
20%
0%
1/2016
5/2016
9/2016
1/2017
5/2017
Month_Year
I
II
III
IV
V
VI

Out[58]:


Accident Level by Weekday Count
Percentage
100%
80%
60%
40%
20%
0%
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
Day_Name
I
II
III
IV
V
Potential Accident Level by Weekday Count
Percentage
100%
80%
60%
40%
20%
0%
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
Day_Name
I
II
III
IV
V
VI

XVI. Pandas Profiling Report:

We have used Pandas Profiling to generate the Overview so far, whose output is as follows:



Overview

Overview | Alerts 10 | Reproduction

Dataset statistics

Number of variables	16
Number of observations	418
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	52.4 KiB
Average record size in memory	128.3 B

Variable types

DateTime	1
Categorical	12
Text	1
Unsupported	1
Numeric	1

Variables

Select Columns

Date

Date

Distinct	287
Distinct (%)	68.7%
Missing	0
Missing (%)	0.0%
Memory size	3.4 KiB

Minimum	2016-01-01 00:00:00
Maximum	2017-07-09 00:00:00

More details

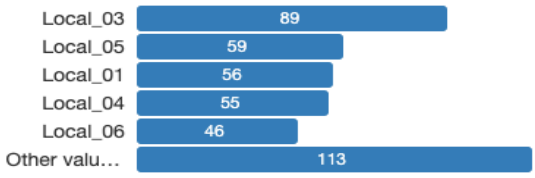
Country
Categorical

Distinct	3
Distinct (%)	0.7%
Missing	0
Missing (%)	0.0%
Memory size	3.4 KiB

More details

Local
Categorical

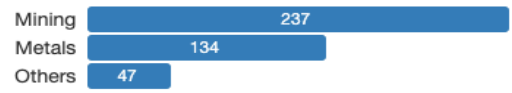
Distinct	12
Distinct (%)	2.9%
Missing	0
Missing (%)	0.0%
Memory size	3.4 KiB



More details

Industry Sector
Categorical

Distinct	3
Distinct (%)	0.7%
Missing	0
Missing (%)	0.0%
Memory size	3.4 KiB



More details

Accident Level
Categorical

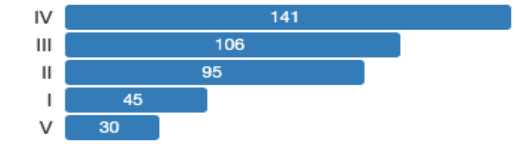
Distinct	5
Distinct (%)	1.2%
Missing	0
Missing (%)	0.0%
Memory size	3.4 KiB

More details

Potential Accident Level

Categorical

Distinct	6
Distinct (%)	1.4%
Missing	0
Missing (%)	0.0%
Memory size	3.4 KiB



More details

Gender

Categorical

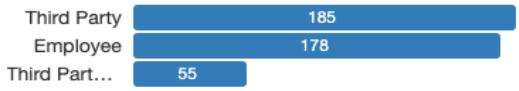
Distinct	2
Distinct (%)	0.5%
Missing	0
Missing (%)	0.0%
Memory size	3.4 KiB

More details

Employee type

Categorical

Distinct	3
Distinct (%)	0.7%
Missing	0
Missing (%)	0.0%
Memory size	3.4 KiB

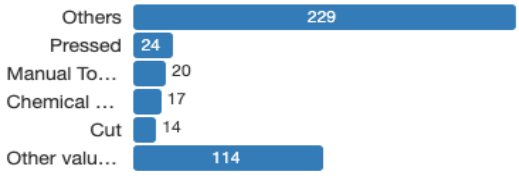


More details

Critical Risk

Categorical

Distinct	33
Distinct (%)	7.9%
Missing	0
Missing (%)	0.0%
Memory size	3.4 KiB



More details

Description
Text

Distinct	411
Distinct (%)	98.3%
Missing	0
Missing (%)	0.0%
Memory size	3.4 KiB

More details

Year
Categorical

Distinct	2
Distinct (%)	0.5%
Missing	0
Missing (%)	0.0%
Memory size	3.4 KiB

More details

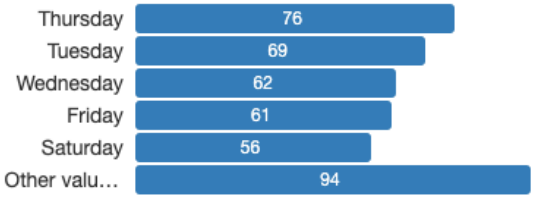
~~Month_Year~~
~~Unsupported~~

REJECTED UNSUPPORTED

Missing	0
Missing (%)	0.0%
Memory size	3.4 KiB

Day_Name
Categorical

Distinct	7
Distinct (%)	1.7%
Missing	0
Missing (%)	0.0%
Memory size	3.4 KiB

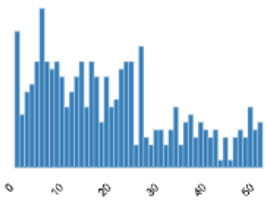


More details

Week_Of_Year

Real number (ℝ)

Distinct	53	**Minimum**	1
Distinct (%)	12.7%	**Maximum**	53
Missing	0	**Zeros**	0
Missing (%)	0.0%	**Zeros (%)**	0.0%
Infinite	0	**Negative**	0
Infinite (%)	0.0%	**Negative (%)**	0.0%
Mean	21.033493	**Memory size**	3.4 KiB

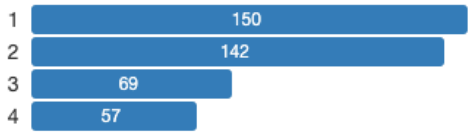


More details

Quarter

Categorical

Distinct	4
Distinct (%)	1.0%
Missing	0
Missing (%)	0.0%
Memory size	3.4 KiB



More details

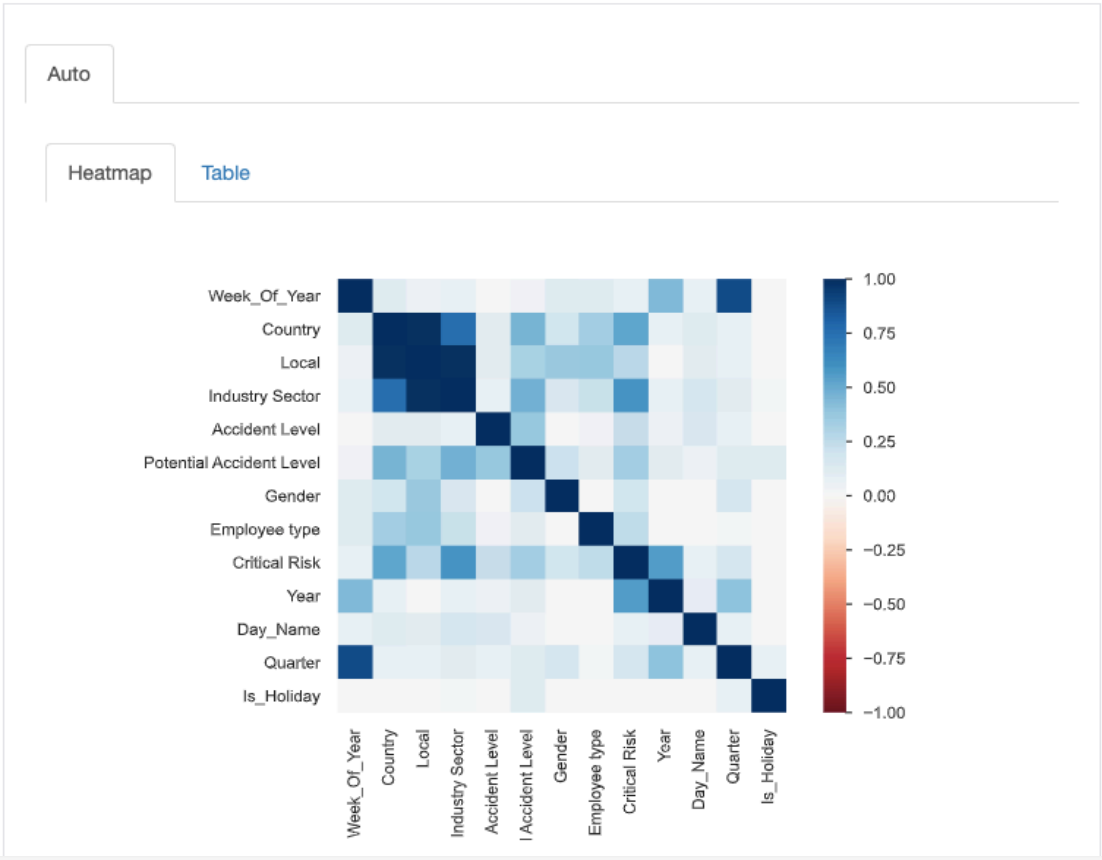
Is_Holiday

Categorical

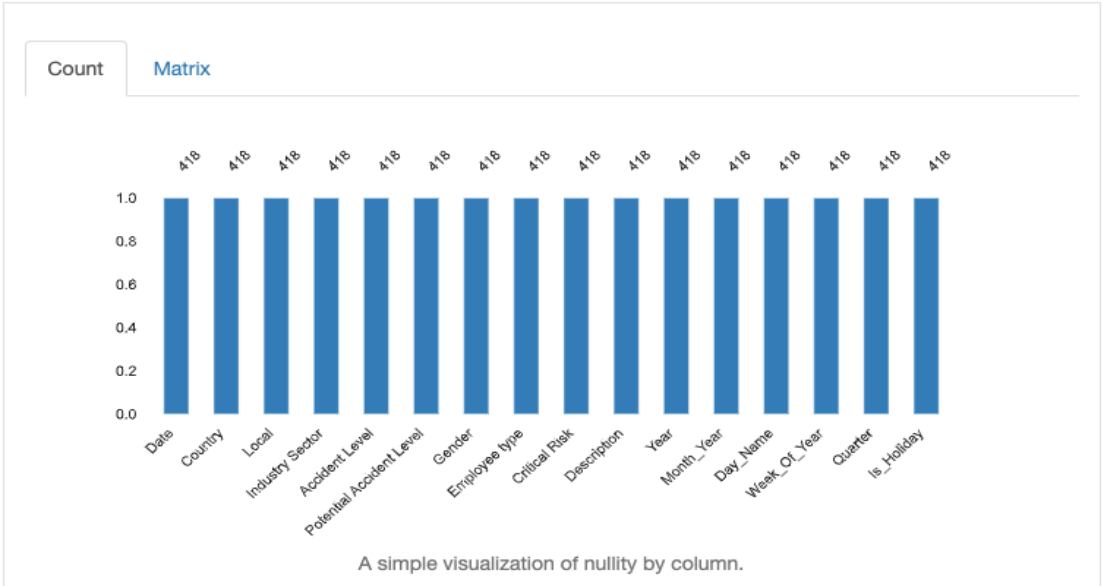
Distinct	2
Distinct (%)	0.5%
Missing	0
Missing (%)	0.0%
Memory size	3.4 KiB

More details

Correlations



Missing values



The displayed results are of higher clarity and can be mostly used as a dashboard inputs for any management level interface.

This has also shown a different viewpoint of many accidents have occurred in 2016 than in 2017. This can be evaluated carefully to understand the improvements taken place in safety practices followed, preventions in place, focus shift from management etc. YoY this needs to be improved to bring to the state of "Zero accidents" level.

Data Preprocessing

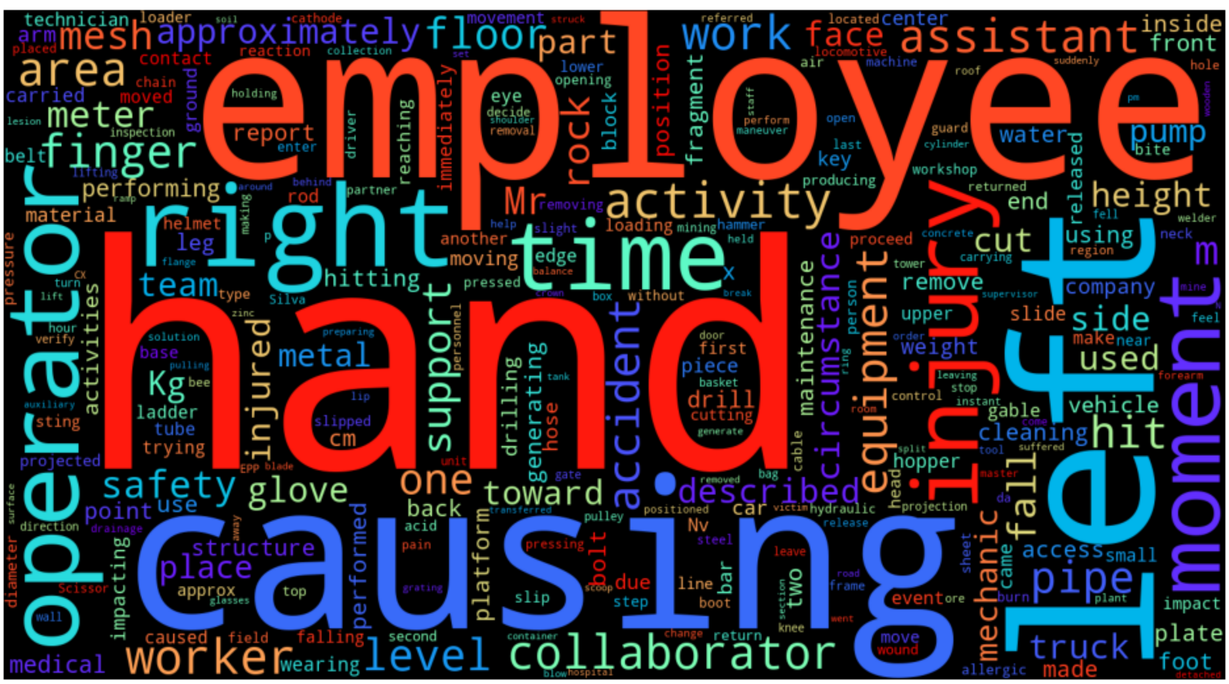
Preprocessing the data removes missing or inconsistent data values resulting from human or Computer error which in turn improve the accuracy, quality of dataset and reliability.

For this purpose, TensorFlow & Nltk Libraries are predominantly used.

This process mostly using the NLP techniques such as Tokenization, Stopwords, Lemmetizer, TfIdf vectorizer etc.

a. First step is to do the Data Description cleansing using Regular Expression and by removing StopWords.
b. Next step is to convert Apostrophe to Standard Lexicons.
c. Than to Lemmatize the description.
d. After lemmatization, the description has minimum of 61 and maximum of 657 characters.

An output of Word Cloud of the description of the Actual dataset.



Following are the NLP pre-processing steps taken before applying model on the data:

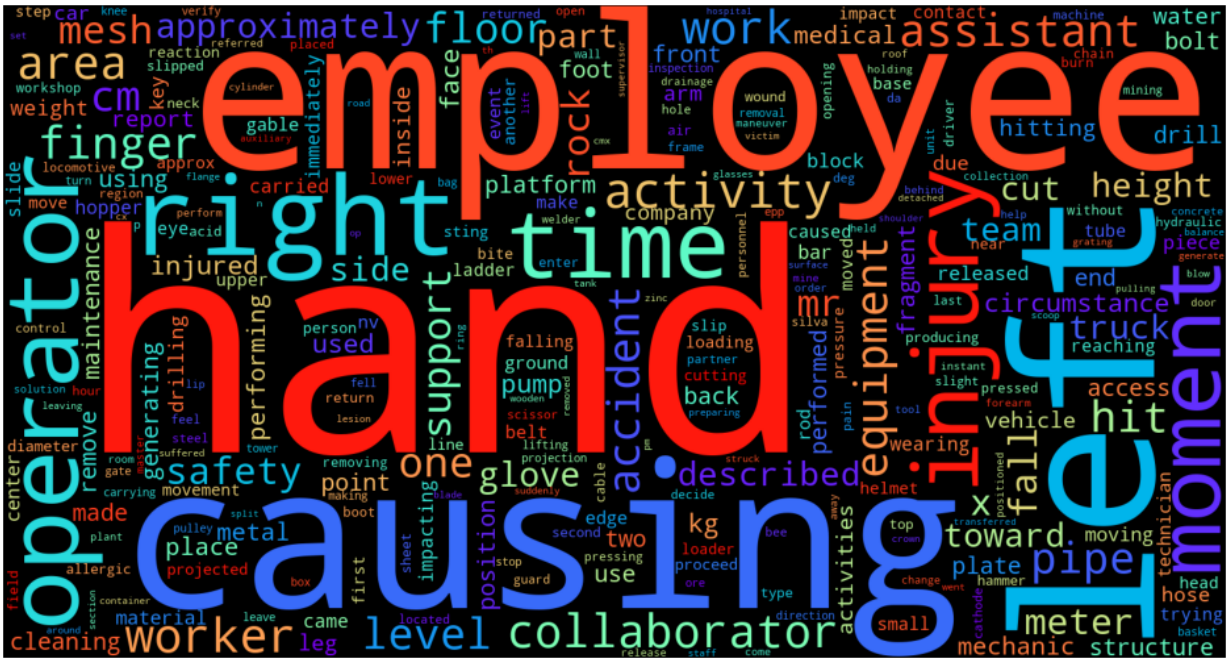
- Converting Accented to ASCII Text
- Removing HTML and URL's
- Fixing Contractions
- Removing Numbers
- Converting to Lower Case
- Removing Punctuations and Whitespaces
- Removing Stop words.

1.1.3.2 Text Pre - Processing

In [71]:

```
# function for text pre-processing
space = ' '
def clean_text(description, punctuations=r'''!()-[]{};:'"\,<>./?@#$%^&*_~'''):
    """
    A method to clean text
    """
    # Replacing the accented words to get the closest possible ASCII text
    string = space.join([unidecode.unidecode(word) for word in description.split()])

    # Cleaning the urls
    string = re.sub(r'https?://\S+|www\.\S+', ' ', string)

    # Cleaning the html elements
    string = re.sub(r'<.*?>', ' ', string)

    #fix contractions (example: "'cause": "because","could've": "could have",etc)
    string = space.join([contractions.fix(word) for word in string.split()])

    #remove number
    string = re.sub(r'\d+', ' ', string)

    # Removing the punctuations using regular expression
    # i.e remove anything which is not word or whitespace character
    string = re.sub(r'[^\w\s]', ' ', string)

    # Converting the text to lower
    string = string.lower()

    # Cleaning the whitespaces
    string = re.sub(r'\s+', ' ', string).strip()

    # Removing stop words
    string = ' '.join([word for word in string.split() if word not in stopwords.words('english')])

    return string

```

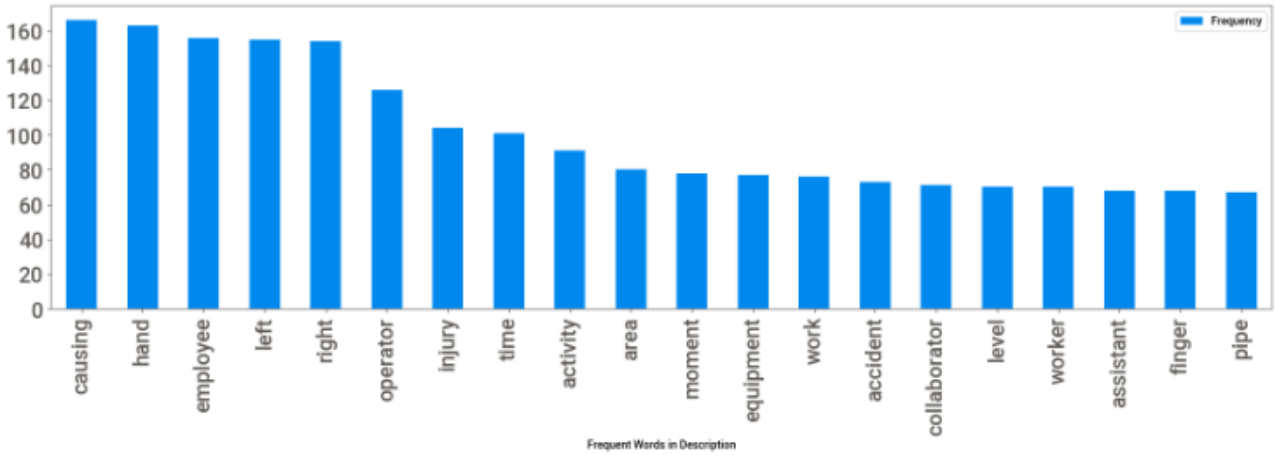
Output from Wordcloud:



Words captured are categorized as below:

i. Body: left, right, back, hand, finger, face, foot and glove
ii. Work force: employee, operator, collaborator, assistant, team, worker and mechanic
iii. Movement: fall, hit, lift and slip
iv. Equipment: pump, meter, drill, truck and tube
v. Accident: accident, activity, cut, moment, safety, injured, injury and cause

Following bar chart shows the distribution frequent words in description:



NLP Pre-processing summary:

1. 74% of data where accident description > 100 is captured in low accident level.
2. 34% of data where accident description > 100 is captured in high medium potential accident level.
3. 25% of data where accident description > 100 is captured in medium potential accident level.
4. 23% of data where accident description > 100 is captured in low potential accident level.
5. Few of the NLP pre-processing steps taken before applying model on the data
6. Converting to lower case, avoid any capital cases
7. Converting apostrophe to the standard lexicons
8. Removing punctuations
9. Lemmatization
10. Removing stop words
11. After pre-processing steps:

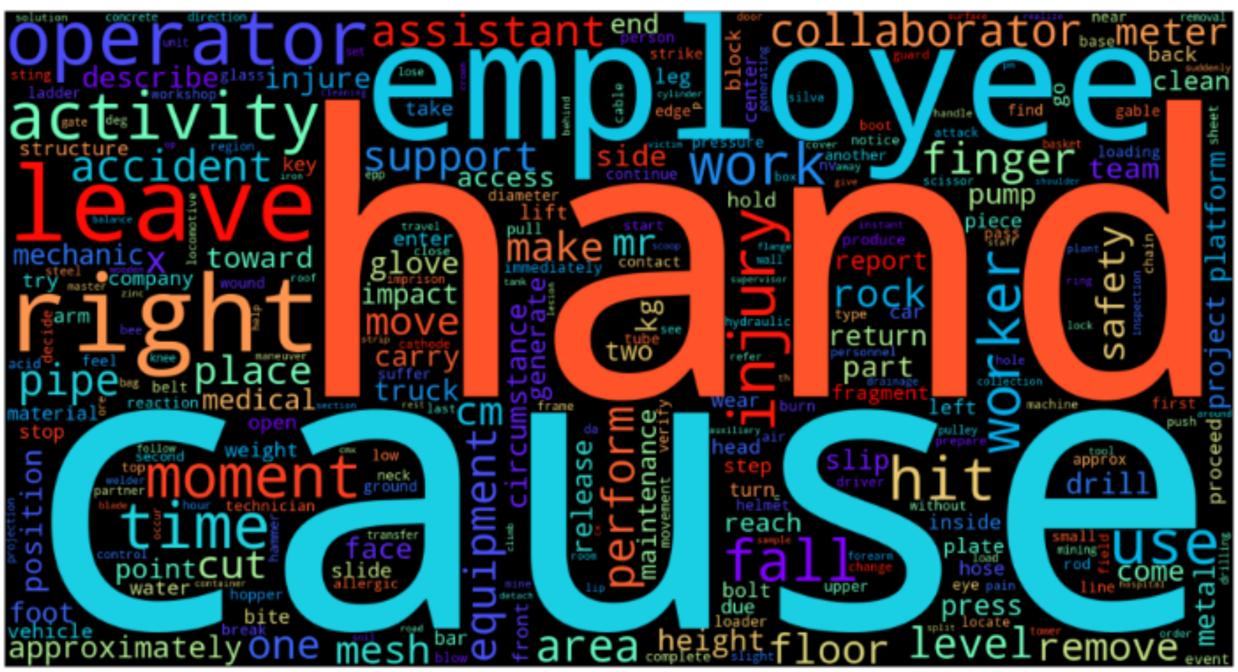
    a. Minimum line length: 61

    b. Maximum line length: 657

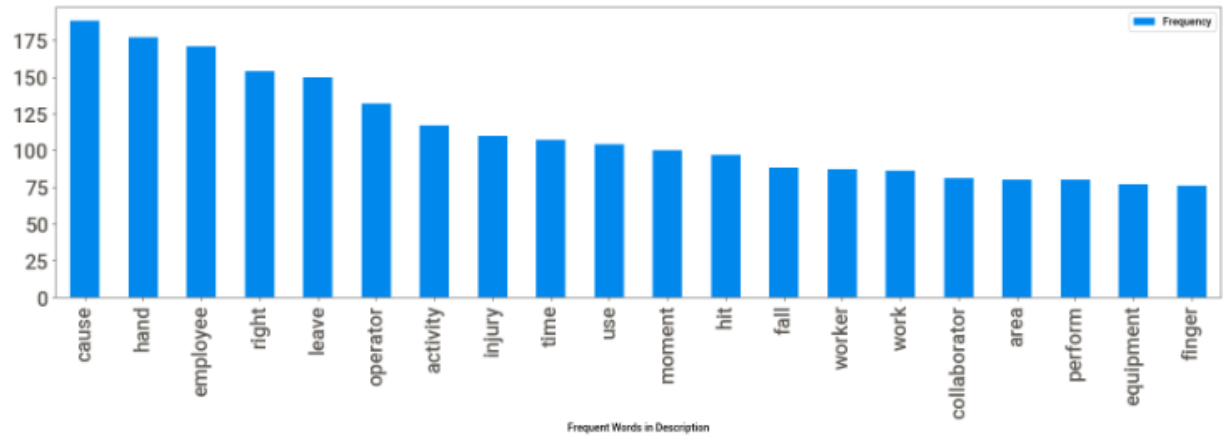
    c. Minimum number of words: 10

    d. Maximum number of words: 98

Lemmatization:

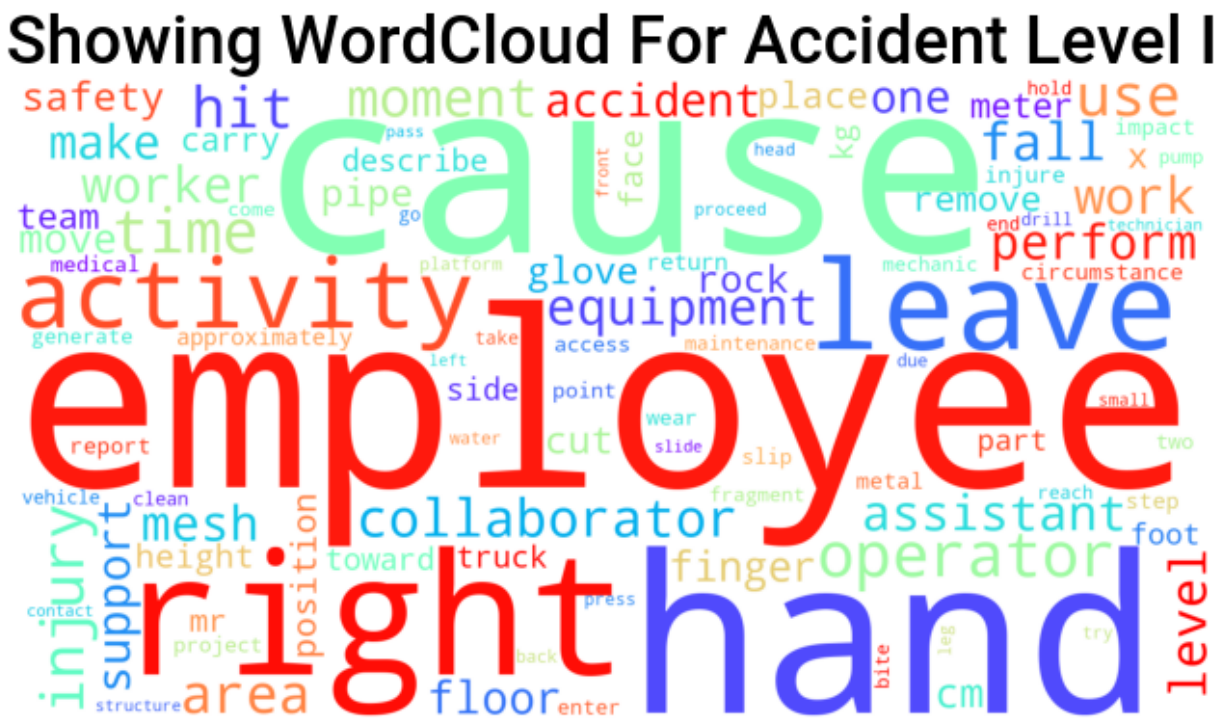


After applying Lemmatization following changes are seen in distribution:



It is now more easier to corelate the words with the incidents and club them into more relevance. So Lemmatization has helped in grouping the words to analyze better.

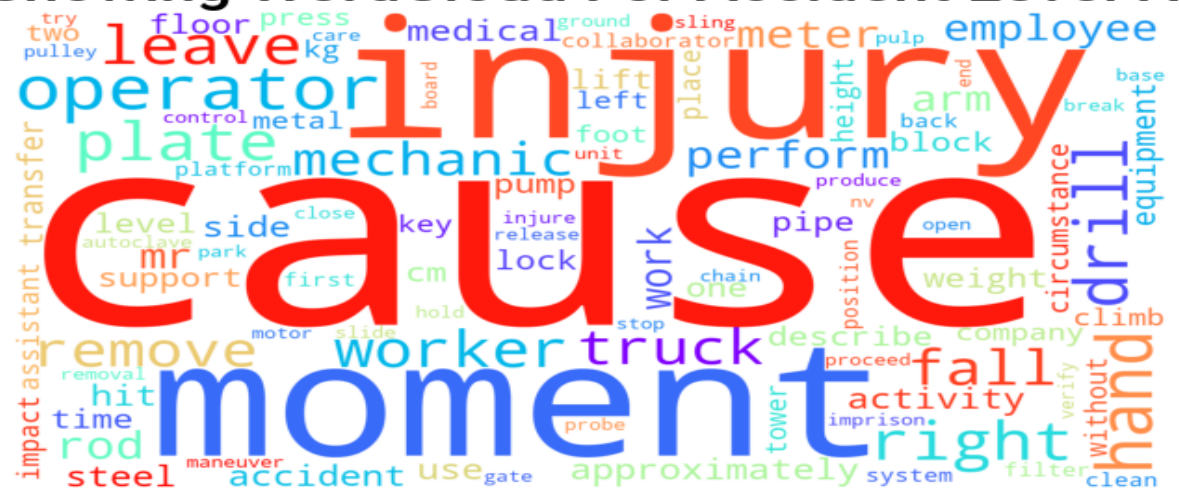
Word Cloud of different Accident Levels



Showing WordCloud For Accident Level II

Showing WordCloud For Accident Level III

Showing WordCloud For Accident Level IV



In the Dataset Descriptions plays vital role as they are key to generate insights and development of Chatbot. So it is essential to analyze the sentences for which are longer. Below shows one such detection:

In [94]:
```
int('--'*45); print('Get the length of each line, find the maximum length and print the maximum length line');
int('Length of line ranges from 64 to 672.');
int('--'*45)

int('Minimum line length: {}\n'.format(df_IndSafety_Final['Cleaned_Description'].str.len().min()))
int('Maximum line length: {}\n'.format(df_IndSafety_Final['Cleaned_Description'].str.len().max()))
int('Line with maximum length: \n{}'.format(df_IndSafety_Final[df_IndSafety_Final['Cleaned_Description'].str.len()
```

```
--------------------------------------------------------------------------------
Get the length of each line, find the maximum length and print the maximum length line
Length of line ranges from 64 to 672.
--------------------------------------------------------------------------------
Minimum line length: 57

Maximum line length: 631

Line with maximum length:
level gallery hold activity bolter equipment operator perform drill first hole support right gable foot deep drill en
d drill rod break leave thread inside drilling machine shank operator assistant decide make two empty percussion atte
mpt free thread shank without success third attempt assistant enter corrugate iron central hole rest bar embed shank
generate pressure moment operator activate percussion generate movement shank hit palm victim leave hand generate des
cribe injury worker wear safety glove time accident end corrugate iron contact leave hand shape like cane worker time
accident position roof support mesh split set
```

Similarly, to identify number of words are also important and results are as below:

```
In [71]:
print('--'*45); print('Get the number of words, find the maximum number of words and print the maximum number of wo
print('Number of words ranges from 10 to 98.');
print('--'*45)

print('Minimum number of words: {}\n'.format(df_merged_final['New_Description'].apply(lambda x: len(x.split(' '))).
print('Maximum number of words: {}\n'.format(df_merged_final['New_Description'].apply(lambda x: len(x.split(' '))).
print('Line with maximum number of words: \n{}'.format(df_merged_final[df_merged_final['New_Description'].apply(lam

executed in 23ms, finished 15:57:38 2023-06-04

------------------------------------------------------------------------------------------
Get the number of words, find the maximum number of words and print the maximum number of words
Number of words ranges from 10 to 98.
------------------------------------------------------------------------------------------
Minimum number of words: 9

Maximum number of words: 95

Line with maximum number of words:
performing sleeve removal maneuver hole meter deep general da silva pressed one side locking nut rod together jack ho
ld entire weight rod maneuver locking procedure effective weight rod secured steel wire rope probe winch moment drill
er pedro released brake winch inefficacy locking done one side chestnut without aid monkey caused sliding rod auxilia
ry prepared manual unlocking rod holding faucet key firmly probe tower composition shifted stem slid hand shifted dow
nward causing left hand strike base probe tower structure causing cut th th quirodactyl employee taken hospital went
medical care wound sutured stitch removed day activity
```

Data Preparation

Good Data Preparation allows for efficient data analysis, limits errors and inaccuracies that occur to data during preprocessing and makes all processed data more accessible to users. This helps us in arriving at a better insight to the analysis and analyzed data.

This mainly involves ***Label Encoding.*** Sklearn library provides a very efficient tool for Encoding the levels of categorical features into numeric values. Label Encoder encode labels with a value between 0 and n_classes-1 where n is the number of distinct classes. If a label repeats it assigns the same value to as assigned earlier.

Final data after encoding looks as follows:

```
df_IndSafety_F_Cpy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 17 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Date                      418 non-null    datetime64[ns]
 1   Country                   418 non-null    int64
 2   Local                     418 non-null    int64
 3   Industry Sector           418 non-null    int64
 4   Accident Level            418 non-null    int64
 5   Potential Accident Level  418 non-null    int64
 6   Gender                    418 non-null    int64
 7   Employee type             418 non-null    int64
 8   Critical Risk             418 non-null    int64
 9   Description               418 non-null    object
 10  Year                      418 non-null    int64
 11  Month_Year                418 non-null    int64
 12  Day_Name                  418 non-null    int64
 13  Week_Of_Year              418 non-null    int64
 14  Quarter                   418 non-null    int64
 15  Is_Holiday                418 non-null    int64
 16  Cleaned_Description       418 non-null    object
dtypes: datetime64[ns](1), int64(14), object(2)
memory usage: 55.6+ KB
```

In [108]:
```
# Printing Head of the Final Dataframe

df_IndSafety_F_Cpy.head()

```

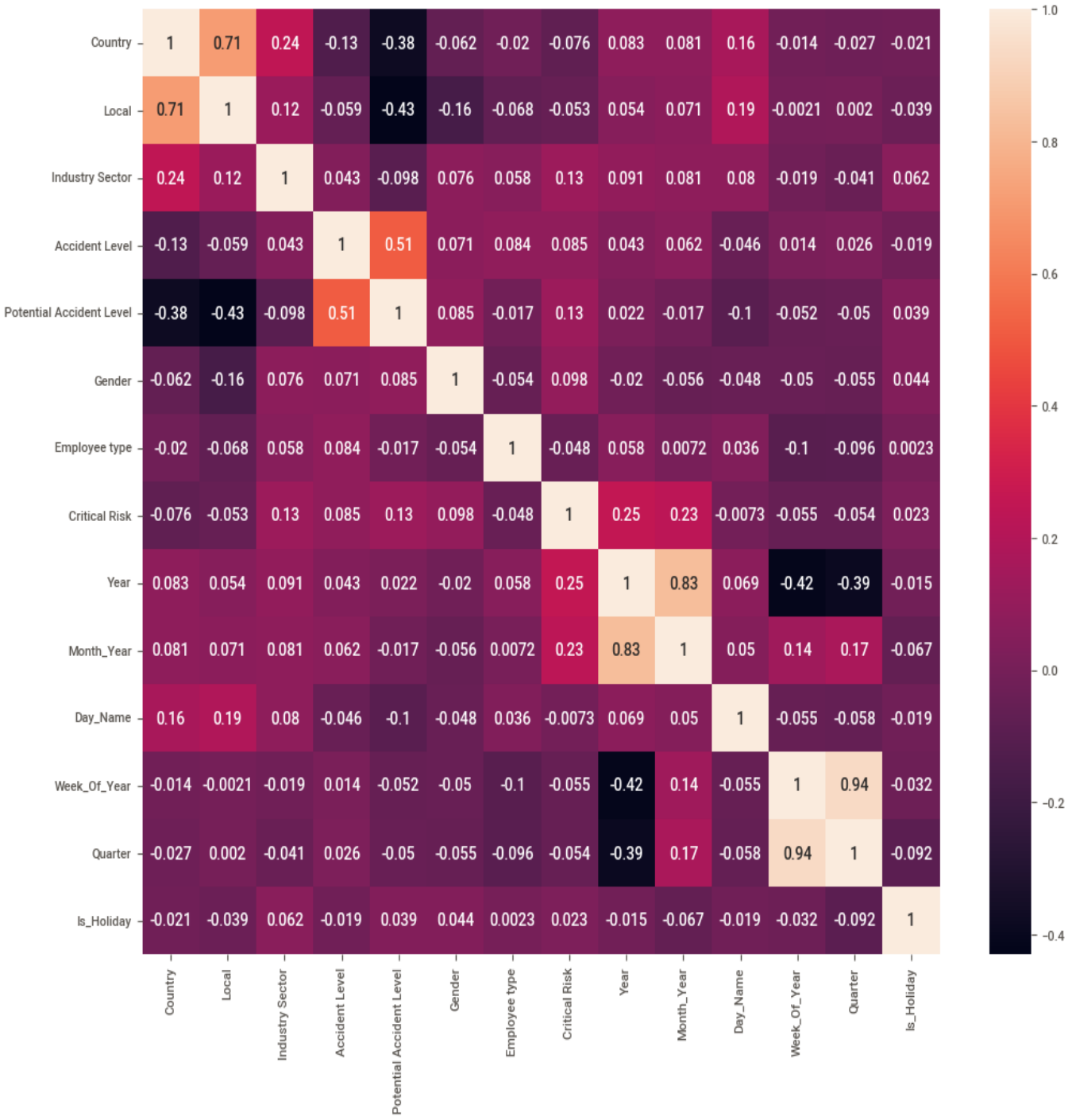
Out[108]:

Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Description	Year	Month_Year	Day_Name	Week_Of_Year	Quarter	Is_Holiday	Cleaned_Description
1	0	3	1	1	20	While removing the drill rod of the Jumbo 08 f...	2016	0	0	53	1	1	remove drill rod jumbo maintenance supervisor ...
1	0	3	1	0	21	During the activation of a sodium sulphide pum...	2016	0	2	53	1	0	activation sodium sulphide pump pipe uncoupled...
1	0	2	1	2	15	In the sub-station MILPO located at level +170...	2016	0	6	1	1	0	sub station milpo locate level collaborator ex...
1	0	0	1	1	16	Being 9:45 am. approximately in the Nv. 1880 C...	2016	0	0	1	1	0	approximately nv cx ob personnel begin task un...
1	3	3	1	1	16	Approximately at 11:45 a.m. in circumstances t...	2016	0	3	1	1	0	approximately circumstance mechanic anthony gr...

Summary:

1. All Columns are Label Encoded except Description and Cleaned Description columns
2. We can see that All Columns have been converted to Numerical Values after Label Encoding.
3. The Description Columns consists of the actual data from the data which we have imported
4. The Cleaned_Description column consists of the cleaned data after applying all the nlp pre processing techniques including lemmatization.
5. So we will be using the Cleaned_Description column for futher Model Building and we apply the nlp techniques to prepare the data to send to the model.
6. The above DataFrame is the final Modifiled dataframe which consists of the final data.
7. From Above We can see that the Target Variable is Accident Level

Heatmap after LabelEncoding:



Dropping of Un-necessary Columns

In [128]:
```
df_IndSafety_F_Cpy_Mdf = df_IndSafety_F_Cpy.drop(['Date','Description','Cleaned_Description','Year',
                                                  'Month_Year','Day_Name','Week_Of_Year',
                                                  'Quarter','Is_Holiday'], axis=1)
```

In [129]:
```
df_IndSafety_F_Cpy_Mdf
```

Out[129]:

	Country	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk
0	0	0	1	0	3	1	1	20
1	1	1	1	0	3	1	0	21
2	0	2	1	0	2	1	2	15
3	0	3	1	0	0	1	1	16
4	0	3	1	3	3	1	1	16
...	...	...	...	...	...	...	...	...
413	0	3	1	0	2	1	1	16
414	0	2	1	0	1	0	0	16
415	1	8	0	0	1	1	0	31
416	1	4	0	0	1	1	0	6
417	0	3	1	0	1	0	1	11

418 rows × 8 columns

Usage of Count Vectorization:

Machines cannot understand characters and words. So when dealing with text data we need to represent it in numbers to be understood by the machine. Countvectorizer is a method to convert text to numerical data.

In [112]:
```
# Creating the Bag of Words model
# setting the max features to 1500
cv = CountVectorizer() #TBD
```

In [113]:
```
# fit and transforming into vectors from the text data
cv_vectors = cv.fit_transform(df_IndSafety_Final['Cleaned_Description']).toarray()
```

In [114]:
```
# Printing the identified Unique words along with their indices
print("Vocabulary: ", cv.vocabulary_)
# Summarizing the Encoded Texts
print("Encoded Document is:")
print(cv_vectors)
```

```
Vocabulary:  {'remove': 1750, 'drill': 637, 'rod': 1811, 'jumbo': 1142, 'maintenance': 1297, 'supervisor': 2119, 'pro
ceed': 1648, 'loosen': 1261, 'support': 2122, 'intermediate': 1100, 'centralizer': 321, 'facilitate': 783, 'removal':
1749, 'see': 1888, 'mechanic': 1347, 'one': 1468, 'end': 699, 'equipment': 722, 'pull': 1677, 'hand': 961, 'bar': 17
0, 'accelerate': 10, 'moment': 1390, 'slide': 1970, 'point': 1598, 'tighten': 2205, 'finger': 825, 'drilling': 639,
'beam': 183, 'activation': 31, 'sodium': 1987, 'sulphide': 2109, 'pump': 1681, 'pipe': 1576, 'uncoupled': 2291, 'sulf
ide': 2105, 'solution': 1995, 'design': 557, 'area': 123, 'reach': 1714, 'maid': 1294, 'immediately': 1044, 'make': 1
298, 'use': 2315, 'emergency': 689, 'shower': 1937, 'direct': 590, 'ambulatory': 87, 'doctor': 625, 'later': 1182, 'h
ospital': 1018, 'note': 1444, 'gram': 936, 'liter': 1237, 'sub': 2091, 'station': 2044, 'milpo': 1368, 'locate': 125
0, 'level': 1213, 'collaborator': 397, 'excavation': 744, 'work': 2425, 'pick': 1566, 'tool': 2225, 'hit': 1000, 'roc
k': 1809, 'flat': 841, 'part': 1521, 'beak': 182, 'bounce': 239, 'steel': 2047, 'tip': 2210, 'safety': 1844, 'shoe':
1926, 'metatarsal': 1359, 'leave': 1201, 'foot': 854, 'cause': 312, 'injury': 1080, 'approximately': 120, 'nv': 1454,
'cx': 512, 'ob': 1456, 'personnel': 1560, 'begin': 192, 'task': 2160, 'unlock': 2306, 'soquet': 1997, 'bolt': 224, 'b
hb': 199, 'machine': 1287, 'penultimate': 1547, 'identify': 1035, 'hexagonal': 991, 'head': 975, 'wear': 2386, 'mr':
1409, 'cristobal': 489, 'auxiliary': 153, 'assistant': 134, 'climb': 377, 'platform': 1592, 'exert': 753, 'pressure':
1636, 'dado': 518, 'key': 1149, 'prevent': 1637, 'come': 404, 'two': 2285, 'rotate': 1825, 'lever': 1214, 'anti': 10
8, 'clockwise': 380, 'direction': 591, 'palm': 1512, 'circumstance': 363, 'anthony': 107, 'group': 949, 'leader': 119
5, 'eduardo': 662, 'eric': 725, 'fernandez': 808, 'injure': 1079, 'three': 2195, 'company': 410, 'impromec': 1052, 'p
erform': 1553, 'pulley': 1678, 'motor': 1403, 'zaf': 2442, 'marcy': 1328, 'cm': 387, 'length': 1208, 'weight': 2391,
'kg': 1151, 'lock': 1252, 'heating': 981, 'fall': 790, 'distance': 612, 'meter': 1360, 'high': 994, 'instep': 1093,
```

Usage of TF-IDF

Term frequency-inverse document frequency is a text vectorizer that transforms the text into a usable vector. It combines 2 concepts, Term Frequency (TF) and Document Frequency (DF).

The term frequency is the number of occurrences of a specific term in a document. Term frequency indicates how important a specific term in a document. Term frequency represents every text from the data as a matrix whose rows are the number of documents and columns are the number of distinct terms throughout all documents.

Document frequency is the number of documents containing a specific term. Document frequency indicates how common the term is.

Inverse document frequency (IDF) is the weight of a term, it aims to reduce the weight of a term if the term's occurrences are scattered throughout all the documents. IDF can be calculated as follow:

$$idf_i = \log(\frac{n}{df_i})$$

Where idf_i is the IDF score for term i, df_i is the number of documents containing term i, and n is the total number of documents. The higher the DF of a term, the lower the IDF for the term. When the number of DF is equal to n which means that the term appears in all documents, the IDF will be zero, since log(1) is zero, when in doubt just put this term in the stopword list because it doesn't provide much information.

The TF-IDF score as the name suggests is just a multiplication of the term frequency matrix with its IDF, it can be calculated as follow:

$$w_{i,j} = tf_{i,j} \times idf_i$$

Where w_{ij} is TF-IDF score for term i in document j, tf_{ij} is term frequency for term i in document j, and idf_i is IDF score for term i.

1.1.4.3 TF-IDF Vectorization

In [115]:
```
# Creating the object to the TfidfVectorizer class

tfidf_v = TfidfVectorizer()
```

In [116]:
```
# fit and transforming into vectors from the text data
tfidf_vectors = tfidf_v.fit_transform(df_IndSafety_F_Cpy['Cleaned_Description']).toarray()

```

In [117]:
```
# get indexing
print('\nWord indexes:')
print(tfidf_v.vocabulary_)

# tf-idf values
print('\ntf-idf values:')
print(tfidf_vectors)

```

```
Word indexes:
{'remove': 1750, 'drill': 637, 'rod': 1811, 'jumbo': 1142, 'maintenance': 1297, 'supervisor': 2119, 'proceed': 1648,
'loosen': 1261, 'support': 2122, 'intermediate': 1100, 'centralizer': 321, 'facilitate': 783, 'removal': 1749, 'see':
1888, 'mechanic': 1347, 'one': 1468, 'end': 699, 'equipment': 722, 'pull': 1677, 'hand': 961, 'bar': 170, 'accelerat
e': 10, 'moment': 1390, 'slide': 1970, 'point': 1598, 'tighten': 2205, 'finger': 825, 'drilling': 639, 'beam': 183,
'activation': 31, 'sodium': 1987, 'sulphide': 2109, 'pump': 1681, 'pipe': 1576, 'uncoupled': 2291, 'sulfide': 2105,
```

Combining the TF-IDF Vectors and Encoded Features for Final Dataframe to train the model.

1.1.4.5 Combine TF-IDF and Encoded Features - Data Frame with TF-IDF

In [138]:
```
# Consider only top 30 GLOVE features
df_IndSafety_EncTFIDF = df_IndSafety_F_Cpy_Mdf.join(df_IndSafety_tfidf.reset_index(drop=True))## Combine Glove and

```

In [139]:
```
df_IndSafety_EncTFIDF
```

Out[139]:

	Country	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	TFIDF_access	TFIDF_accident	...	TFIDF_verify type allergic	TFIDF_wall gable approx	TFIDF_warehouse help forklift
0	0	0	1	0	3	1	1	20	0.0	0.0	...	0.0	0.0	0.0
1	1	1	1	0	3	1	0	21	0.0	0.0	...	0.0	0.0	0.0
2	0	2	1	0	2	1	2	15	0.0	0.0	...	0.0	0.0	0.0
3	0	3	1	0	0	1	1	16	0.0	0.0	...	0.0	0.0	0.0
4	0	3	1	3	3	1	1	16	0.0	0.0	...	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
413	0	3	1	0	2	1	1	16	0.0	0.0	...	0.0	0.0	0.0
414	0	2	1	0	1	0	0	16	0.0	0.0	...	0.0	0.0	0.0
415	1	8	0	0	1	1	0	31	0.0	0.0	...	0.0	0.0	0.0
416	1	4	0	0	1	1	0	6	0.0	0.0	...	0.0	0.0	0.0
417	0	3	1	0	1	0	1	11	0.0	0.0	...	0.0	0.0	0.0

418 rows × 608 columns

In [140]:
```
df_IndSafety_EncTFIDF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Columns: 608 entries, Country to TFIDF_worker wear safety
dtypes: float64(600), int64(8)
memory usage: 1.9 MB
```

Word2Vec Model:

This module implements the word2vec family of algorithms, using highly optimized C routines, data streaming and Pythonic interfaces.

The word2vec algorithms include skip-gram and CBOW models, using either hierarchical softmax or negative sampling

In [144]:
```
# Model creation
model= Word2Vec(words_list, min_count = 1)

```

In [145]:
```
model
```

Out[145]:
```
<gensim.models.word2vec.Word2Vec at 0x7ff4b1b42d70>
```

In [146]:
```
# saving the model
model.save("word2vec.model")
```

In [147]:
```
# Let's have a quick look at the vocabulary words:

words = model.wv.index_to_key
len(words)
```

Out[147]:
```
2465
```

In [149]:
```
w2v_Model = Word2Vec.load('model.bin')
print(w2v_Model)
```

```
Word2Vec<vocab=36, vector_size=100, alpha=0.025>
```

In [150]:
```
# Creating word2vec model with 50 embeddings
embeddings_index = {}
f = open(r"glove.6B.50d.txt",encoding="utf8")
#f = open(r"/content/drive/My Drive/PG in AIML/Capstone Project - Main
for line in tqdm(f):
    values = line.split(' ')
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Found %s word vectors.' % len(embeddings_index))

```

```
400000it [00:07, 52399.33it/s]
```

```
Found 400000 word vectors.
```

In [151]:
```
embeddings_index.get('cause')

```

Out[151]:
```
array([ 1.1322  ,  0.14747 ,  0.16434 , -0.18466 , -0.47143 ,  1.4388  ,
        0.71446 ,  0.78135 ,  0.61864 ,  0.20594 ,  0.43364 ,  0.070101,
        0.052516, -0.3708  ,  0.23736 ,  0.39409 , -0.14853 , -0.60318 ,
       -0.26755 , -0.64166 , -0.62525 , -0.45662 ,  0.42132 , -0.33883 ,
        0.62718 , -1.6358  , -0.89482 , -0.21618 ,  1.0543  ,  0.80585 ,
        3.0055  ,  0.39413 , -0.099487, -1.2947  , -0.71042 , -0.018395,
        0.21888 , -0.7658  ,  0.72606 ,  0.4171  , -0.63542 ,  0.15135 ,
       -0.18722 ,  0.44311 ,  1.1302  ,  0.18293 , -0.40505 ,  0.17468 ,
        0.40155 , -0.092546], dtype=float32)
```

In [152]:
```
# Takinng embedding dimension which will be used later for model training
EMBEDDING_DIM = embeddings_index.get('cause').shape[0]
EMBEDDING_DIM
```

Out[152]:
```
50
```

In [153]:
```
# this function creates a normalized vector for the whole sentence
def sent2vec(s):
    words = word_tokenize(s)
    M = []
    for w in words:
        try:
            M.append(embeddings_index[w])
        except:
            continue
    M = np.array(M)
    v = M.sum(axis=0)
    if type(v) != np.ndarray:
        return np.zeros(300)
    return v / np.sqrt((v ** 2).sum())
```

In [154]:
```
# create sentence GLOVE embeddings vectors using the above function for training and validati
df_IndSafety_Glove = [sent2vec(x) for x in tqdm(df_IndSafety_F_Cpy['Cleaned_Description'])]
```

```
100%|██████████| 418/418 [00:00<00:00, 1987.40it/s]
```

In [155]:
```
df_IndSafety_Glove[0]
```

Out[155]:
```
array([ 0.01731114, -0.01626446,  0.07316305, -0.02552152, -0.0139915 ,
        0.05045992, -0.01475705,  0.03345921,  0.08447054, -0.04745678,
        0.01806958,  0.04117773, -0.03499098,  0.15020484, -0.06887885,
        0.10433374,  0.00155631, -0.0124464 , -0.01654327, -0.20546113,
        0.12687446, -0.0525015 , -0.07336877, -0.09402264,  0.01794343,
       -0.39977798,  0.04642893,  0.03918728,  0.13990197, -0.03209396,
        0.7815898 ,  0.00930543, -0.18064523,  0.01465347,  0.03275797,
        0.04269066,  0.08595435,  0.02138985,  0.05419882, -0.00394803,
       -0.0444368 ,  0.00988354, -0.01003907,  0.06372146,  0.02312261,
       -0.0624687 ,  0.09743617,  0.02269093,  0.00423052,  0.01259812],
      dtype=float32)
```

Final DataFrame of the Word2Vec model Vectors of the DataFrame is as follows:

```
In [156]: 1 pd.DataFrame(df_IndSafety_Glove)
```

Out[156]:

	0	1	2	3	4	5	6	7	8	9	...	40	41	42	43	
0	0.017311	-0.016264	0.073163	-0.025522	-0.013991	0.050460	-0.014757	0.033459	0.084471	-0.047457	...	-0.044437	0.009884	-0.010039	0.063721	0
1	0.204752	0.089949	0.126549	-0.086702	-0.023371	0.166419	0.077482	0.036423	0.078604	0.127686	...	0.094435	-0.020414	-0.018487	0.012895	0
2	0.058569	-0.008832	0.076437	-0.050197	0.047608	0.051086	-0.123550	0.061411	0.111704	-0.030065	...	-0.038093	0.009612	-0.009600	-0.010986	0
3	0.076211	-0.026125	0.100905	-0.014340	0.058835	0.027332	0.030381	0.042071	0.023017	-0.102079	...	0.005486	0.042703	-0.025288	0.052312	0
4	0.012652	0.053714	0.168427	-0.038406	0.061469	0.144642	0.006663	-0.012547	0.047249	-0.061490	...	-0.031674	0.057372	0.017762	0.029730	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
413	0.017752	0.012909	0.115695	-0.070039	0.135316	0.087573	-0.014374	-0.007287	0.016109	-0.159298	...	0.035653	0.006421	-0.037195	-0.022906	-0
414	0.062268	-0.003337	0.042851	-0.038650	0.013218	0.053677	-0.047273	0.010086	0.018432	0.002129	...	-0.044696	0.075782	-0.025473	0.051284	0
415	0.140395	0.001436	0.006149	-0.031282	0.017315	0.012646	-0.079173	0.072888	0.072433	-0.067133	...	-0.093988	0.015227	0.007384	0.087687	0
416	0.014022	0.046171	0.078621	-0.021976	0.054686	0.112269	0.001645	0.044719	0.035551	0.035637	...	0.017272	0.102482	-0.048826	0.049901	0
417	0.048084	0.028716	0.044810	-0.040377	0.058441	-0.062462	-0.075128	0.067221	0.025657	-0.114163	...	-0.026713	0.016289	-0.001786	0.080042	0

418 rows × 50 columns

Combining the Glove Vectors and Encoded Features for Final Dataframe to train the model.

```
In [159]: 1 df_IndSafety_EncGlove
```

Out[159]:

	Country	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	0	1	...	40	41	42	43	44	
0	0	0	1	0	3	1	1	20	0.017311	-0.016264	...	-0.044437	0.009884	-0.010039	0.063721	0.023123	-0
1	1	1	1	0	3	1	0	21	0.204752	0.089949	...	0.094435	-0.020414	-0.018487	0.012895	0.046118	0
2	0	2	1	0	2	1	2	15	0.058569	-0.008832	...	-0.038093	0.009612	-0.009600	-0.010986	0.080454	-0
3	0	3	1	0	0	1	1	16	0.076211	-0.026125	...	0.005486	0.042703	-0.025288	0.052312	0.020192	-0
4	0	3	1	3	3	1	1	16	0.012652	0.053714	...	-0.031674	0.057372	0.017762	0.029730	0.012007	-0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
413	0	3	1	0	2	1	1	16	0.017752	0.012909	...	0.035653	0.006421	-0.037195	-0.022906	-0.123638	-0
414	0	2	1	0	1	0	0	16	0.062268	-0.003337	...	-0.044696	0.075782	-0.025473	0.051284	0.092618	-0
415	1	8	0	0	1	1	0	31	0.140395	0.001436	...	-0.093988	0.015227	0.007384	0.087687	0.118107	-0
416	1	4	0	0	1	1	0	6	0.014022	0.046171	...	0.017272	0.102482	-0.048826	0.049901	0.061838	-0
417	0	3	1	0	1	0	1	11	0.048084	0.028716	...	-0.026713	0.016289	-0.001786	0.080042	0.037904	0

418 rows × 58 columns

```
In [161]: 1 df_IndSafety_EncGlove.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 58 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Country                   418 non-null    int64
 1   Local                     418 non-null    int64
 2   Industry Sector           418 non-null    int64
 3   Accident Level            418 non-null    int64
 4   Potential Accident Level  418 non-null    int64
 5   Gender                    418 non-null    int64
 6   Employee type             418 non-null    int64
 7   Critical Risk             418 non-null    int64
 8   0                         418 non-null    float32
.....
 52  44                        418 non-null    float32
 53  45                        418 non-null    float32
 54  46                        418 non-null    float32
 55  47                        418 non-null    float32
 56  48                        418 non-null    float32
 57  49                        418 non-null    float32
dtypes: float32(50), int64(8)
memory usage: 107.9 KB
```

Above the info of the final Glove Encoded Dataframe to trian the model.

Design Train & Test basic ML Classifiers

The Train- Test split is a technique for evaluating the performance of a ML algorithm. It Can be used for Classification or Regression problems and can be used for any Supervised Learning algorithm. It involves taking a dataset and dividing into two subsets. Goal is to produce the Best Fit model that generalizes well to any New & Unknown Data. Fitted model is evaluated using new examples from the held out validated & tested datasets. Thus, improving the model's accuracy in classifying the new data.

Working with TF-IDF Original Data

Creating X,Y and Train and Test Sets of TF-IDF Data

In [162]:
```
X = df_IndSafety_EncTFIDF.drop(['Accident Level','Potential Accident Level'], axis = 1) # Considering all Predictor
Y = df_IndSafety_EncTFIDF['Accident Level']

```

In [163]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state = 1, stratify = Y)

```

In [164]:
```
X_train, X_test, Y_train_dummy, Y_test_dummy = train_test_split(X, dummy_y, test_size = 0.20, random_state = 1, str

```

In [165]:
```
print('X_train shape : ({0},{1})'.format(X_train.shape[0], X_train.shape[1]))
print('Y_train shape : ({0},)'.format(Y_train.shape[0]))
print('X_test shape : ({0},{1})'.format(X_test.shape[0], X_test.shape[1]))
print('Y_test shape : ({0},)'.format(Y_test.shape[0]))
```

```
X_train shape : (334,606)
Y_train shape : (334,)
X_test shape : (84,606)
Y_test shape : (84,)
```

Resampling Techniques — Oversample minority class - TF-IDF DataFrame

In [169]:
```
# Display new accident level counts
X_upsampled['Accident Level'].value_counts()

```

Out[169]:
```
0    247
1    247
2    247
3    247
4    247
Name: Accident Level, dtype: int64
```

In [170]:
```
# Separate input features and target
X_train_up = X_upsampled.drop(['Accident Level'], axis = 1) # Considering all Predictors
Y_train_up = X_upsampled['Accident Level']

```

SMOTE - Generate synthetic samples - upsample smaller class - TFIDF DataFrame

In [173]:
```
sm = SMOTE(random_state=1)
X_train_smote, Y_train_smote = sm.fit_resample(X_train, Y_train)

```

In [174]:
```
X_train_smote
```

Out[174]:

	Country	Local	Industry Sector	Gender	Employee type	Critical Risk	TFIDF_access	TFIDF_accident	TFIDF_acid	TFIDF_activity	...	TFIDF_verify type allergic	TFIDF_wall gable approx	TFIDF_ware help t
0	1	4	0	1	2	15	0.000000	0.000000	0.0	0.000000	...	0.0	0.0	
1	1	7	0	1	0	16	0.000000	0.000000	0.0	0.183366	...	0.0	0.0	
2	1	4	0	1	0	11	0.000000	0.000000	0.0	0.000000	...	0.0	0.0	
3	0	2	1	1	1	16	0.000000	0.416389	0.0	0.000000	...	0.0	0.0	
4	0	2	1	1	1	32	0.000000	0.000000	0.0	0.000000	...	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1230	0	2	1	1	1	29	0.000000	0.000000	0.0	0.000000	...	0.0	0.0	
1231	0	2	1	1	1	9	0.000000	0.136072	0.0	0.014193	...	0.0	0.0	
1232	0	2	1	1	1	17	0.000000	0.091300	0.0	0.077885	...	0.0	0.0	
1233	0	2	1	1	1	15	0.198220	0.315500	0.0	0.133144	...	0.0	0.0	
1234	0	2	1	1	1	13	0.129458	0.252022	0.0	0.086957	...	0.0	0.0	

1235 rows × 606 columns

In [181]:
```
# Display new accident level counts
Y_train_smote['Accident Level'].value_counts()
```

Out[181]:
```
1    247
0    247
3    247
2    247
4    247
Name: Accident Level, dtype: int64
```

In [182]:
```
# convert integers to dummy variables (i.e. one hot encoded)
Y_train_smote_dummy = to_categorical(Y_train_smote['Accident Level'])
Y_train_smote_dummy

```

Out[182]:
```
array([[0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       ...,
       [0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1.]], dtype=float32)
```

Machine Learning has various Classification models which are run and checked the level of accuracy.

- Logistic Regression
- Ridge Classifier
- K Neighbour Classifier
- Support Vector Classifier (SVC)
- Decision Tree Classifier
- Random Forest Classifier

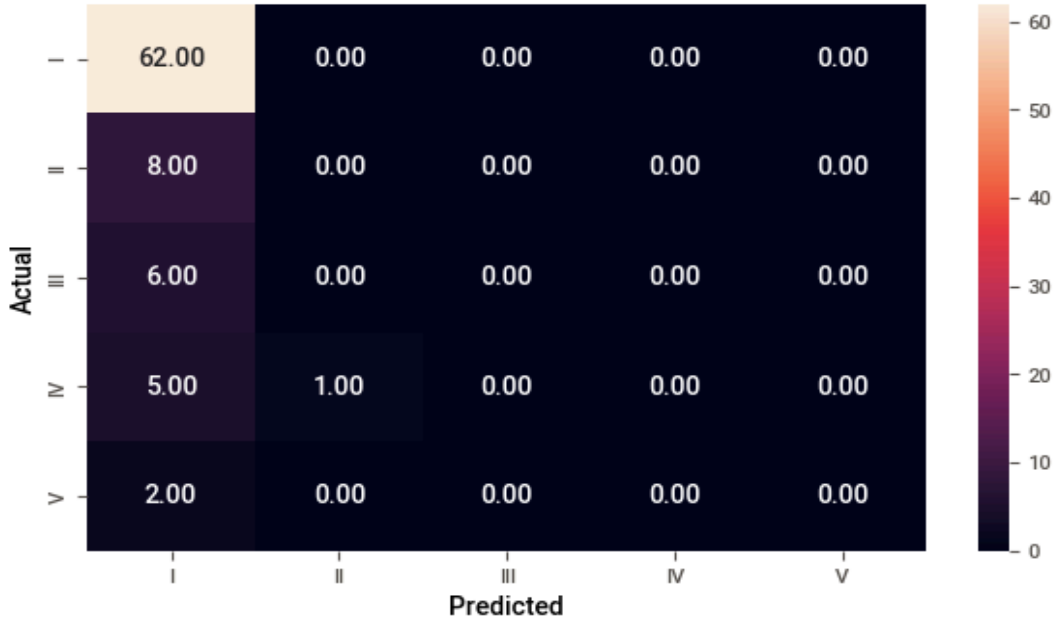
- Bagging Classifier
- Extra Tree Classifier
- ADA Boosting Classifier
- Gradient Boosting Classifier

Once we train the ML model with various Classifiers, following Confusion Matrix can be plotted with given Dataset. Following are the results from them:

Design Train and Test basic ML Classifiers on TF-IDF Original Data:

In this process we use different classifiers and compare the accuracy classes. Models are Trained and Tested and results are as follows.

```
LogisticRegression(multi_class='multinomial', random_state=1)
************************************************************************
```




```
Classification Report -  LogisticRegression - Original_TFIDF_Data
              precision    recall  f1-score   support

           0       0.75      1.00      0.86        62
           1       0.00      0.00      0.00         8
           2       0.00      0.00      0.00         6
           3       0.00      0.00      0.00         6
           4       0.00      0.00      0.00         2

    accuracy                           0.74        84
   macro avg       0.15      0.20      0.17        84
weighted avg       0.55      0.74      0.63        84
```

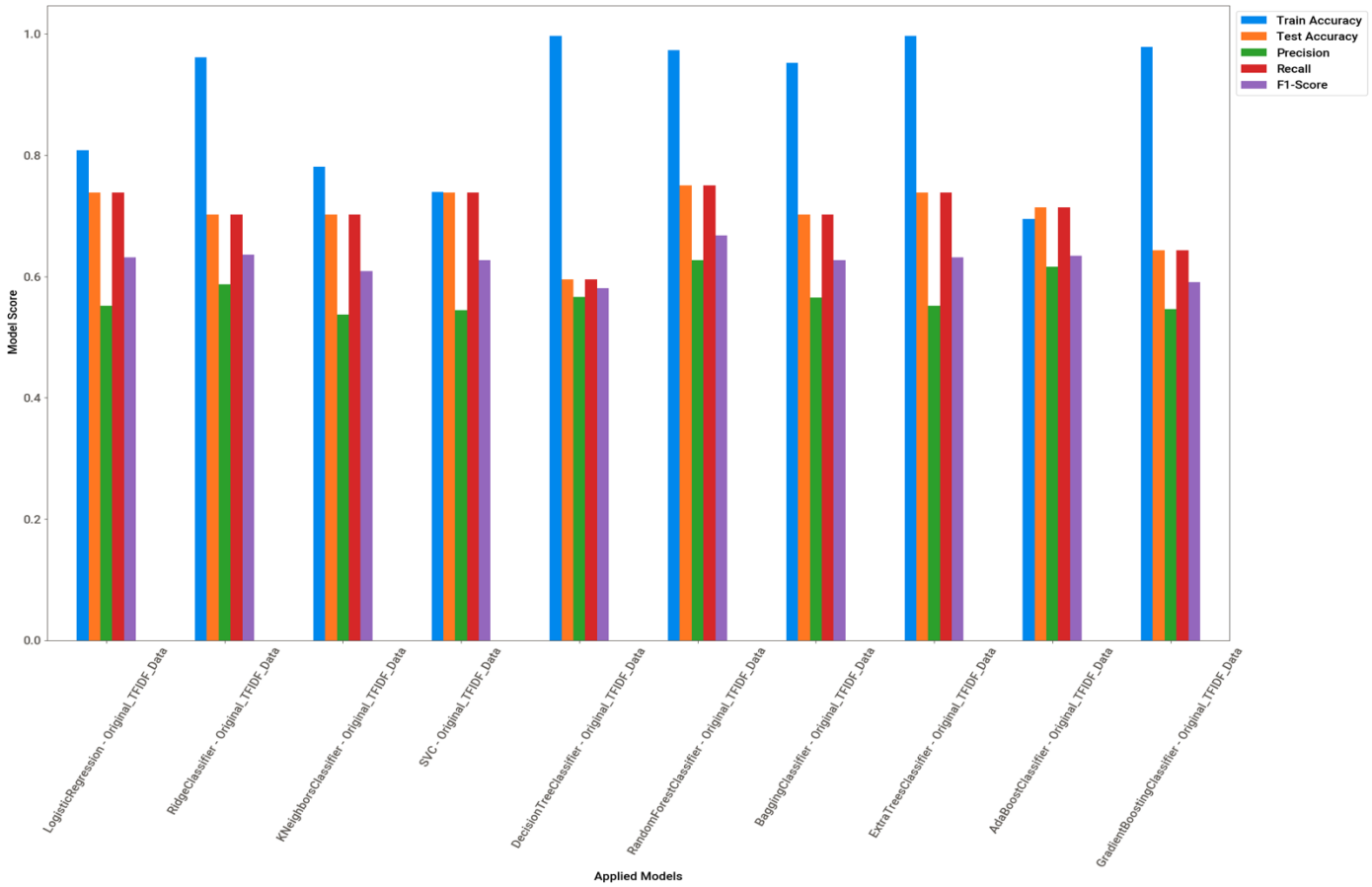
Above is the Output of Logistic Regression. Likewise the overall performace of the models of Original Data Frame is as follows.

```
In [193]: 1 results_normal_Data_Df
```

Out[193]:

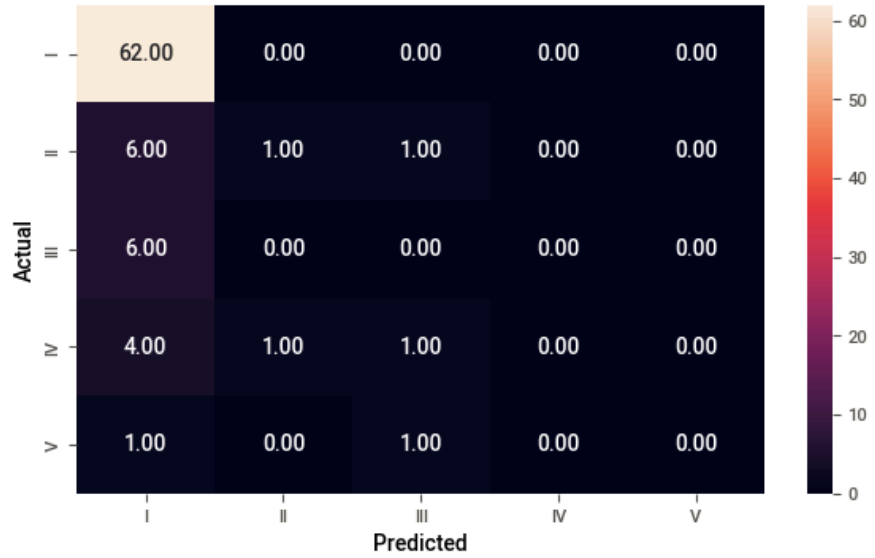
	Method	Train Accuracy	Test Accuracy	Precision	Recall	F1-Score
0	LogisticRegression - Original_TFIDF_Data	0.808383	0.738095	0.551348	0.738095	0.631199
1	RidgeClassifier - Original_TFIDF_Data	0.961078	0.702381	0.587093	0.702381	0.636301
2	KNeighborsClassifier - Original_TFIDF_Data	0.781437	0.702381	0.537625	0.702381	0.609058
3	SVC - Original_TFIDF_Data	0.739521	0.738095	0.544785	0.738095	0.626875
4	DecisionTreeClassifier - Original_TFIDF_Data	0.997006	0.595238	0.566614	0.595238	0.580543
5	RandomForestClassifier - Original_TFIDF_Data	0.973054	0.750000	0.626884	0.750000	0.668153
6	BaggingClassifier - Original_TFIDF_Data	0.952096	0.702381	0.565553	0.702381	0.626584
7	ExtraTreesClassifier - Original_TFIDF_Data	0.997006	0.738095	0.551348	0.738095	0.631199
8	AdaBoostClassifier - Original_TFIDF_Data	0.694611	0.714286	0.615774	0.714286	0.633755
9	GradientBoostingClassifier - Original_TFIDF_Data	0.979042	0.642857	0.545988	0.642857	0.590476

Plotting the Score of all Classification Models is as follows.



From above we can see that Random Forest Classifier is the best perfomed model. Following is the Confusion Matrix and Classification report of the Model.

```
RandomForestClassifier(n_estimators=10, random_state=1)
**************************************************************************
```



Classification Report - RandomForestClassifier - Original_TFIDF_Data

	precision	recall	f1-score	support
0	0.78	1.00	0.88	62
1	0.50	0.12	0.20	8
2	0.00	0.00	0.00	6
3	0.00	0.00	0.00	6
4	0.00	0.00	0.00	2
accuracy			0.75	84
macro avg	0.26	0.23	0.22	84
weighted avg	0.63	0.75	0.67	84

Design Train and Test basic ML Classifiers on TF-IDF Up Sample Data:

In this process we use different classifiers and compare the accuracy classes. Models are Trained and Tested and results are as follows. Let's see overall Scores of the Models which we have trained the models using the DataFrame which is upsampled using traditional resampling method.

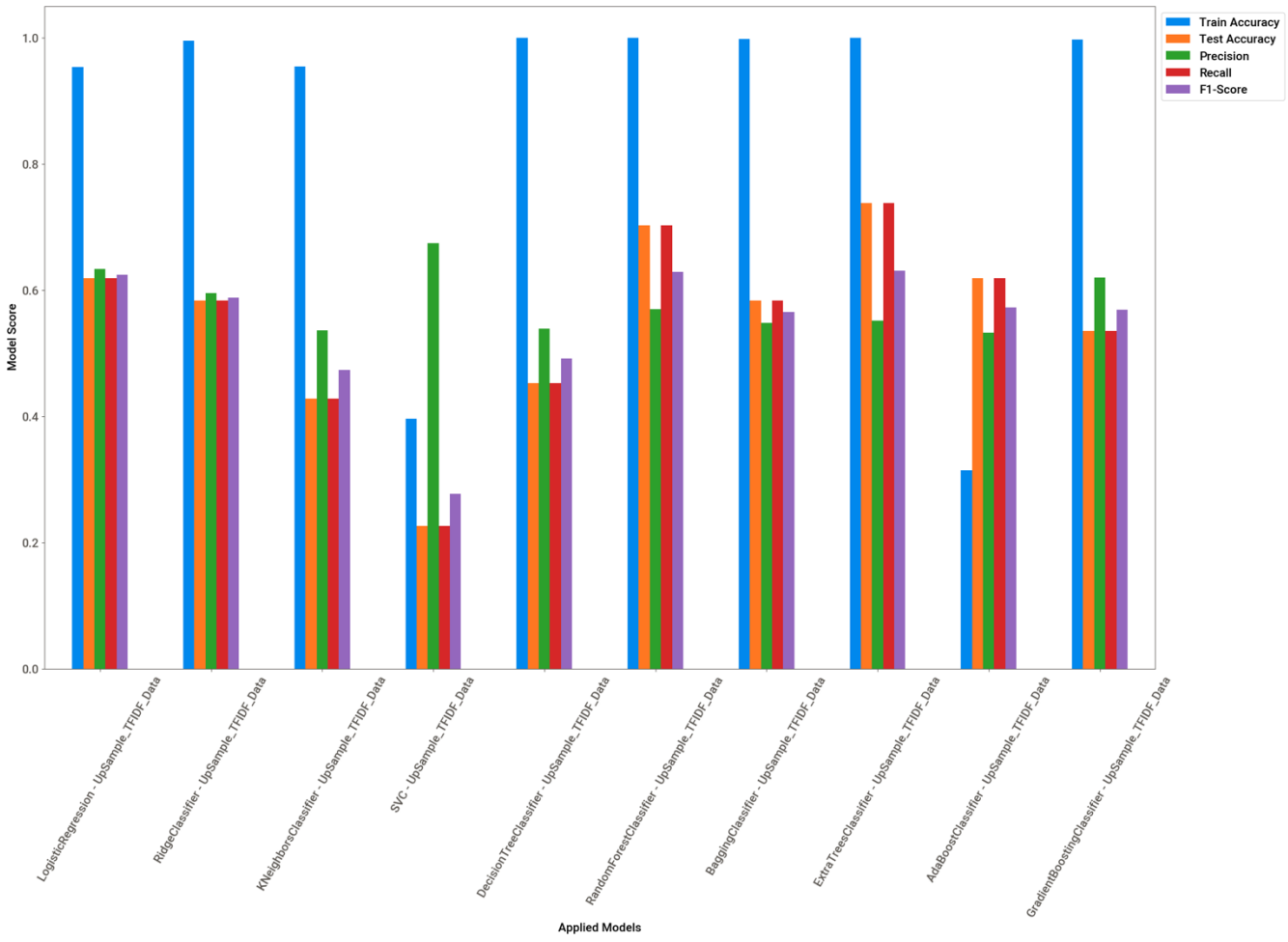
Following are the scores of the model.

In [197]:
```
1 results_upsample_Data_Df
2
```

Out[197]:

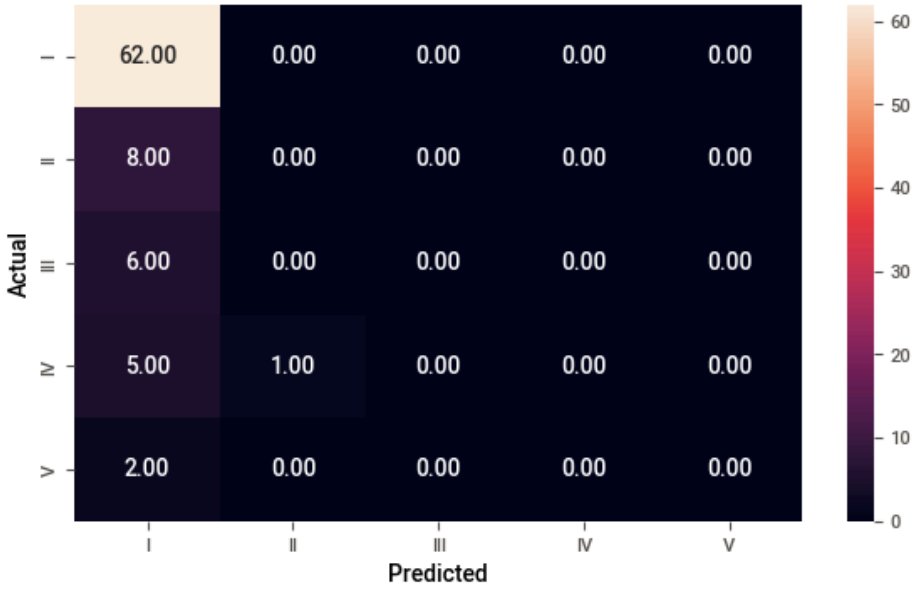
	Method	Train Accuracy	Test Accuracy	Precision	Recall	F1-Score
0	LogisticRegression - UpSample_TFIDF_Data	0.953036	0.619048	0.633769	0.619048	0.624896
1	RidgeClassifier - UpSample_TFIDF_Data	0.995142	0.583333	0.595250	0.583333	0.588304
2	KNeighborsClassifier - UpSample_TFIDF_Data	0.953846	0.428571	0.536637	0.428571	0.473751
3	SVC - UpSample_TFIDF_Data	0.396761	0.226190	0.674825	0.226190	0.277904
4	DecisionTreeClassifier - UpSample_TFIDF_Data	0.999190	0.452381	0.539377	0.452381	0.492063
5	RandomForestClassifier - UpSample_TFIDF_Data	0.999190	0.702381	0.569573	0.702381	0.628663
6	BaggingClassifier - UpSample_TFIDF_Data	0.997571	0.583333	0.547980	0.583333	0.565104
7	ExtraTreesClassifier - UpSample_TFIDF_Data	0.999190	0.738095	0.551348	0.738095	0.631199
8	AdaBoostClassifier - UpSample_TFIDF_Data	0.314980	0.619048	0.533069	0.619048	0.572850
9	GradientBoostingClassifier - UpSample_TFIDF_Data	0.996761	0.535714	0.619561	0.535714	0.569382

Plotting the Score of all Classification Models is as follows



From above we can see that Extra Tree Classifier is the best perfomed model. Following is the Confusion Matrix and Classification report of the Model.

```
ExtraTreesClassifier(bootstrap=True, criterion='entropy', max_features='auto',
                     n_estimators=50, oob_score=True)
**********************************************************************
```




```
Classification Report -  ExtraTreesClassifier - UpSample_TFIDF_Data
              precision    recall  f1-score   support

           0       0.75      1.00      0.86        62
           1       0.00      0.00      0.00         8
           2       0.00      0.00      0.00         6
           3       0.00      0.00      0.00         6
           4       0.00      0.00      0.00         2

    accuracy                           0.74        84
   macro avg       0.15      0.20      0.17        84
weighted avg       0.55      0.74      0.63        84
```

Design Train and Test basic ML Classifiers on TF-IDF Up Sampled using SMOTE Technique:

In this process we use different classifiers and compare the accuracy classes. Models are Trained and Tested and results are as follows. Let's see overall Scores of the Models which we have trained the models using the DataFrame which is upsampled using SMOTE resampling method.

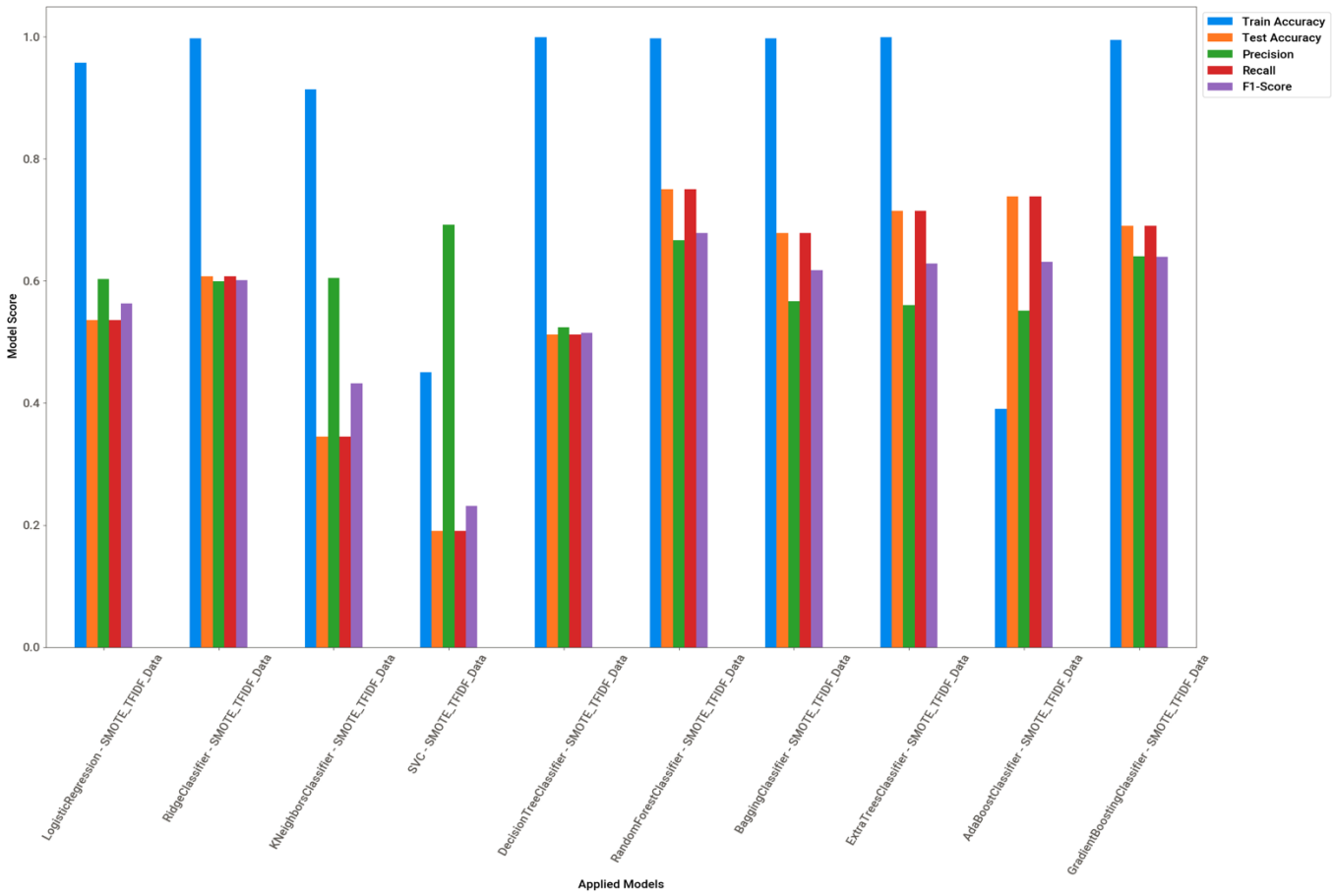
Following are the scores of the model.

```
In [201]:
1 results_smote_Data_Df
2
```

Out[201]:

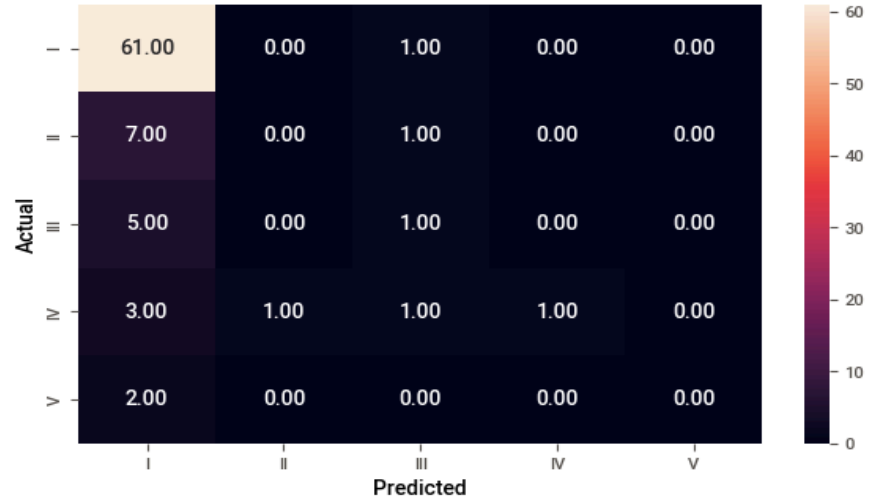
	Method	Train Accuracy	Test Accuracy	Precision	Recall	F1-Score
0	LogisticRegression - SMOTE_TFIDF_Data	0.957085	0.535714	0.603046	0.535714	0.562940
1	RidgeClassifier - SMOTE_TFIDF_Data	0.996761	0.607143	0.599206	0.607143	0.600907
2	KNeighborsClassifier - SMOTE_TFIDF_Data	0.913360	0.345238	0.604729	0.345238	0.432128
3	SVC - SMOTE_TFIDF_Data	0.450202	0.190476	0.691895	0.190476	0.231604
4	DecisionTreeClassifier - SMOTE_TFIDF_Data	0.999190	0.511905	0.523903	0.511905	0.515234
5	RandomForestClassifier - SMOTE_TFIDF_Data	0.996761	0.750000	0.666514	0.750000	0.677891
6	BaggingClassifier - SMOTE_TFIDF_Data	0.996761	0.678571	0.566984	0.678571	0.617012
7	ExtraTreesClassifier - SMOTE_TFIDF_Data	0.999190	0.714286	0.560579	0.714286	0.628166
8	AdaBoostClassifier - SMOTE_TFIDF_Data	0.390283	0.738095	0.551348	0.738095	0.631199
9	GradientBoostingClassifier - SMOTE_TFIDF_Data	0.994332	0.690476	0.639961	0.690476	0.639106

Plotting the Score of all Classification Models is as follows



From above we can see that Random Forest Classifier is the best perfomed model. Following is the Confusion Matrix and Classification report of the Model.

```
RandomForestClassifier(n_estimators=10, random_state=1)
**********************************************************************
```




```
Classification Report -  RandomForestClassifier - SMOTE_TFIDF_Data
              precision    recall  f1-score   support

           0       0.78      0.98      0.87        62
           1       0.00      0.00      0.00         8
           2       0.25      0.17      0.20         6
           3       1.00      0.17      0.29         6
           4       0.00      0.00      0.00         2

    accuracy                           0.75        84
   macro avg       0.41      0.26      0.27        84
weighted avg       0.67      0.75      0.68        84
```

Top 10 Models performed best on Various kinds of data sorted based on F1-Score is as follows:

In [204]:
```
results_Final_Df.sort_values(ascending=False,by='F1-Score')
```

Out[204]:

	Method	Train Accuracy	Test Accuracy	Precision	Recall	F1-Score
25	RandomForestClassifier - SMOTE_TFIDF_Data	0.996761	0.750000	0.666514	0.750000	0.677891
5	RandomForestClassifier - Original_TFIDF_Data	0.973054	0.750000	0.626884	0.750000	0.668153
29	GradientBoostingClassifier - SMOTE_TFIDF_Data	0.994332	0.690476	0.639961	0.690476	0.639106
1	RidgeClassifier - Original_TFIDF_Data	0.961078	0.702381	0.587093	0.702381	0.636301
8	AdaBoostClassifier - Original_TFIDF_Data	0.694611	0.714286	0.615774	0.714286	0.633755
28	AdaBoostClassifier - SMOTE_TFIDF_Data	0.390283	0.738095	0.551348	0.738095	0.631199
17	ExtraTreesClassifier - UpSample_TFIDF_Data	0.999190	0.738095	0.551348	0.738095	0.631199
0	LogisticRegression - Original_TFIDF_Data	0.808383	0.738095	0.551348	0.738095	0.631199
7	ExtraTreesClassifier - Original_TFIDF_Data	0.997006	0.738095	0.551348	0.738095	0.631199
15	RandomForestClassifier - UpSample_TFIDF_Data	0.999190	0.702381	0.569573	0.702381	0.628663

Working with Glove Original Data

Creating X,Y and Train and Test Sets of TF-IDF Data

In [206]:
```
X_glove = df_IndSafety_EncGlove.drop(['Accident Level','Potential Accident Level'], axis = 1) # Considering all Pre
Y_glove = df_IndSafety_EncGlove['Accident Level']

```

In [207]:
```
X_train_glove, X_test_glove, Y_train_glove, Y_test_glove = train_test_split(X_glove, Y_glove,
                                                                            test_size = 0.20, random_state = 1,
                                                                            stratify = Y_glove)

```

In [208]:
```
X_train_glove, X_test_glove, Y_train_dummy_glove, Y_test_dummy_glove = train_test_split(X_glove, dummy_y,
                                                                                        test_size = 0.20,
                                                                                        random_state = 1,
                                                                                        stratify = Y_glove)

```

In [209]:
```
print('X_train shape : ({0},{1})'.format(X_train_glove.shape[0], X_train_glove.shape[1]))
print('Y_train shape : ({0},)'.format(Y_train_glove.shape[0]))
print('X_test shape : ({0},{1})'.format(X_test_glove.shape[0], X_test_glove.shape[1]))
print('Y_test shape : ({0},)'.format(Y_test_glove.shape[0]))

```

```
X_train shape : (334,56)
Y_train shape : (334,)
X_test shape : (84,56)
Y_test shape : (84,)
```

Resampling Techniques — Oversample minority class - TF-IDF DataFrame

In [212]:
```
# Combine majority class with upsampled minority classes
X_upsampled = pd.concat([acclevel_1_majority, acclevel_2_minority_upsampled, acclevel_3_minority_upsampled,
                         acclevel_4_minority_upsampled, acclevel_5_minority_upsampled])

```

In [213]:
```
# Display new accident level counts
X_upsampled['Accident Level'].value_counts()

```

Out[213]:
```
0    247
1    247
2    247
3    247
4    247
Name: Accident Level, dtype: int64
```

In [214]:
```
# Separate input features and target
X_train_up_glove = X_upsampled.drop(['Accident Level'], axis = 1) # Considering all Predictors
Y_train_up_glove = X_upsampled['Accident Level']

```

SMOTE - Generate synthetic samples - upsample smaller class - TFIDF DataFrame

In [218]:
```
sm = SMOTE(random_state=1)
X_train_smote_glove, Y_train_smote_glove = sm.fit_resample(X_train_glove, Y_train_glove)

```

In [219]:
```
X_train_smote_glove

```

Out[219]:

	Country	Local	Industry Sector	Gender	Employee type	Critical Risk	0	1	2	3	...	40	41	42	43	44
0	1	4	0	1	2	15	0.003922	-0.001373	0.095306	-0.087973	...	0.024827	0.083784	-0.009529	0.097148	-0.013241

```
In [226]:
# Display new accident level counts
Y_train_smote_glove['Accident Level'].value_counts()

Out[226]: 1    247
          0    247
          3    247
          2    247
          4    247
          Name: Accident Level, dtype: int64
```

```
In [227]:
# convert integers to dummy variables (i.e. one hot encoded)
Y_train_smote_dummy_glove = to_categorical(Y_train_smote_glove['Accident Level'])
Y_train_smote_dummy_glove

Out[227]: array([[0., 1., 0., 0., 0.],
                 [0., 1., 0., 0., 0.],
                 [1., 0., 0., 0., 0.],
                 ...,
                 [0., 0., 0., 0., 1.],
                 [0., 0., 0., 0., 1.],
                 [0., 0., 0., 0., 1.]], dtype=float32)
```

Machine Learning has various Classification models which are run and checked the level of accuracy.

- Logistic Regression
- Ridge Classifier
- K Neighbour Classifier
- Support Vector Classifier (SVC)
- Decision Tree Classifier
- Random Forest Classifier
- Bagging Classifier
- Extra Tree Classifier
- ADA Boosting Classifier
- Gradient Boosting Classifier

Once we train the ML model with various Classifiers, following Confusion Matrix can be plotted with given Dataset. Following are the results from them:

Design Train and Test basic ML Classifiers on Glove - Original Data:

In this process we use different classifiers and compare the accuracy classes. Models are Trained and Tested and results are as follows.

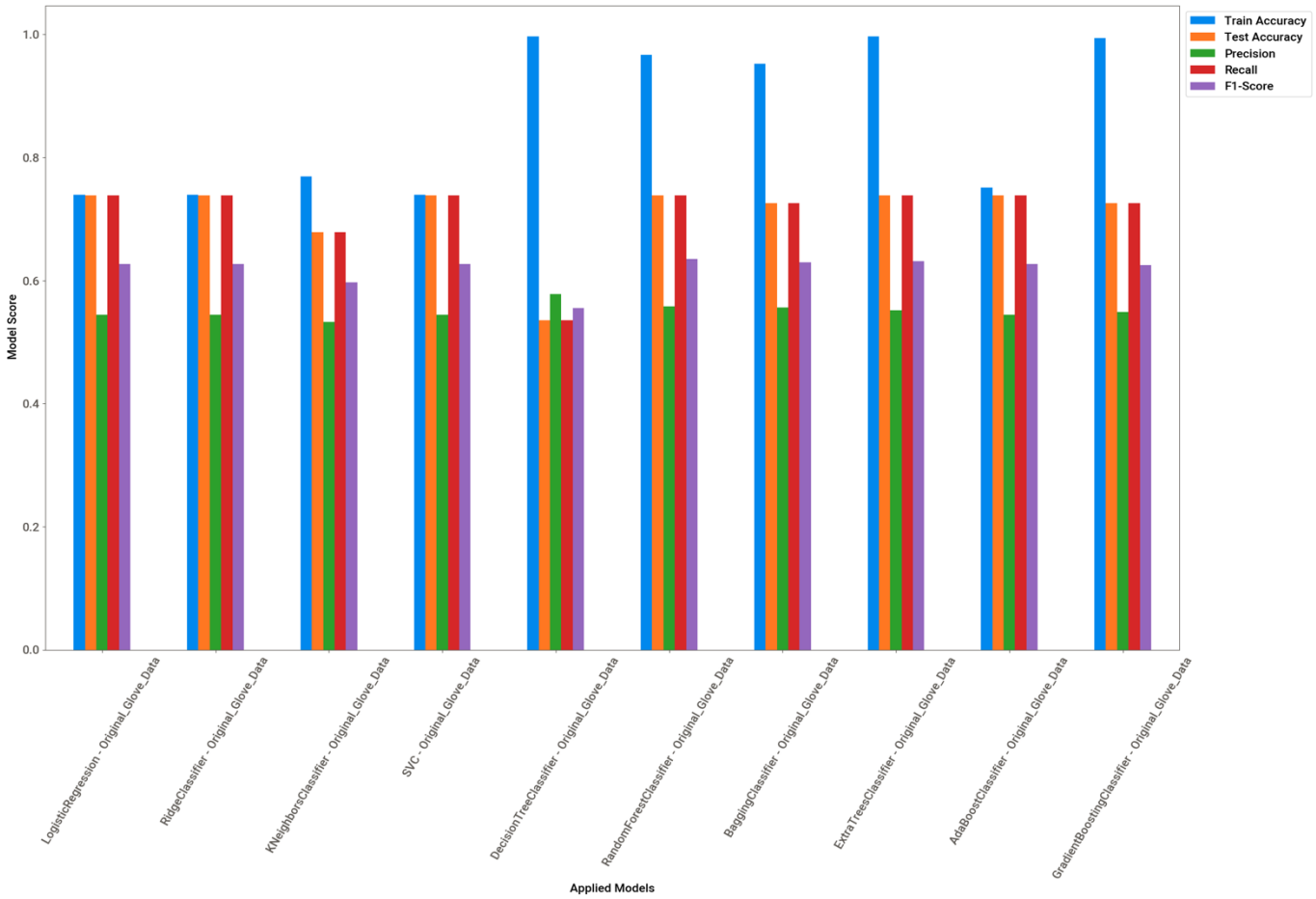
The Overall results of the Models which are trained on the Original Data of the Glove Dataframe

```
In [230]:
1 results_normal_Data_Df_glove
2
```

Out[230]:

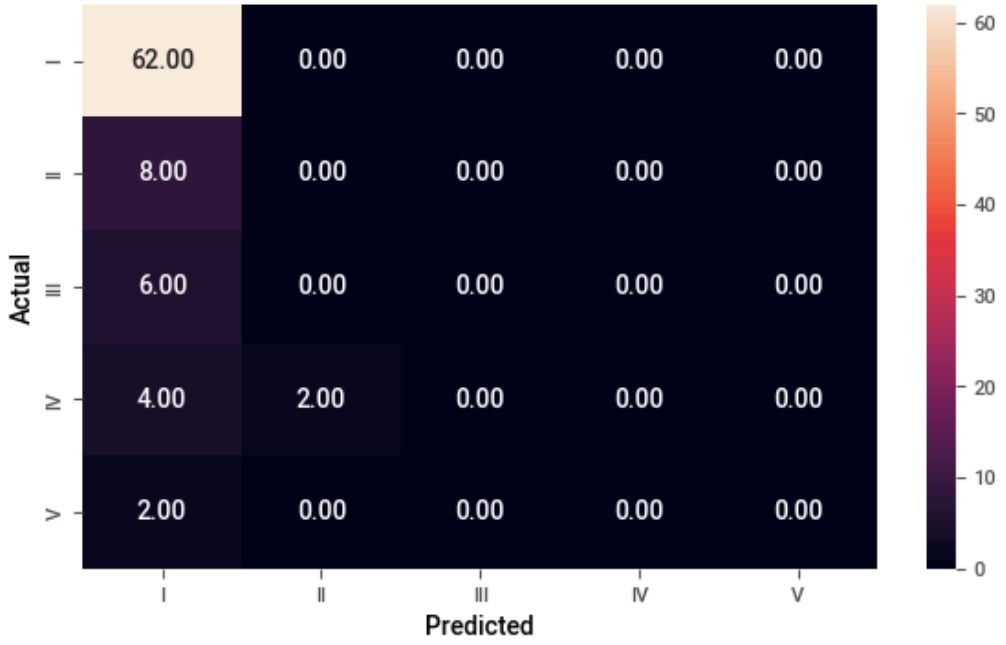
	Method	Train Accuracy	Test Accuracy	Precision	Recall	F1-Score
0	LogisticRegression - Original_Glove_Data	0.739521	0.738095	0.544785	0.738095	0.626875
1	RidgeClassifier - Original_Glove_Data	0.739521	0.738095	0.544785	0.738095	0.626875
2	KNeighborsClassifier - Original_Glove_Data	0.769461	0.678571	0.532550	0.678571	0.596758
3	SVC - Original_Glove_Data	0.739521	0.738095	0.544785	0.738095	0.626875
4	DecisionTreeClassifier - Original_Glove_Data	0.997006	0.535714	0.577595	0.535714	0.555607
5	RandomForestClassifier - Original_Glove_Data	0.967066	0.738095	0.558072	0.738095	0.635582
6	BaggingClassifier - Original_Glove_Data	0.952096	0.726190	0.555850	0.726190	0.629704
7	ExtraTreesClassifier - Original_Glove_Data	0.997006	0.738095	0.551348	0.738095	0.631199
8	AdaBoostClassifier - Original_Glove_Data	0.751497	0.738095	0.544785	0.738095	0.626875
9	GradientBoostingClassifier - Original_Glove_Data	0.994012	0.726190	0.549071	0.726190	0.625331

Plotting the Score of all Classification Models is as follows.



From above we can see that Random Forest Classifier is the best perfomed model. Following is the Confusion Matrix and Classification report of the Model.

```
RandomForestClassifier(n_estimators=10, random_state=1)
**************************************************************************
```



Classification Report - RandomForestClassifier - Original_Glove_Data

	precision	recall	f1-score	support
0	0.76	1.00	0.86	62
1	0.00	0.00	0.00	8
2	0.00	0.00	0.00	6
3	0.00	0.00	0.00	6
4	0.00	0.00	0.00	2
accuracy			0.74	84
macro avg	0.15	0.20	0.17	84
weighted avg	0.56	0.74	0.64	84

Design Train and Test basic ML Classifiers on Glove Up Sample Data:

In this process we use different classifiers and compare the accuracy classes. Models are Trained and Tested and results are as follows. Let's see overall Scores of the Models which we have trained the models using the DataFrame which is upsampled using traditional resampling method.

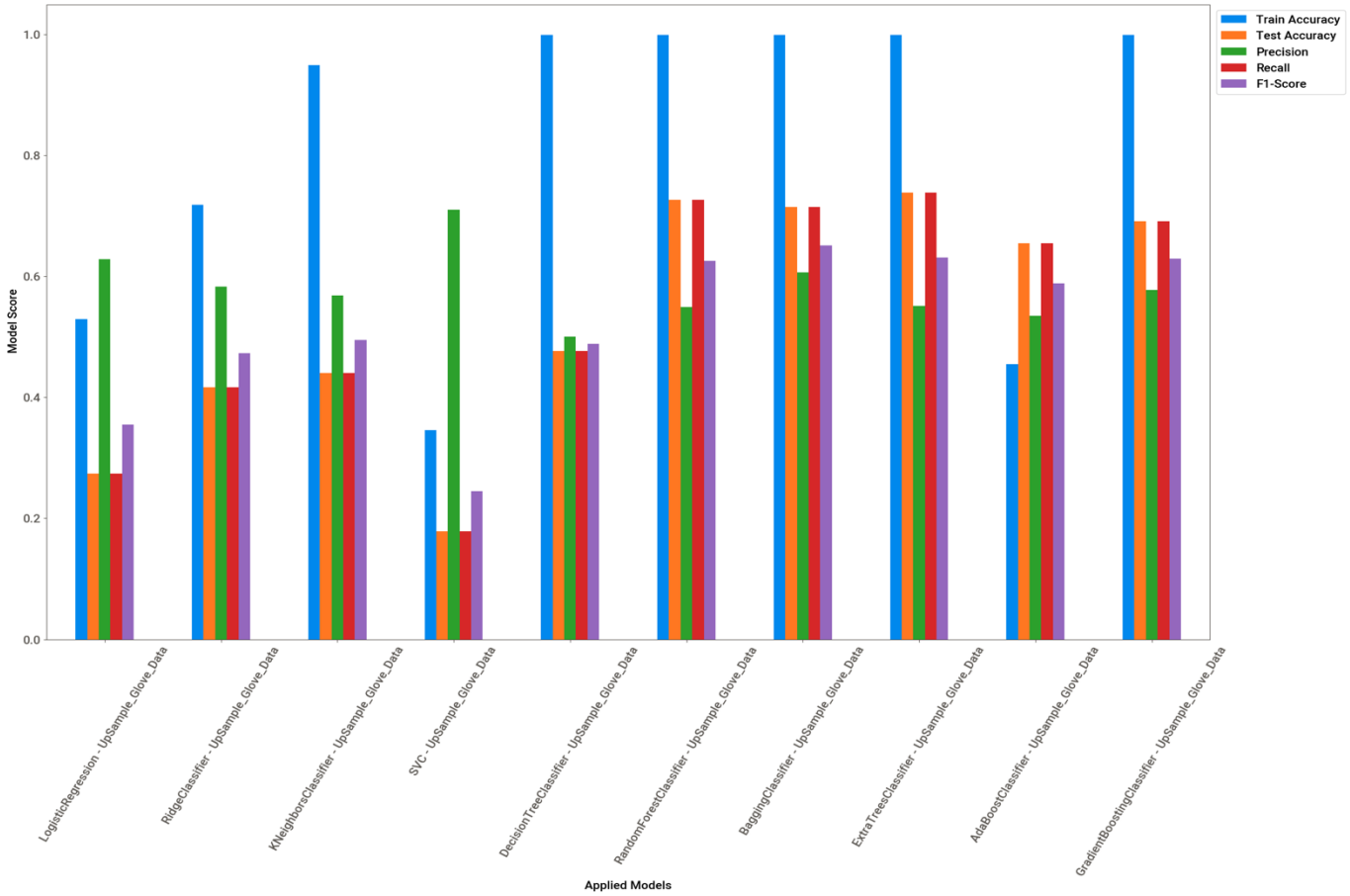
Following are the scores of the model.

```
In [235]:
1 results_upsample_Data_Df_glove
2
```

Out[235]:

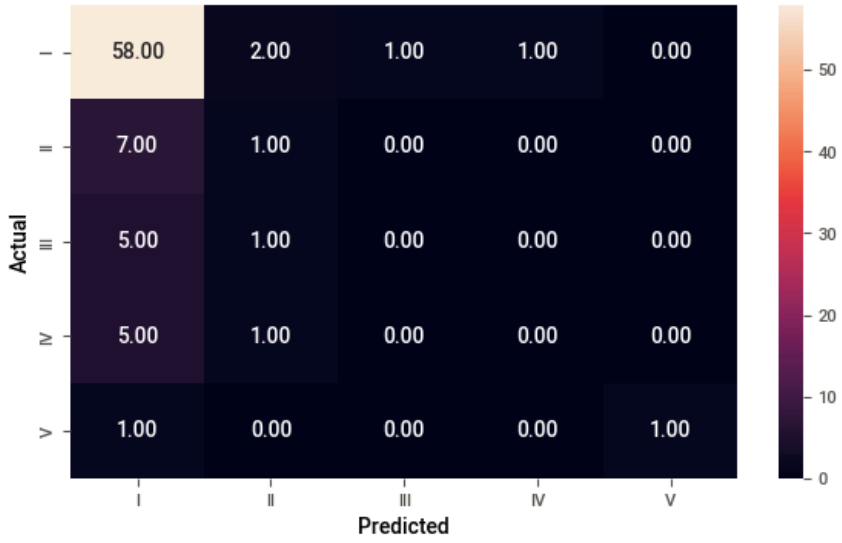
	Method	Train Accuracy	Test Accuracy	Precision	Recall	F1-Score
0	LogisticRegression - UpSample_Glove_Data	0.528745	0.273810	0.627876	0.273810	0.354828
1	RidgeClassifier - UpSample_Glove_Data	0.718219	0.416667	0.582634	0.416667	0.472569
2	KNeighborsClassifier - UpSample_Glove_Data	0.948988	0.440476	0.568098	0.440476	0.494792
3	SVC - UpSample_Glove_Data	0.345749	0.178571	0.709632	0.178571	0.244585
4	DecisionTreeClassifier - UpSample_Glove_Data	0.999190	0.476190	0.500404	0.476190	0.487997
5	RandomForestClassifier - UpSample_Glove_Data	0.999190	0.726190	0.549071	0.726190	0.625331
6	BaggingClassifier - UpSample_Glove_Data	0.999190	0.714286	0.606140	0.714286	0.650953
7	ExtraTreesClassifier - UpSample_Glove_Data	0.999190	0.738095	0.551348	0.738095	0.631199
8	AdaBoostClassifier - UpSample_Glove_Data	0.455061	0.654762	0.534148	0.654762	0.588337
9	GradientBoostingClassifier - UpSample_Glove_Data	0.998381	0.690476	0.577606	0.690476	0.629007

Plotting the Score of all Classification Models is as follows



From above we can see that Bagging Classifier is the best perfomed model. Following is the Confusion Matrix and Classification report of the Model.

```
BaggingClassifier(max_samples=0.75, n_estimators=30, oob_score=True,
                  random_state=1)
********************************************************************************
```



Classification Report - BaggingClassifier - UpSample_Glove_Data

	precision	recall	f1-score	support
0	0.76	0.94	0.84	62
1	0.20	0.12	0.15	8
2	0.00	0.00	0.00	6
3	0.00	0.00	0.00	6
4	1.00	0.50	0.67	2
accuracy			0.71	84
macro avg	0.39	0.31	0.33	84
weighted avg	0.61	0.71	0.65	84

Design Train and Test basic ML Classifiers on Glove Up Sampled using SMOTE Technique:

In this process we use different classifiers and compare the accuracy classes. Models are Trained and Tested and results are as follows. Let's see overall Scores of the Models which we have trained the models using the DataFrame which is up sampled using SMOTE resampling method.

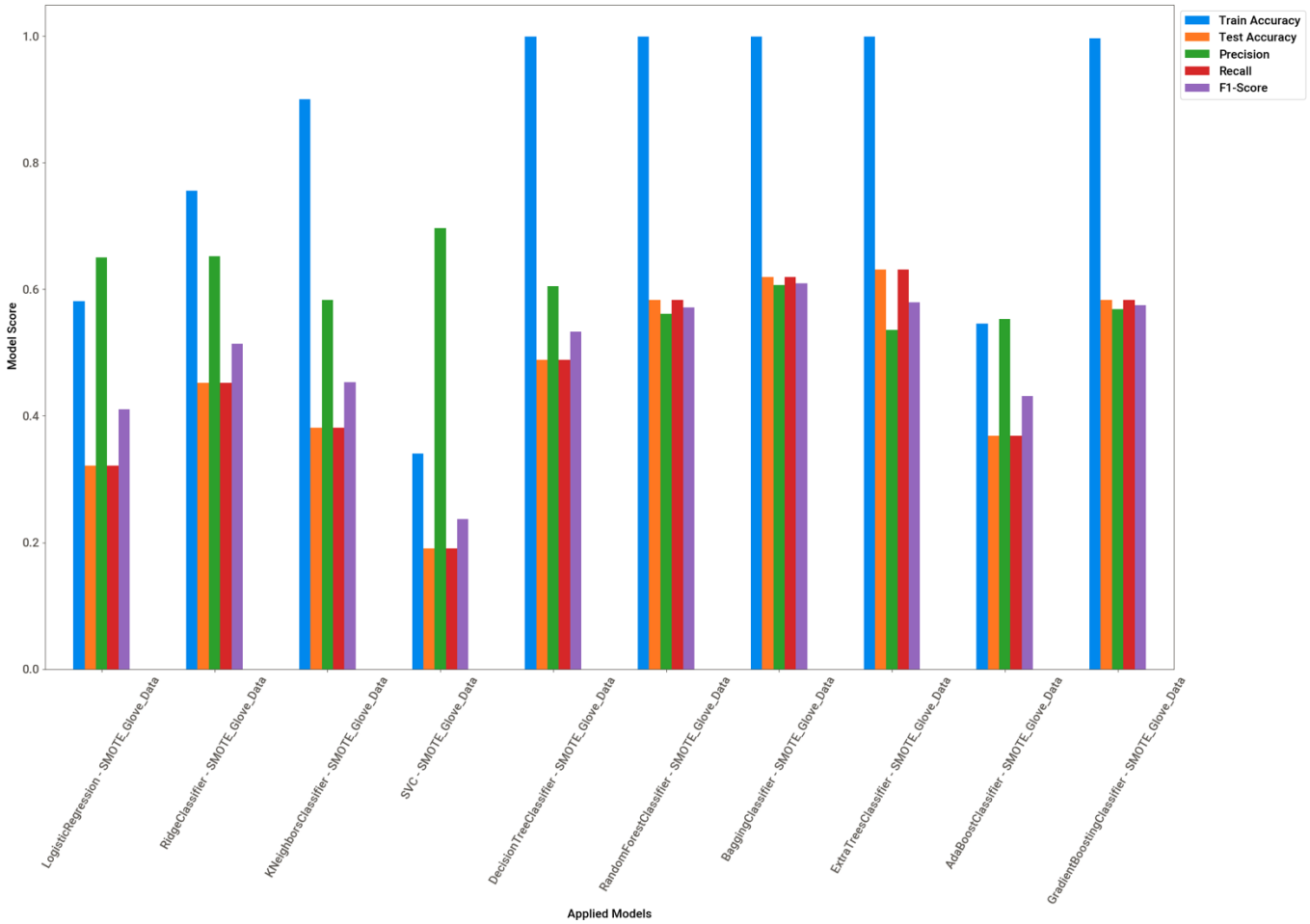
Following are the scores of the model.

```
In [240]:
1 results_smote_Data_Df_glove
2
```

Out[240]:

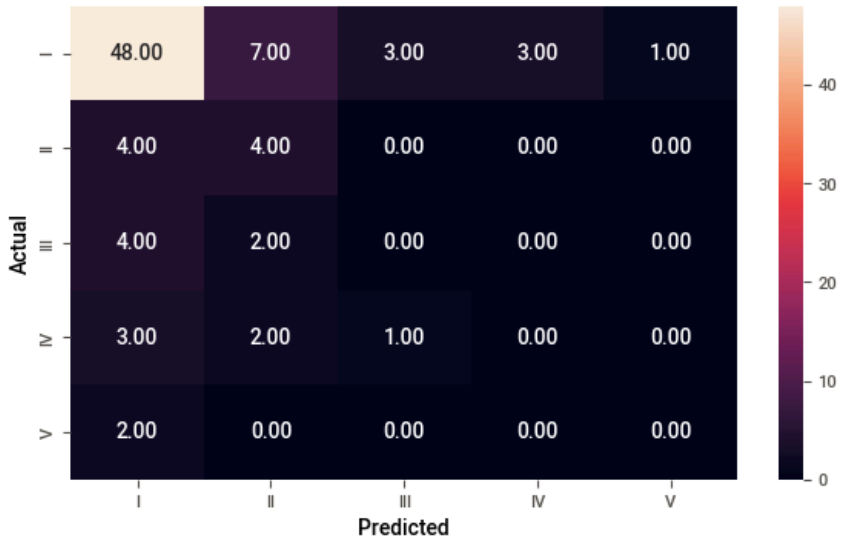
	Method	Train Accuracy	Test Accuracy	Precision	Recall	F1-Score
0	LogisticRegression - SMOTE_Glove_Data	0.581377	0.321429	0.650290	0.321429	0.409955
1	RidgeClassifier - SMOTE_Glove_Data	0.755466	0.452381	0.652091	0.452381	0.514272
2	KNeighborsClassifier - SMOTE_Glove_Data	0.899595	0.380952	0.582841	0.380952	0.453348
3	SVC - SMOTE_Glove_Data	0.340081	0.190476	0.696153	0.190476	0.237243
4	DecisionTreeClassifier - SMOTE_Glove_Data	0.999190	0.488095	0.604863	0.488095	0.532838
5	RandomForestClassifier - SMOTE_Glove_Data	0.999190	0.583333	0.560897	0.583333	0.571429
6	BaggingClassifier - SMOTE_Glove_Data	0.999190	0.619048	0.606193	0.619048	0.609201
7	ExtraTreesClassifier - SMOTE_Glove_Data	0.999190	0.630952	0.535877	0.630952	0.579541
8	AdaBoostClassifier - SMOTE_Glove_Data	0.545749	0.369048	0.553042	0.369048	0.431000
9	GradientBoostingClassifier - SMOTE_Glove_Data	0.995951	0.583333	0.568707	0.583333	0.575139

Plotting the Score of all Classification Models is as follows



From above we can see that Bagging Classifier is the best perfomed model. Following is the Confusion Matrix and Classification report of the Model.

```
BaggingClassifier(max_samples=0.75, n_estimators=30, oob_score=True,
                  random_state=1)
********************************************************************************
```



Classification Report - BaggingClassifier - SMOTE_Glove_Data

	precision	recall	f1-score	support
0	0.79	0.77	0.78	62
1	0.27	0.50	0.35	8
2	0.00	0.00	0.00	6
3	0.00	0.00	0.00	6
4	0.00	0.00	0.00	2
accuracy			0.62	84
macro avg	0.21	0.25	0.23	84
weighted avg	0.61	0.62	0.61	84

Comments

After applying SMOTE all model getting overfitted in training hence we are not recommended for deployment.

Following table represents overall summary:

In [243]:
```
1 results_Final_Df.sort_values(ascending=False,by='F1-Score')
2
```

Out[243]:

	Method	Train Accuracy	Test Accuracy	Precision	Recall	F1-Score
25	RandomForestClassifier - SMOTE_TFIDF_Data	0.996761	0.750000	0.666514	0.750000	0.677891
5	RandomForestClassifier - Original_TFIDF_Data	0.973054	0.750000	0.626884	0.750000	0.668153
46	BaggingClassifier - UpSample_Glove_Data	0.999190	0.714286	0.606140	0.714286	0.650953
29	GradientBoostingClassifier - SMOTE_TFIDF_Data	0.994332	0.690476	0.639961	0.690476	0.639106
1	RidgeClassifier - Original_TFIDF_Data	0.961078	0.702381	0.587093	0.702381	0.636301
35	RandomForestClassifier - Original_Glove_Data	0.967066	0.738095	0.558072	0.738095	0.635582
8	AdaBoostClassifier - Original_TFIDF_Data	0.694611	0.714286	0.615774	0.714286	0.633755
0	LogisticRegression - Original_TFIDF_Data	0.808383	0.738095	0.551348	0.738095	0.631199
28	AdaBoostClassifier - SMOTE_TFIDF_Data	0.390283	0.738095	0.551348	0.738095	0.631199
47	ExtraTreesClassifier - UpSample_Glove_Data	0.999190	0.738095	0.551348	0.738095	0.631199
17	ExtraTreesClassifier - UpSample_TFIDF_Data	0.999190	0.738095	0.551348	0.738095	0.631199
7	ExtraTreesClassifier - Original_TFIDF_Data	0.997006	0.738095	0.551348	0.738095	0.631199
37	ExtraTreesClassifier - Original_Glove_Data	0.997006	0.738095	0.551348	0.738095	0.631199
36	BaggingClassifier - Original_Glove_Data	0.952096	0.726190	0.555850	0.726190	0.629704
49	GradientBoostingClassifier - UpSample_Glove_Data	0.998381	0.690476	0.577606	0.690476	0.629007
15	RandomForestClassifier - UpSample_TFIDF_Data	0.999190	0.702381	0.569573	0.702381	0.628663
27	ExtraTreesClassifier - SMOTE_TFIDF_Data	0.999190	0.714286	0.560579	0.714286	0.628166
31	RidgeClassifier - Original_Glove_Data	0.739521	0.738095	0.544785	0.738095	0.626875
33	SVC - Original_Glove_Data	0.739521	0.738095	0.544785	0.738095	0.626875
38	AdaBoostClassifier - Original_Glove_Data	0.751497	0.738095	0.544785	0.738095	0.626875
30	LogisticRegression - Original_Glove_Data	0.739521	0.738095	0.544785	0.738095	0.626875
3	SVC - Original_TFIDF_Data	0.739521	0.738095	0.544785	0.738095	0.626875
6	BaggingClassifier - Original_TFIDF_Data	0.952096	0.702381	0.565553	0.702381	0.626584
45	RandomForestClassifier - UpSample_Glove_Data	0.999190	0.726190	0.549071	0.726190	0.625331
39	GradientBoostingClassifier - Original_Glove_Data	0.994012	0.726190	0.549071	0.726190	0.625331
10	LogisticRegression - UpSample_TFIDF_Data	0.953036	0.619048	0.633769	0.619048	0.624896
26	BaggingClassifier - SMOTE_TFIDF_Data	0.996761	0.678571	0.566984	0.678571	0.617012
56	BaggingClassifier - SMOTE_Glove_Data	0.999190	0.619048	0.606193	0.619048	0.609201
2	KNeighborsClassifier - Original_TFIDF_Data	0.781437	0.702381	0.537625	0.702381	0.609058
21	RidgeClassifier - SMOTE_TFIDF_Data	0.996761	0.607143	0.599206	0.607143	0.600907
32	KNeighborsClassifier - Original_Glove_Data	0.769461	0.678571	0.532550	0.678571	0.596758
9	GradientBoostingClassifier - Original_TFIDF_Data	0.979042	0.642857	0.545988	0.642857	0.590476

48	AdaBoostClassifier - UpSample_Glove_Data	0.455061	0.654762	0.534148	0.654762	0.588337
11	RidgeClassifier - UpSample_TFIDF_Data	0.995142	0.583333	0.595250	0.583333	0.588304
4	DecisionTreeClassifier - Original_TFIDF_Data	0.997006	0.595238	0.566614	0.595238	0.580543
57	ExtraTreesClassifier - SMOTE_Glove_Data	0.999190	0.630952	0.535877	0.630952	0.579541
59	GradientBoostingClassifier - SMOTE_Glove_Data	0.995951	0.583333	0.568707	0.583333	0.575139
18	AdaBoostClassifier - UpSample_TFIDF_Data	0.314980	0.619048	0.533069	0.619048	0.572850
55	RandomForestClassifier - SMOTE_Glove_Data	0.999190	0.583333	0.560897	0.583333	0.571429
19	GradientBoostingClassifier - UpSample_TFIDF_Data	0.996761	0.535714	0.619561	0.535714	0.569382
16	BaggingClassifier - UpSample_TFIDF_Data	0.997571	0.583333	0.547980	0.583333	0.565104
20	LogisticRegression - SMOTE_TFIDF_Data	0.957085	0.535714	0.603046	0.535714	0.562940
34	DecisionTreeClassifier - Original_Glove_Data	0.997006	0.535714	0.577595	0.535714	0.555607
54	DecisionTreeClassifier - SMOTE_Glove_Data	0.999190	0.488095	0.604863	0.488095	0.532838
24	DecisionTreeClassifier - SMOTE_TFIDF_Data	0.999190	0.511905	0.523903	0.511905	0.515234
51	RidgeClassifier - SMOTE_Glove_Data	0.755466	0.452381	0.652091	0.452381	0.514272
42	KNeighborsClassifier - UpSample_Glove_Data	0.948988	0.440476	0.568098	0.440476	0.494792
14	DecisionTreeClassifier - UpSample_TFIDF_Data	0.999190	0.452381	0.539377	0.452381	0.492063
44	DecisionTreeClassifier - UpSample_Glove_Data	0.999190	0.476190	0.500404	0.476190	0.487997
12	KNeighborsClassifier - UpSample_TFIDF_Data	0.953846	0.428571	0.536637	0.428571	0.473751
41	RidgeClassifier - UpSample_Glove_Data	0.718219	0.416667	0.582634	0.416667	0.472569
52	KNeighborsClassifier - SMOTE_Glove_Data	0.899595	0.380952	0.582841	0.380952	0.453348
22	KNeighborsClassifier - SMOTE_TFIDF_Data	0.913360	0.345238	0.604729	0.345238	0.432128
58	AdaBoostClassifier - SMOTE_Glove_Data	0.545749	0.369048	0.553042	0.369048	0.431000
50	LogisticRegression - SMOTE_Glove_Data	0.581377	0.321429	0.650290	0.321429	0.409955
40	LogisticRegression - UpSample_Glove_Data	0.528745	0.273810	0.627876	0.273810	0.354828
13	SVC - UpSample_TFIDF_Data	0.396761	0.226190	0.674825	0.226190	0.277904
43	SVC - UpSample_Glove_Data	0.345749	0.178571	0.709632	0.178571	0.244585
53	SVC - SMOTE_Glove_Data	0.340081	0.190476	0.696153	0.190476	0.237243
23	SVC - SMOTE_TFIDF_Data	0.450202	0.190476	0.691895	0.190476	0.231604

For HyperParameter Tuning of the Models following are the Points:

1. We shall consider Glove Data and TF-IDF Data.

2. We Shall Consider SVC, RandomForestClassifier, GradientBoostingClassifier, BaggingClassifier, ExtraTreesClassifier for Hyperparameter Tuning

3. From Above we can see that the models SVC, RandomForestClassifier, ExtraTreeClassifier, LGBMClassifier and XGBClassifier has more than 90% of Accuracy scores.

4. Random Forest Classifier has more f1 score compared to all other Classifiers

5. LogisticRegression, ADABoostingClassifier and KNNClassifier are not performing better so we can ignore these classifier for further improvising the model performance.

Few insights can be listed as below:

1. From above comparison we can see that the models SVC, RandomForestClassifier, ExtraTreeClassifier, LGBMClassifier and XGBClassifier has more than 90% of Accuracy scores.

2. ExtraTreeClassifier has more f1 score compared to all other Classifiers

3. LogisticRegression, ADABoostingClassifier and KNNClassifier are not performing better so we can ignore these classifiers for further improvising the model performance.

How we Improvised the Model – GridSearch CV

From Above Basic Machine Learning models accuracy score we are considering only 5 models to check which model suits best among SVC, RandomForestClassifier, Gradient Boosting, ExtraTreeClassifier, Bagging Classifier

We are using GridsearchCV to run with above ML algorithms. This is a library function that is a member of Sklearn's model selection package. It helps to look through predefined hyperparameters and fit the estimator / model on training set. This process helps in selecting the best parameters from the listed hyperparameters. Following are the results.

```
In [254]:
1 gridSearch_cv_results
2
```

Out[254]:

Model	Training Score	Accuracy	Recall	Precision	F1 score
SVC	0.967	0.738095	0.738	0.638	0.680
Random Forest Classifier	0.991	0.738095	0.738	0.551	0.631
Gradient Boosting Classifier	0.968	0.702381	0.702	0.579	0.634
ExtraTreesClassifier	0.995	0.714286	0.714	0.554	0.624
BaggingClassifier	0.972	0.702381	0.702	0.558	0.622
Random Forest Classifier_TFIDF_Up	0.994	0.738095	0.738	0.551	0.631
Random Forest Classifier_Glove_Smote	0.984	0.738095	0.738	0.558	0.636

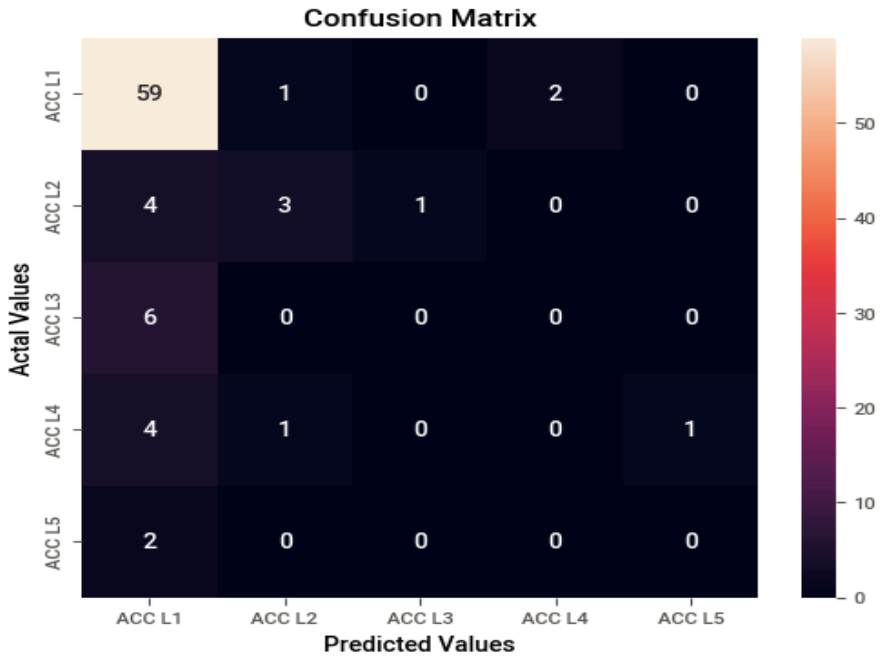
Insights:

1. From Above we can see that SVC model has performed better compared to other Models.
2. Basic SVC model has 19 percent Accuracy and 23.1% f1-score
3. But after changing the parameters and applying the model the Accuracy has gone up to 73.81% and also f1-score has not changed much and is 68.0%
4. Best parameters of SVC model are :

   Best params: {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
   Best score: 0.967

From above we can see that SVC Classifier is the best perfomed model. Following is the Confusion Matrix and Classification report of the Model.

```
Running model: SVC
Best params: {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
Best score: 0.967
              precision    recall  f1-score   support

           0       0.79      0.95      0.86        62
           1       0.60      0.38      0.46         8
           2       0.00      0.00      0.00         6
           3       0.00      0.00      0.00         6
           4       0.00      0.00      0.00         2

    accuracy                           0.74        84
   macro avg       0.28      0.27      0.26        84
weighted avg       0.64      0.74      0.68        84
```



Details of Steps involved in Milestone-2:

The main purpose of a neural network is to try to find the relationship between features in a data set., and it consists of a set of algorithms that mimic the work of the human brain. A “neuron” in a neural network is a mathematical function that collects and classifies information according to a specific architecture.

Classification problem involves predicting if something belongs to one class or not. In other words, while doing it we try to see something is one thing or another.

Types of Classification

- Suppose that you want to predict if a person has diabetes or not. İf you are facing this kind of situation, there are two possibilities, right? That is called **Binary Classification.**
- Suppose that you want to identify if a photo is of a toy, a person, or a cat, right? this is called **Multi-class Classification** because there are more than two options.
- Suppose you want to decide that which categories should be assigned to an article. If so, it is called **Multi-label Classification**, because one article could have more than one category assigned. Let’s take our explanation through this article. We may assign categories like “Deep Learning, TensorFlow, Classification” etc. to this article

Now we can move forward because we have a common understanding of the problem we will be working on. So, it is time for coding. I hope you are writing them down with me because the only way to get better, make fewer mistakes is to write more code.

Design, Train & Test NN Classifiers:

Convert Classification to Numeric problem:

In this section, we will create a classification model that uses categorical columns and TF-IDF features from accident description and label encoded target variable. We can use simple densely connected neural networks to make predictions.

Since we have ordinal relationship between each category in target variable, I have considered this one as numerical/regression problem and try to observe the ANN behavior

We are building simple Sequential Neural Network Model to Build the architecture. Model Summary is as follows.

In [259]:
```
model_ann_numval.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 50)                30350

 dense_1 (Dense)             (None, 100)               5100

 dense_2 (Dense)             (None, 150)               15150

 dense_3 (Dense)             (None, 40)                6040

 dense_4 (Dense)             (None, 1)                 41

=================================================================
Total params: 56,681
Trainable params: 56,681
Non-trainable params: 0
_________________________________________________________________
```

Fitting the Above Model with 50 Epochs as follows.

In [261]:
```
# fit the keras model on the dataset
training_history__ann_numval = model_ann_numval.fit(X_train, Y_train, epochs=50,
                                                    batch_size=8, verbose=1, validation_data=(X_test, Y_test),
                                                    callbacks=[rlrp_ann_numval])
```

```
Epoch 1/50
42/42 [==============================] - 2s 10ms/step - loss: 1.1826 - accuracy: 0.4132 - val_loss: 1.0413 - val_accu
racy: 0.4762 - lr: 0.0010
Epoch 2/50
42/42 [==============================] - 0s 4ms/step - loss: 1.1332 - accuracy: 0.5150 - val_loss: 1.8327 - val_accur
acy: 0.7381 - lr: 0.0010
Epoch 3/50
42/42 [==============================] - 0s 4ms/step - loss: 0.9627 - accuracy: 0.4880 - val_loss: 1.0085 - val_accur
acy: 0.2976 - lr: 0.0010
Epoch 4/50
42/42 [==============================] - 0s 4ms/step - loss: 0.8070 - accuracy: 0.4940 - val_loss: 1.0023 - val_accur
acy: 0.3452 - lr: 0.0010
Epoch 5/50
42/42 [==============================] - 0s 4ms/step - loss: 0.6746 - accuracy: 0.5778 - val_loss: 1.0423 - val_accur
acy: 0.5476 - lr: 0.0010
Epoch 6/50
42/42 [==============================] - 0s 4ms/step - loss: 0.4862 - accuracy: 0.5898 - val_loss: 1.3050 - val_accur
acy: 0.6905 - lr: 0.0010
Epoch 7/50
42/42 [==============================] - 0s 4ms/step - loss: 0.3521 - accuracy: 0.6677 - val_loss: 1.2467 - val_accur
acy: 0.3571 - lr: 0.0010
Epoch 8/50
42/42 [==============================] - 0s 6ms/step - loss: 0.3482 - accuracy: 0.6677 - val_loss: 1.1886 - val_accur
acy: 0.4881 - lr: 0.0010
Epoch 9/50
42/42 [==============================] - 0s 6ms/step - loss: 0.2201 - accuracy: 0.7425 - val_loss: 1.2930 - val_accur
acy: 0.6310 - lr: 0.0010
```

Training and Test Accuracy of the Model is as follows:

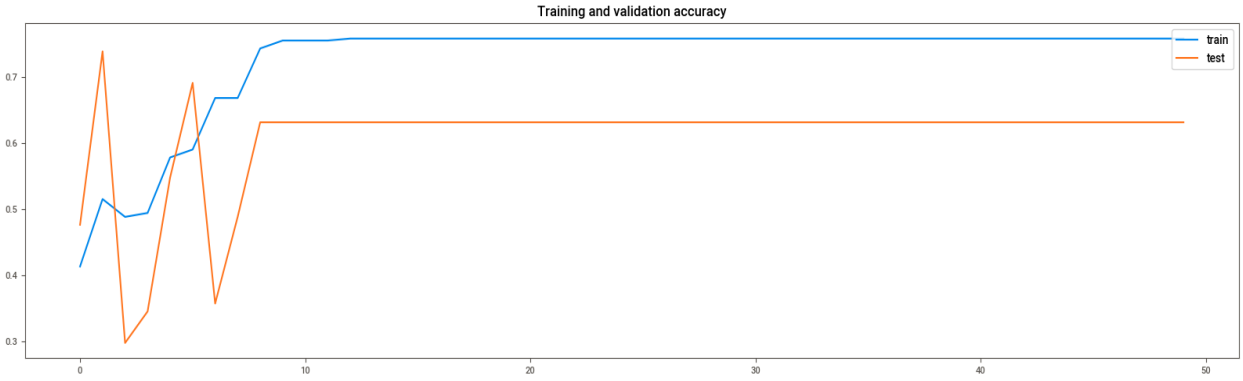
In [262]:
```
# evaluate the keras model
_, train_accuracy_ann_numval = model_ann_numval.evaluate(X_train, Y_train, batch_size=8, verbose=0)
_, test_accuracy_ann_numval = model_ann_numval.evaluate(X_test, Y_test, batch_size=8, verbose=0)
print('Train accuracy: %.2f' % (train_accuracy_ann_numval*100))
print('Test accuracy: %.2f' % (test_accuracy_ann_numval*100))
```

```
Train accuracy: 75.75
Test accuracy: 63.10
```

Loss Learning Curves of the above model is as follows:

Above one is underfit model, it can be identified from the learning curve of the tr aining loss only. It is sh**owing** noisy values of relatively high loss, indicating that the model was unable to learn the training dataset at all and model does not have a suitable capacity for the complexity of the dataset.

Following plot shows Training and Validation Accuracy



Above plot indicates the Accuracies of the Train and Test data at each epochs and we can see that there is no learning of data after almost 9 epochs. So lets try with other types of models.

Creating a Model with Categorical features With Glove Data

In this section, we will create a classification model that uses categorical columns along with the Glove data. Since the data for these columns is well structured and doesn't contain any sequential or spatial pattern, we can use simple densely connected neural networks to make predictions.

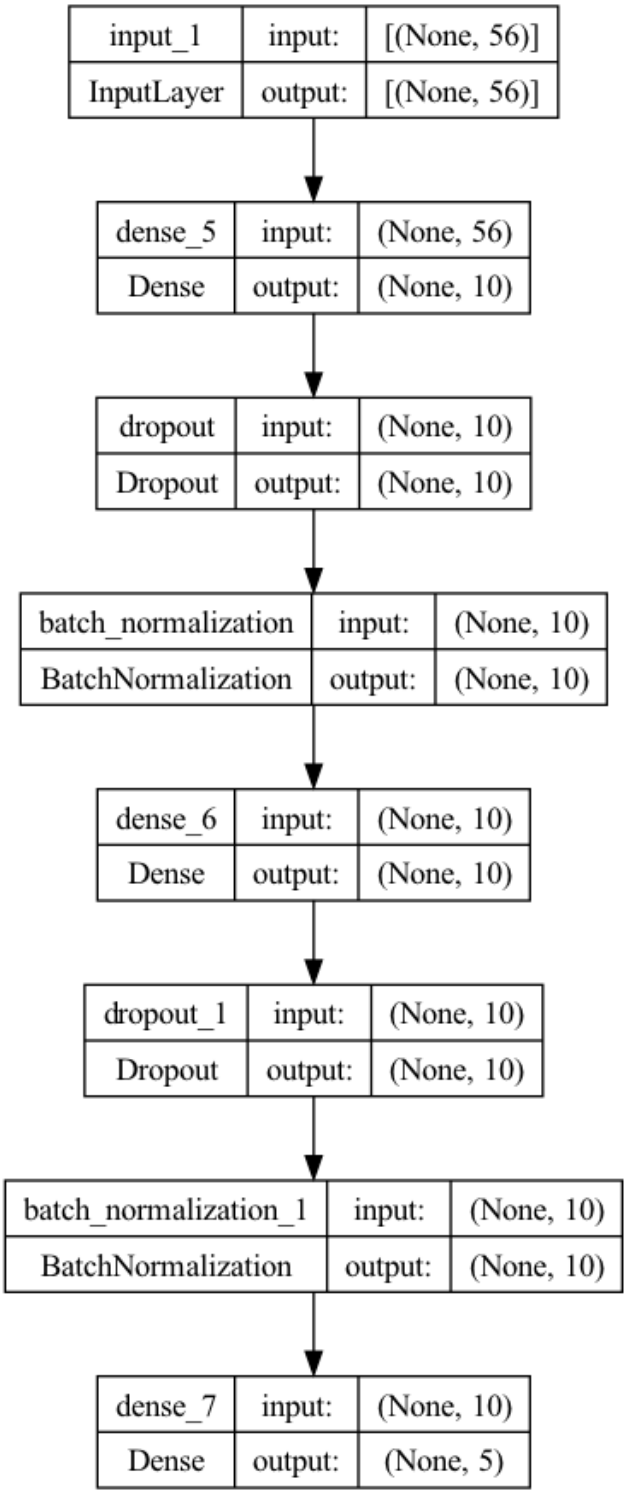
Model Summary is as follows:

```
In [272]: 1 print(model.summary())
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 56)]              0

 dense_5 (Dense)             (None, 10)                570

 dropout (Dropout)           (None, 10)                0

 batch_normalization (BatchN  (None, 10)               40
 ormalization)

 dense_6 (Dense)             (None, 10)                110

 dropout_1 (Dropout)         (None, 10)                0

 batch_normalization_1 (Batc  (None, 10)               40
 hNormalization)

 dense_7 (Dense)             (None, 5)                 55

=================================================================
Total params: 815
Trainable params: 775
Non-trainable params: 40
_________________________________________________________________
None
```

We are using One Input Layer followed by dense and dropout layers. Then we are passing through Batch Normalization and then again to One dense and dropout layers and again through batch normalization then one Output Layer.

Visualizing the Model Architecture in Plotting as follows



Fitting the Above Model with 50 Epochs as follows.

In [276]:
```
# fit the keras model on the dataset
training_history_ann_cat = model_ann_cat.fit(X_cat_train, y_cat_train, epochs=50, batch_size=8, verbose=1, validati
```

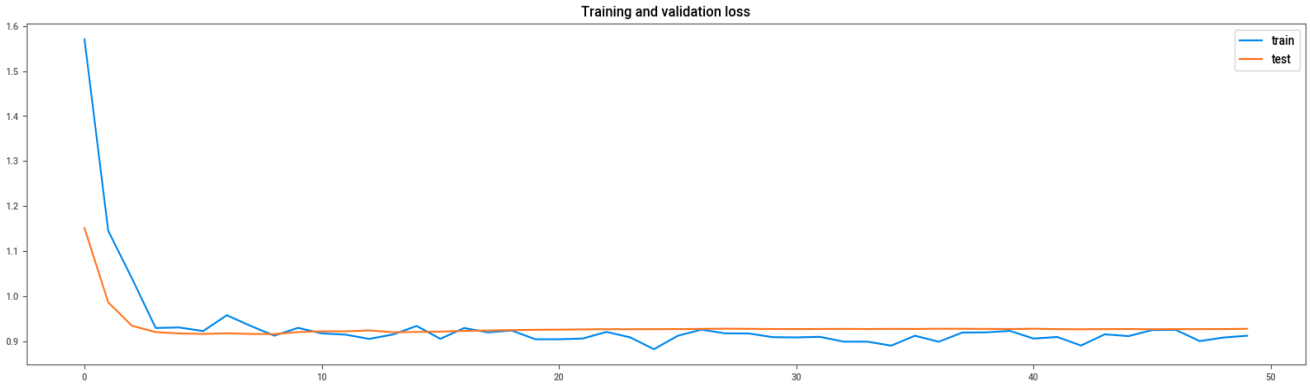
```
Epoch 1/50
11/11 [==============================] - 0s 3ms/steposs: 1.6142 - acc: 0.3789
- train_f1: 0.000000 - train_precision: 0.000000 - train_recall 0.000000
42/42 [==============================] - 5s 21ms/step - loss: 1.5704 - acc: 0.4072 - val_loss: 1.1511 - val_acc: 0.73
81 - lr: 0.0100
Epoch 2/50
11/11 [==============================] - 0s 7ms/steposs: 1.1638 - acc: 0.
- train_f1: 0.739521 - train_precision: 0.739521 - train_recall 0.739521
42/42 [==============================] - 0s 11ms/step - loss: 1.1452 - acc: 0.6287 - val_loss: 0.9863 - val_acc: 0.73
81 - lr: 0.0100
Epoch 3/50
11/11 [==============================] - 0s 3ms/steposs: 1.0209 - acc: 0.73
- train_f1: 0.739521 - train_precision: 0.739521 - train_recall 0.739521
42/42 [==============================] - 0s 10ms/step - loss: 1.0395 - acc: 0.7305 - val_loss: 0.9343 - val_acc: 0.73
81 - lr: 0.0100
Epoch 4/50
11/11 [==============================] - 0s 3ms/steposs: 0.9130 - acc: 0.74
- train_f1: 0.739521 - train_precision: 0.739521 - train_recall 0.739521
42/42 [==============================] - 0s 12ms/step - loss: 0.9295 - acc: 0.7395 - val_loss: 0.9203 - val_acc: 0.73
81 - lr: 0.0100
Epoch 5/50
11/11 [==============================] - 0s 3ms/steposs: 0.9490 - acc: 0.71
- train_f1: 0.739521 - train_precision: 0.739521 - train_recall 0.739521
42/42 [==============================] - 0s 10ms/step - loss: 0.9305 - acc: 0.7395 - val_loss: 0.9173 - val_acc: 0.73
81 - lr: 0.0100
Epoch 6/50
11/11 [==============================] - 0s 3ms/steposs: 0.9097 - acc: 0.74
- train_f1: 0.739521 - train_precision: 0.739521 - train_recall 0.739521
42/42 [==============================] - 0s 9ms/step - loss: 0.9225 - acc: 0.7395 - val_loss: 0.9162 - val_acc: 0.738
1 - lr: 0.0100
```

Training and Test Accuracy of the Model is as follows:

In [277]:
```
# evaluate the keras model
train_loss_ann_cat,train_accuracy_ann_cat = model_ann_cat.evaluate(X_cat_train, y_cat_train, batch_size=8, verbose=
val_loss_ann_cat,test_accuracy_ann_cat = model_ann_cat.evaluate(X_cat_test, y_cat_test, batch_size=8, verbose=1)


print('Train accuracy: %.2f' % (train_accuracy_ann_cat*100))
print('Test accuracy: %.2f' % (test_accuracy_ann_cat*100))

print('Train loss: %.2f' % (train_loss_ann_cat*100))
print('Val loss: %.2f' % (val_loss_ann_cat*100))
```

```
42/42 [==============================] - 0s 2ms/step - loss: 0.8831 - acc: 0.7395
11/11 [==============================] - 0s 2ms/step - loss: 0.9275 - acc: 0.7381
Train accuracy: 73.95
Test accuracy: 73.81
Train loss: 88.31
Val loss: 92.75
```

Loss Learning Curves of the above model is as follows:



Classification Report and f1 Score of the above model is as follows:

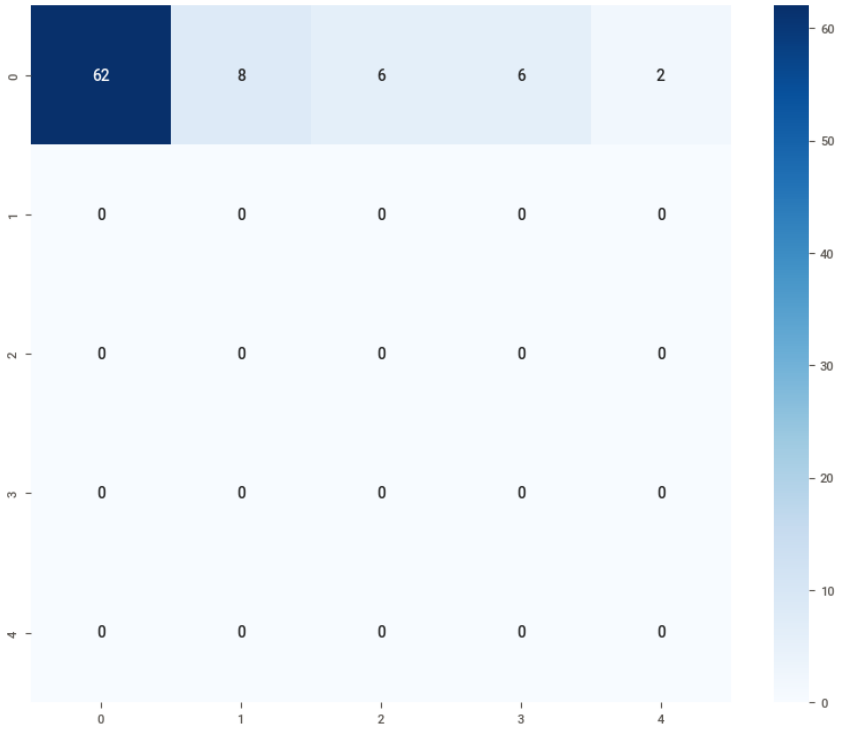
In [279]:
```
f1_score_ann_cat=f1_score(x, y_cat_test, average='micro')

print('f1_score: %.2f' % (f1_score_ann_cat*100))
```

```
f1_score: 73.81
```

In [280]:
```
print(f'Classification Report:\n{classification_report(x, y_cat_test)}')
```

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

           0       1.00      0.74      0.85        84
           1       0.00      0.00      0.00         0
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0
           4       0.00      0.00      0.00         0

   micro avg       0.74      0.74      0.74        84
   macro avg       0.20      0.15      0.17        84
weighted avg       1.00      0.74      0.85        84
 samples avg       0.74      0.74      0.74        84
```

Confusion Matrix is :



Following plot shows Training and Validation Accuracy

```
In [283]:
# plot accuracy learning curves
plt.plot(epochs, training_history_ann_cat.history['acc'], label = 'train')
plt.plot(epochs, training_history_ann_cat.history['val_acc'], label = 'test')
plt.legend(loc = 'upper right')
plt.title ('Training and validation accuracy')
```

```
Out[283]: Text(0.5, 1.0, 'Training and validation accuracy')
```

Above one is good fit, it is identified by a training and validation loss that decreas es to a point of stability with a minimal gap between the two final loss values. The los s of the model will almost always be lower on the training dataset than the validation dataset.

We could see it accuracy continually rise during training. As expected, we see the learning curves for accuracy on the test dataset plateau, indicating that the model has no longer overfit the training dataset and it is generalized model

Multiclass classification-ANN Model- Target variable - One hot encoded with Original Data:

In this section, we will create a classification model that uses categorical columns and TF-IDF features from accident description and one-hot encoded target variable. We can use simple densely connected neural networks to make predictions.

In this Data Frame the data consists of the features of TF-IDF Vectors. We have built this with max features of 200. And N grams of 3. So Total of 600 Vectors. So the data frame consists of 606 features and which is passed to the model.

Following is the Pictorial representation of the model summary which displays the level wise nodes and params.

In [286]:
```
model.summary()
```

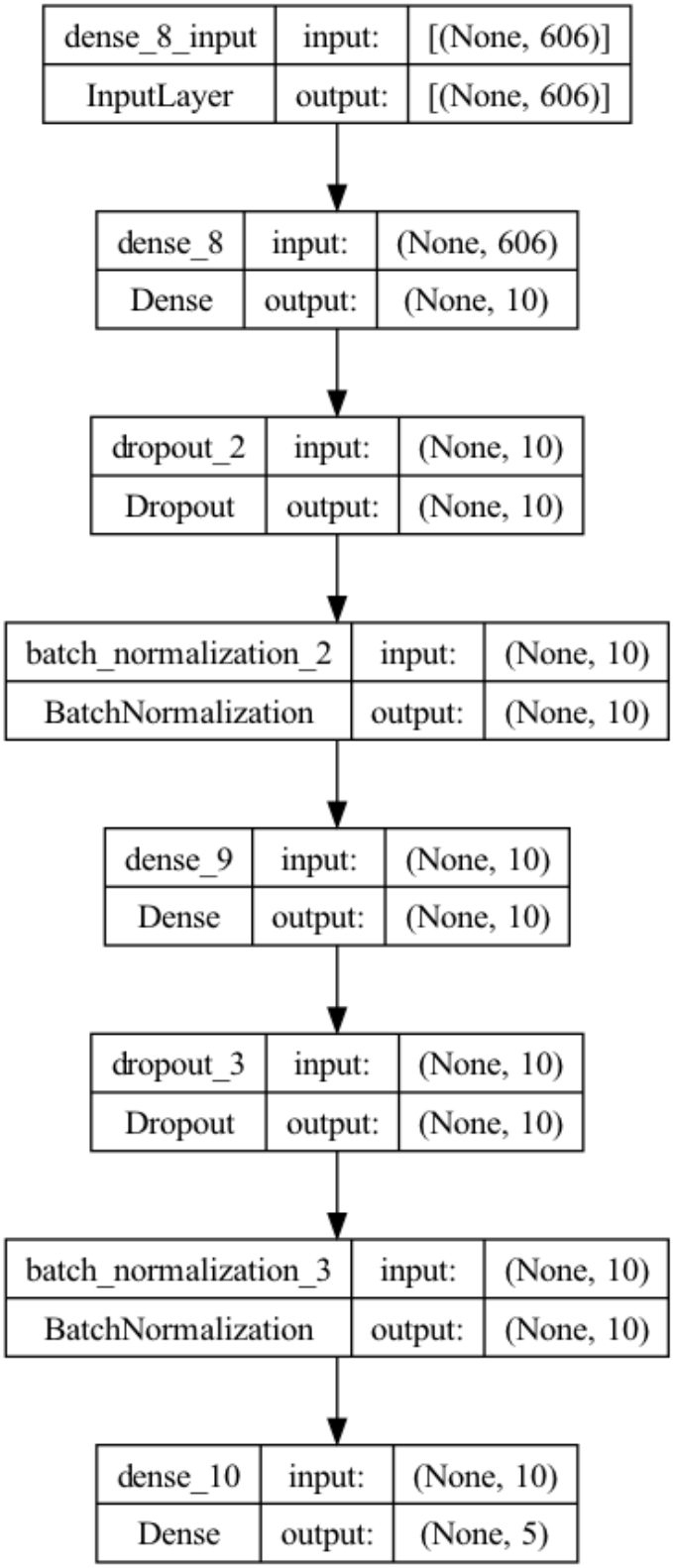
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 10)                6070

 dropout_2 (Dropout)         (None, 10)                0

 batch_normalization_2 (Batc  (None, 10)               40
 hNormalization)

 dense_9 (Dense)             (None, 10)                110

 dropout_3 (Dropout)         (None, 10)                0

 batch_normalization_3 (Batc  (None, 10)               40
 hNormalization)

 dense_10 (Dense)            (None, 5)                 55

=================================================================
Total params: 6,315
Trainable params: 6,275
Non-trainable params: 40
```

Fitting the Above Model with 50 Epochs as follows.

In [289]:
```
# fit the keras model on the dataset
training_history_ann_o_data = model_ann_o_data.fit(X_train, Y_train_dummy, epochs=50,
                                                   batch_size=8, verbose=1, validation_data=(X_test, Y_test_dummy),
                                                   callbacks=[rlrp, metrics])
```

```
Epoch 1/50
3/3 [==============================] - 0s 4ms/step loss: 1.8356 - categorical_accuracy: 0.2419
— train_f1: 0.000000 — train_precision: 0.000000 — train_recall 0.000000
42/42 [==============================] - 2s 19ms/step - loss: 1.7931 - categorical_accuracy: 0.2665 - val_loss: 2.000
3 - val_categorical_accuracy: 0.1905 - lr: 0.0010
Epoch 2/50
3/3 [==============================] - 0s 5ms/step loss: 1.5355 - categorical_accuracy: 0.45
— train_f1: 0.068966 — train_precision: 1.000000 — train_recall 0.035714
42/42 [==============================] - 0s 7ms/step - loss: 1.5298 - categorical_accuracy: 0.4611 - val_loss: 1.1425
- val_categorical_accuracy: 0.7381 - lr: 0.0010
Epoch 3/50
3/3 [==============================] - 0s 6ms/step loss: 1.3965 - categorical_accuracy: 0.53
— train_f1: 0.649351 — train_precision: 0.714286 — train_recall 0.595238
42/42 [==============================] - 0s 9ms/step - loss: 1.3771 - categorical_accuracy: 0.5389 - val_loss: 1.0628
- val_categorical_accuracy: 0.7381 - lr: 0.0010
Epoch 4/50
3/3 [==============================] - 0s 3ms/step loss: 1.2830 - categorical_accuracy: 0.56
— train_f1: 0.730539 — train_precision: 0.734940 — train_recall 0.726190
42/42 [==============================] - 0s 12ms/step - loss: 1.2574 - categorical_accuracy: 0.5898 - val_loss: 1.016
6 - val_categorical_accuracy: 0.7381 - lr: 0.0010
Epoch 5/50
3/3 [==============================] - 0s 5ms/step loss: 1.2587 - categorical_accuracy: 0.59
— train_f1: 0.722892 — train_precision: 0.731707 — train_recall 0.714286
42/42 [==============================] - 0s 8ms/step - loss: 1.2392 - categorical_accuracy: 0.6078 - val_loss: 0.9835
- val_categorical_accuracy: 0.7381 - lr: 0.0010
Epoch 6/50
3/3 [==============================] - 0s 3ms/step loss: 1.0945 - categorical_accuracy: 0.65
— train_f1: 0.738095 — train_precision: 0.738095 — train_recall 0.738095
42/42 [==============================] - 0s 10ms/step - loss: 1.0873 - categorical_accuracy: 0.6617 - val_loss: 0.957
2 - val_categorical_accuracy: 0.7381 - lr: 0.0010
Epoch 7/50
3/3 [==============================] - 0s 3ms/step loss: 1.0165 - categorical_accuracy: 0.69
```

Visualizing the Model Architecture in Plotting as follows



Training and Test Accuracy of the Model is as follows:

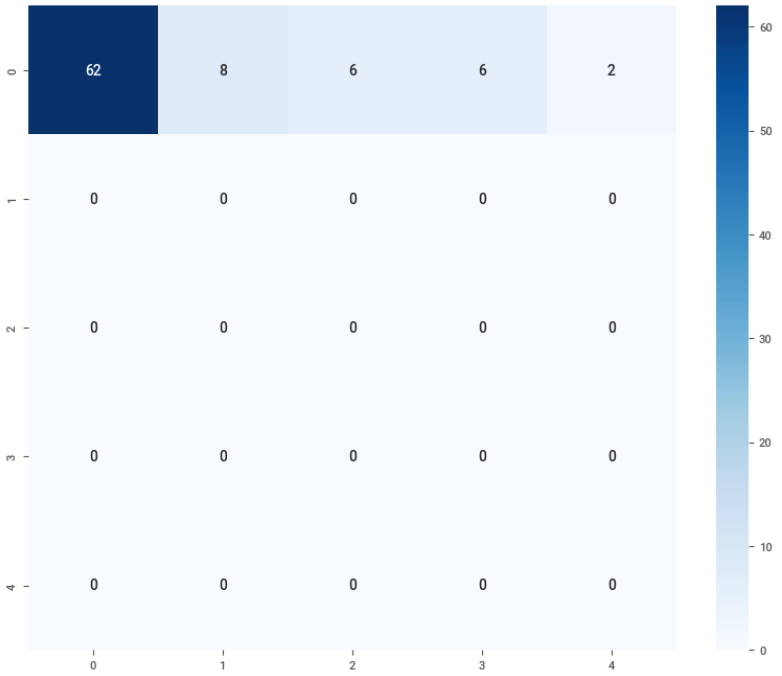
```
In [290]:
# evaluate the keras model
train_loss_ann_o_data,train_accuracy_ann_o_data = model_ann_o_data.evaluate(X_train, Y_train_du
val_loss_ann_o_data,test_accuracy_ann_o_data = model_ann_o_data.evaluate(X_test, Y_test_dummy,


print('Train accuracy: %.2f' % (train_accuracy_ann_o_data*100))
print('Test accuracy: %.2f' % (test_accuracy_ann_o_data*100))

print('Train loss: %.2f' % (train_loss_ann_o_data*100))
print('Val loss: %.2f' % (val_loss_ann_o_data*100))
```

```
42/42 [==============================] - 0s 3ms/step - loss: 0.9047 - categorical_accuracy: 0.7395
11/11 [==============================] - 0s 8ms/step - loss: 0.9286 - categorical_accuracy: 0.7381
Train accuracy: 73.95
Test accuracy: 73.81
Train loss: 90.47
Val loss: 92.86
```

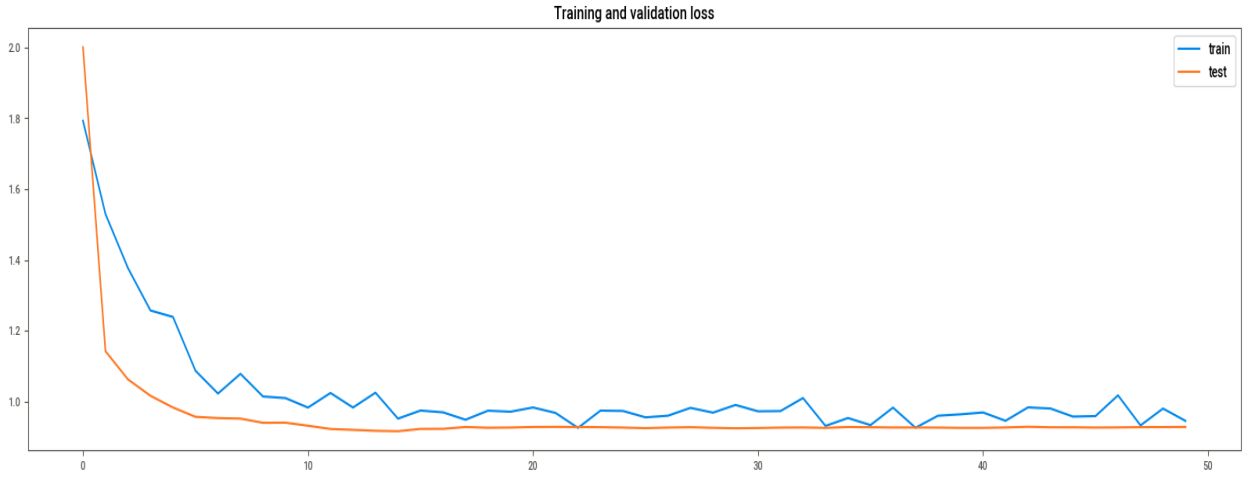
Classification Report and Confusion Matrix of the above trained model is as follows:

```
In [293]:
print(f'Classification Report:\n{classification_report(x, Y_test_dummy)}')
```

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

           0       1.00      0.74      0.85        84
           1       0.00      0.00      0.00         0
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0
           4       0.00      0.00      0.00         0

   micro avg       0.74      0.74      0.74        84
   macro avg       0.20      0.15      0.17        84
weighted avg       1.00      0.74      0.85        84
 samples avg       0.74      0.74      0.74        84
```

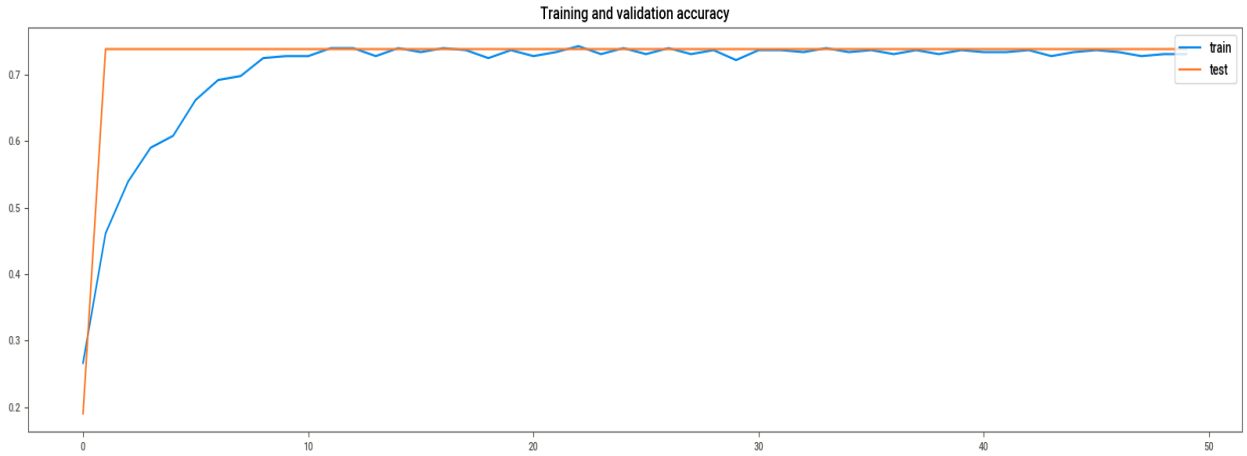


Loss Learning Curves of the above model is as follows:



Above one is good fit, it is identified by a training and validation loss that decreases to a point of stability with a minimal gap between the two final loss values. The loss of the model will almost always be lower on the training dataset than the validation dataset.

Following plot shows Training and Validation Accuracy



We could see it accuracy continually rise during training. As expected, we see the learning curves for accuracy on the test dataset plateau, indicating that the model has no l onger overfit the training dataset and it is generalized model.

Multiclass classification-ANN Model- Target variable - One hot encoded with SMOTE Data:

In this section, we will create a classification model that uses categorical columns and tf-idf features from accident description and one-hot encoded target variable. We can use simple densely connected neural networks to make predictions.

In this Data Frame the data consists of the features of TF-IDF Vectors. We have built this with max features of 200. And N grams of 3. So Total of 600 Vectors. So the data frame consists of 606 features and which is passed to the model. Where the Data Frame is up sampled using SMOTE Technique according to the Accident Level Column.

Following is the Pictorial representation of the model summary which displays the level wise nodes and params.

```
In [299]: 1 model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_11 (Dense)            (None, 10)                6070

 dropout_4 (Dropout)         (None, 10)                0

 batch_normalization_4 (Batc  (None, 10)               40
 hNormalization)

 dense_12 (Dense)            (None, 10)                110

 dropout_5 (Dropout)         (None, 10)                0

 batch_normalization_5 (Batc  (None, 10)               40
 hNormalization)

 dense_13 (Dense)            (None, 5)                 55

=================================================================
Total params: 6,315
Trainable params: 6,275
Non-trainable params: 40
_________________________________________________________________
```

Fitting the Above Model with 50 Epochs as follows.

In [301]:
```
# fit the keras model on the dataset
training_history_ann_smote = model_ann_smote.fit(X_train_smote, Y_train_smote_dummy, epochs=50, batch_size=8,
                                                 verbose=1, validation_data=(X_test, Y_test_dummy),
                                                 callbacks=[rlrp, metrics])
```

```
Epoch 1/50
39/39 [==============================] - 0s 2ms/step loss: 1.7140 - categorical_accuracy: 0.
- train_f1: 0.000000 - train_precision: 0.000000 - train_recall 0.000000
155/155 [==============================] - 3s 6ms/step - loss: 1.7138 - categorical_accuracy: 0.2211 - val_loss: 1.61
79 - val_categorical_accuracy: 0.1190 - lr: 0.0100
Epoch 2/50
39/39 [==============================] - 0s 2ms/step loss: 1.5155 - categorical_accuracy: 0.
- train_f1: 0.001618 - train_precision: 1.000000 - train_recall 0.000810
155/155 [==============================] - 1s 5ms/step - loss: 1.5138 - categorical_accuracy: 0.3441 - val_loss: 1.45
12 - val_categorical_accuracy: 0.2381 - lr: 0.0100
Epoch 3/50
39/39 [==============================] - 0s 2ms/step loss: 1.4001 - categorical_accuracy: 0.
- train_f1: 0.001614 - train_precision: 0.250000 - train_recall 0.000810
155/155 [==============================] - 1s 5ms/step - loss: 1.4008 - categorical_accuracy: 0.3757 - val_loss: 1.38
41 - val_categorical_accuracy: 0.2976 - lr: 0.0100
Epoch 4/50
39/39 [==============================] - 0s 2ms/step loss: 1.3266 - categorical_accuracy: 0.
- train_f1: 0.327632 - train_precision: 0.873684 - train_recall 0.201619
155/155 [==============================] - 1s 7ms/step - loss: 1.3200 - categorical_accuracy: 0.4235 - val_loss: 1.47
88 - val_categorical_accuracy: 0.3214 - lr: 0.0100
Epoch 5/50
39/39 [==============================] - 0s 2ms/step loss: 1.2628 - categorical_accuracy: 0.
- train_f1: 0.003210 - train_precision: 0.181818 - train_recall 0.001619
155/155 [==============================] - 1s 6ms/step - loss: 1.2622 - categorical_accuracy: 0.4850 - val_loss: 1.38
60 - val_categorical_accuracy: 0.6786 - lr: 0.0100
```

Training and Test Accuracy of the Model is as follows:

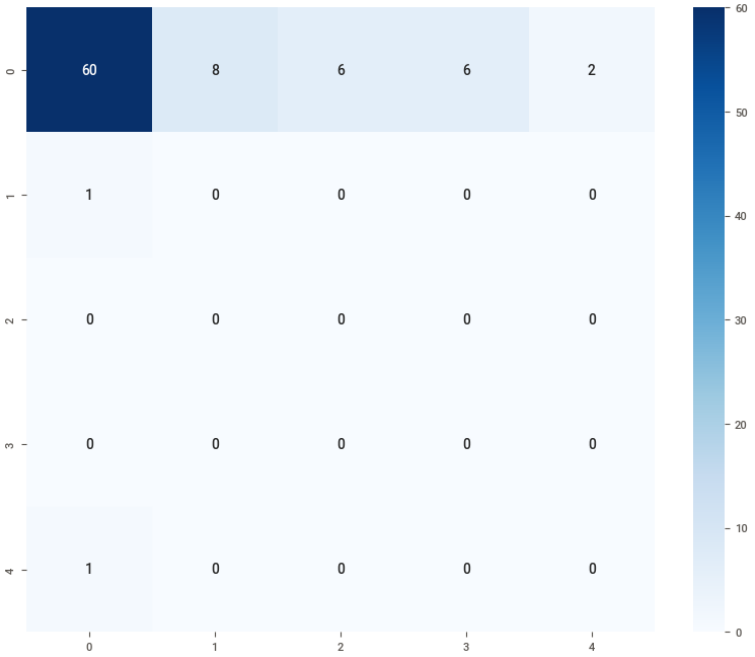
In [302]:
```
# evaluate the keras model
train_loss_ann_smote,train_accuracy_ann_smote = model_ann_smote.evaluate(X_train_smote, Y_train_smo
                                                                         batch_size=8, verbose=1)
val_loss_ann_smote,test_accuracy_ann_smote = model_ann_smote.evaluate(X_test, Y_test_dummy,
                                                                      batch_size=8, verbose=1)


print('Train accuracy: %.2f' % (train_accuracy_ann_smote*100))
print('Test accuracy: %.2f' % (test_accuracy_ann_smote*100))

print('Train loss: %.2f' % (train_loss_ann_smote*100))
print('Val loss: %.2f' % (val_loss_ann_smote*100))
```

```
155/155 [==============================] - 0s 2ms/step - loss: 0.8795 - categorical_accuracy: 0.8397
11/11 [==============================] - 0s 3ms/step - loss: 1.4554 - categorical_accuracy: 0.5595
Train accuracy: 83.97
Test accuracy: 55.95
Train loss: 87.95
Val loss: 145.54
```

Classification Report and Confusion Matrix of the above trained model is as follows:

In [305]:
```
print(f'Classification Report:\n{classification_report(x, Y_test_dummy)}')
```

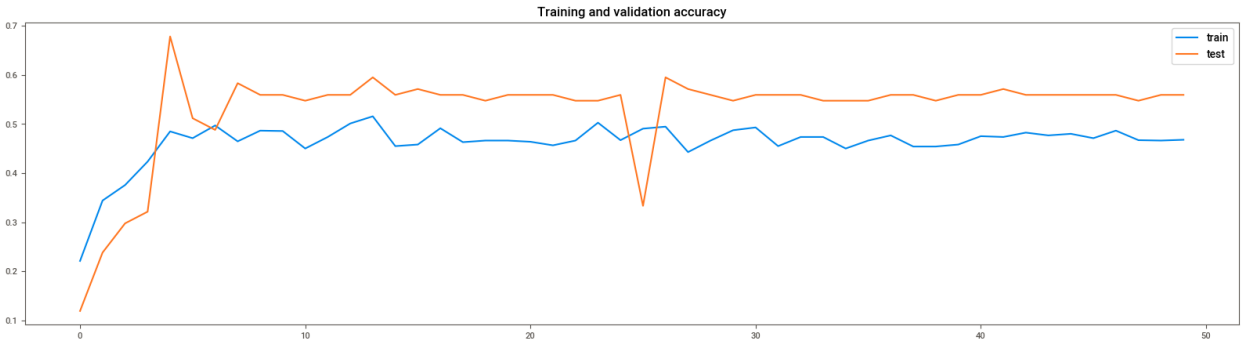
```
Classification Report:
              precision    recall  f1-score   support

           0       0.02      0.50      0.03         2
           1       0.00      0.00      0.00         1
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0
           4       0.00      0.00      0.00         1

   micro avg       0.01      0.25      0.02         4
   macro avg       0.00      0.10      0.01         4
weighted avg       0.01      0.25      0.02         4
 samples avg       0.01      0.01      0.01         4
```



Loss Learning Curves of the above model is as follows:

Above one is overfit model, it can be identified from the learning curve of the training and validation loss only.

Following plot shows Training and Validation Accuracy



Design, Train & Test RNN or LSTM Classifiers:

Creating a Model with Multiple Inputs with TF-IDF Data

The first submodel will accept textual input in the form of accident description. This submodel will consist of an input shape layer, an embedding layer, and bidirectional LSTM layer of 128 neurons followed by max pool layer, drop out and dense layers. The second submodel will accept input in the form of meta information which consists of dense, batch norm and drop out layers.

The output from the dropout layer of the first submodel and the output from the batch norm layer of the second submodel will be concatenated together and will be used as concatenated input to another dense layer with 10 neurons. Finally, the output dense layer will have five neuorns corresponding to each accident level.

Following is the Model Summary

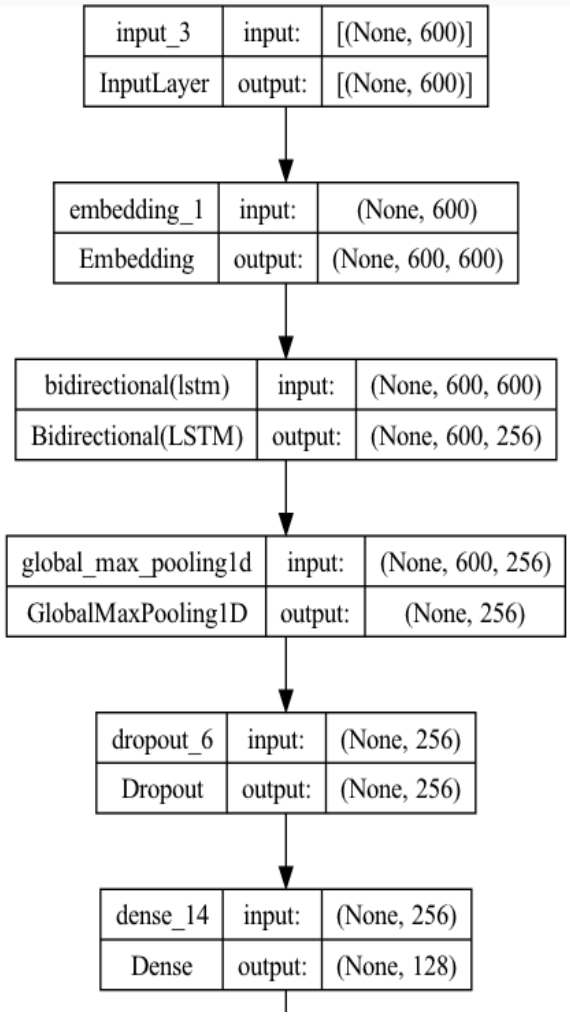
```
In [329]: 1 print(model.summary())
```

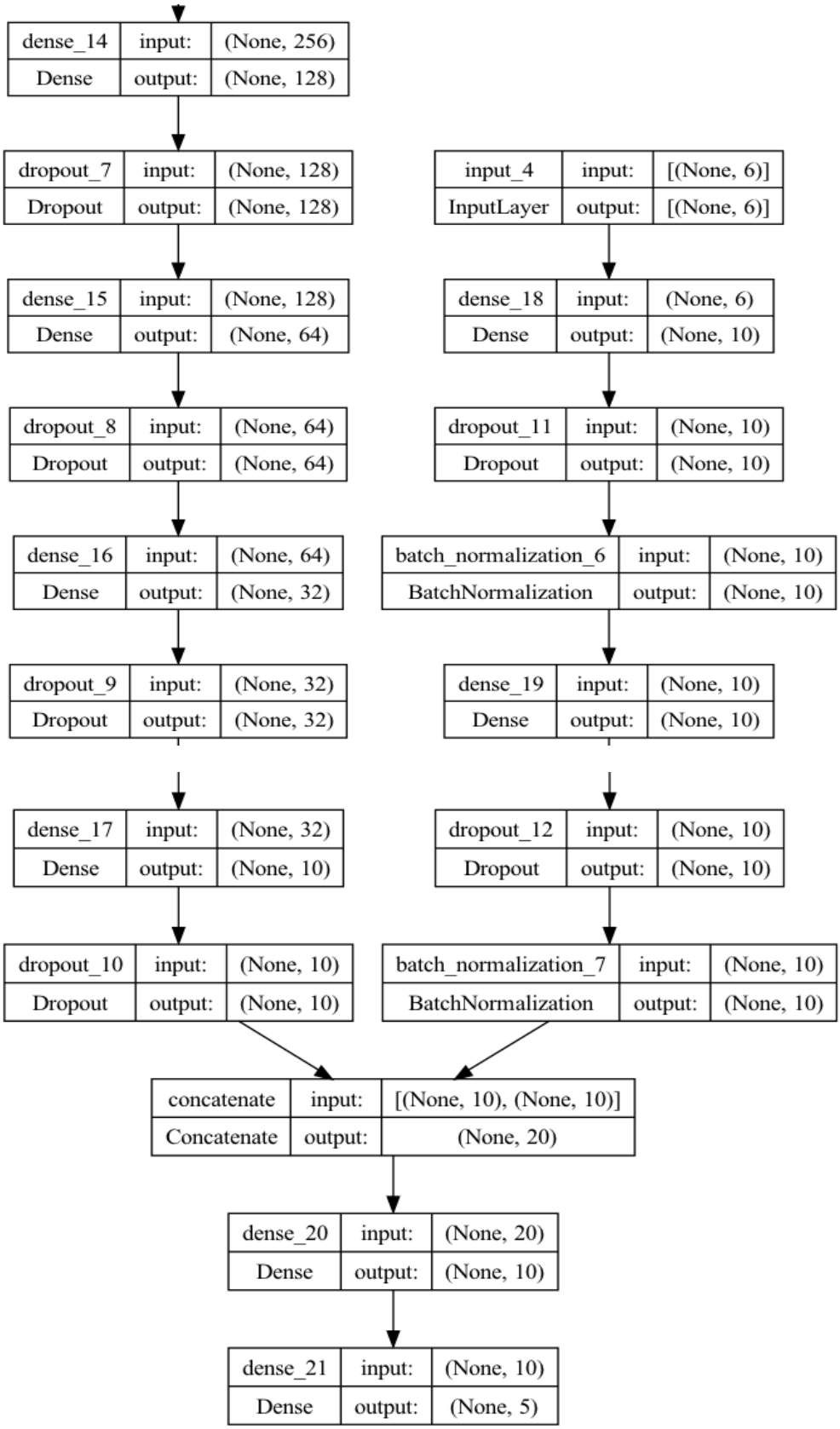
```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_3 (InputLayer)           [(None, 600)]        0           []

 embedding_1 (Embedding)        (None, 600, 600)     741000      ['input_3[0][0]']

 bidirectional (Bidirectional)  (None, 600, 256)     746496      ['embedding_1[0][0]']

 global_max_pooling1d (GlobalMa  (None, 256)         0           ['bidirectional[0][0]']
 xPooling1D)

 dropout_6 (Dropout)            (None, 256)          0           ['global_max_pooling1d[0][0]']

 dense_14 (Dense)               (None, 128)          32896       ['dropout_6[0][0]']

 dropout_7 (Dropout)            (None, 128)          0           ['dense_14[0][0]']

 input_4 (InputLayer)           [(None, 6)]          0           []

 dense_15 (Dense)               (None, 64)           8256        ['dropout_7[0][0]']

 dense_18 (Dense)               (None, 10)           70          ['input_4[0][0]']

 dropout_8 (Dropout)            (None, 64)           0           ['dense_15[0][0]']

 dropout_11 (Dropout)           (None, 10)           0           ['dense_18[0][0]']

 dense_16 (Dense)               (None, 32)           2080        ['dropout_8[0][0]']

 batch_normalization_6 (BatchNo  (None, 10)          40          ['dropout_11[0][0]']
 rmalization)

 dropout_9 (Dropout)            (None, 32)           0           ['dense_16[0][0]']
```

```
dense_19 (Dense)                (None, 10)           110         ['batch_normalization_6[0][0]']

dense_17 (Dense)                (None, 10)           330         ['dropout_9[0][0]']

dropout_12 (Dropout)            (None, 10)           0           ['dense_19[0][0]']

dropout_10 (Dropout)            (None, 10)           0           ['dense_17[0][0]']

batch_normalization_7 (BatchNo  (None, 10)           40          ['dropout_12[0][0]']
rmalization)

concatenate (Concatenate)       (None, 20)           0           ['dropout_10[0][0]',
                                                                  'batch_normalization_7[0][0]']

dense_20 (Dense)                (None, 10)           210         ['concatenate[0][0]']

dense_21 (Dense)                (None, 5)            55          ['dense_20[0][0]']

==================================================================================================
Total params: 1,531,583
Trainable params: 790,543
Non-trainable params: 741,040
__________________________________________________________________________________________________
```

Visualizing the Model Architecture in Plotting as follows

[332]:




dense_14 input: (None, 256)
Dense output: (None, 128)
dropout_7 input: (None, 128)
Dropout output: (None, 128)
dense_15 input: (None, 128)
Dense output: (None, 64)
dropout_8 input: (None, 64)
Dropout output: (None, 64)
dense_16 input: (None, 64)
Dense output: (None, 32)
dropout_9 input: (None, 32)
Dropout output: (None, 32)
dense_17 input: (None, 32)
Dense output: (None, 10)
dropout_10 input: (None, 10)
Dropout output: (None, 10)
input_4 input: [(None, 6)]
InputLayer output: [(None, 6)]
dense_18 input: (None, 6)
Dense output: (None, 10)
dropout_11 input: (None, 10)
Dropout output: (None, 10)
batch_normalization_6 input: (None, 10)
BatchNormalization output: (None, 10)
dense_19 input: (None, 10)
Dense output: (None, 10)
dropout_12 input: (None, 10)
Dropout output: (None, 10)
batch_normalization_7 input: (None, 10)
BatchNormalization output: (None, 10)
concatenate input: [(None, 10), (None, 10)]
Concatenate output: (None, 20)
dense_20 input: (None, 20)
Dense output: (None, 10)
dense_21 input: (None, 10)
Dense output: (None, 5)

After Training the Model, Training and Test Accuracy of the Model is as follows:

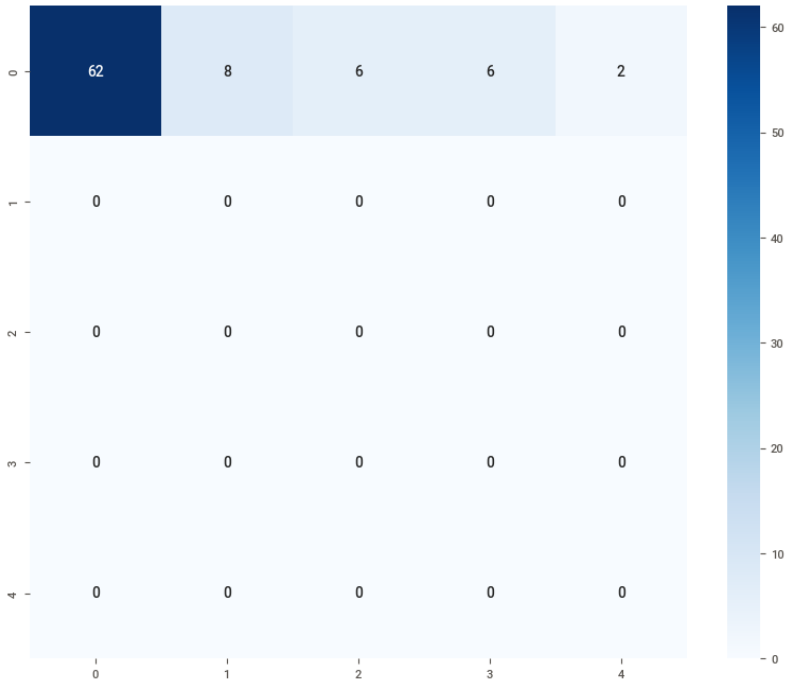
In [334]:
```
# evaluate the keras model
train_loss_lstm_mul_tfidf_input,train_accuracy_lstm_mul_tfidf_input = model_lstm_mul_tfidf
val_loss_lstm_mul_tfidf_input,test_accuracy_lstm_mul_tfidf_input = model_lstm_mul_tfidf_in


print('Train accuracy: %.2f' % (train_accuracy_lstm_mul_tfidf_input*100))
print('Test accuracy: %.2f' % (test_accuracy_lstm_mul_tfidf_input *100))

print('Train loss: %.2f' % (train_loss_lstm_mul_tfidf_input*100))
print('Val loss: %.2f' % (val_loss_lstm_mul_tfidf_input*100))

```

```
155/155 [==============================] - 43s 277ms/step - loss: 1.6401 - acc: 0.2105
11/11 [==============================] - 3s 264ms/step - loss: 1.6046 - acc: 0.1190
Train accuracy: 21.05
Test accuracy: 11.90
Train loss: 164.01
Val loss: 160.46
```

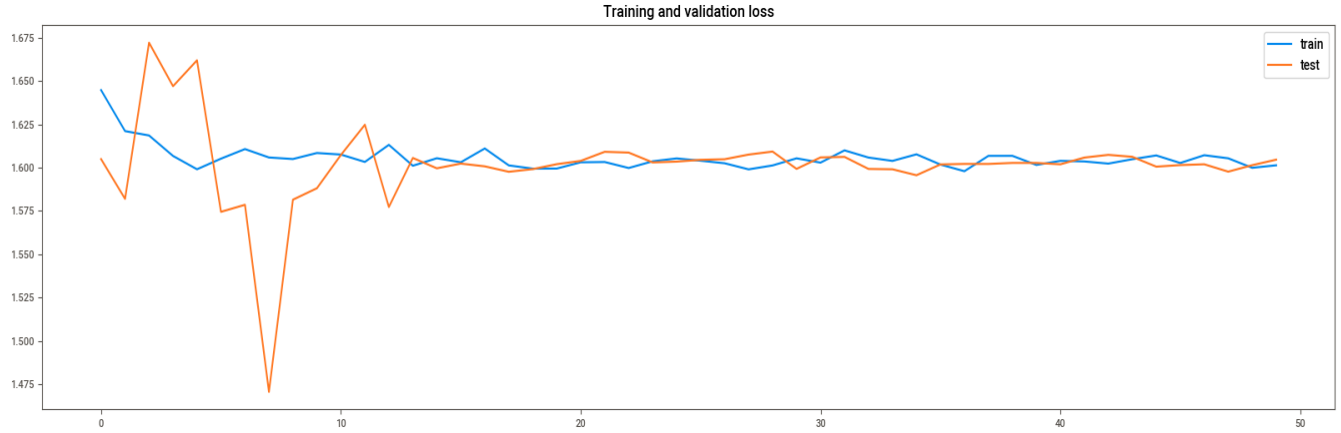
Classification Report and Confusion Matrix of the above trained model is as follows:

In [337]:
```
print(f'Classification Report:\n{classification_report(x, Y_test_dummy)}')
```

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

           0       0.03      1.00      0.06         2
           1       0.00      0.00      0.00         0
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0
           4       0.00      0.00      0.00         0

   micro avg       0.02      1.00      0.05         2
   macro avg       0.01      0.20      0.01         2
weighted avg       0.03      1.00      0.06         2
 samples avg       0.02      0.02      0.02         2
```

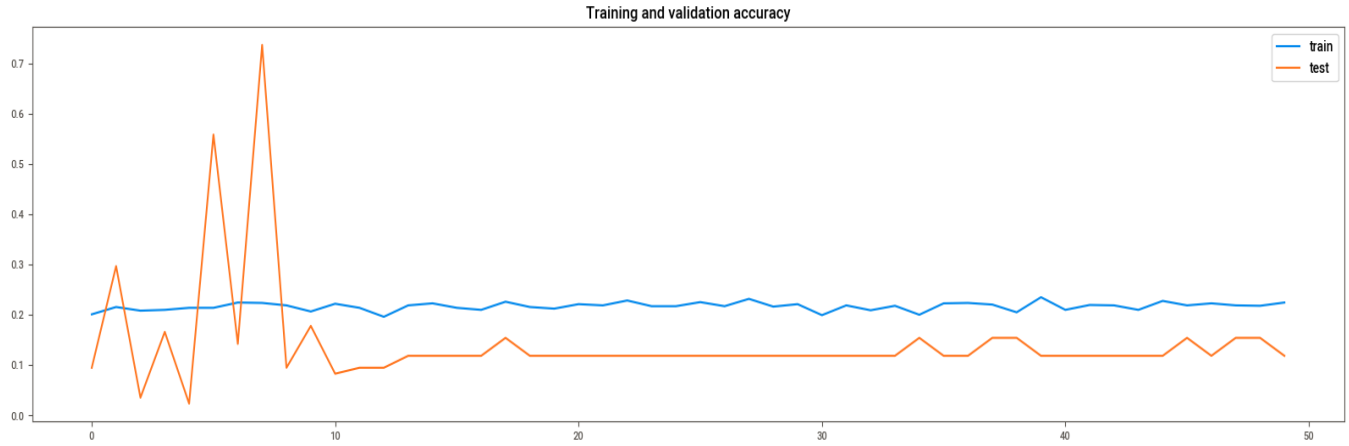


Loss Learning Curves of the above model is as follows:



Above one is good fit, it is identified by a training and validation loss that decreases to a point of stability with a minimal gap between the two final loss values. The loss of the model will almost always be lower on the training dataset than the validation dataset.

Following plot shows Training and Validation Accuracy



We could see it accuracy continually rise during training. As expected, we see the learning curves for accuracy on the test dataset plateau, indicating that the model has no longer overfit the training dataset and it is generalized model.

Creating a Model with Text Inputs Only

In this section, we will create a classification model that uses accident description column alone.

In [343]:
```
# Select input and output features
X_text = df_IndSafety_F_Cpy['Cleaned_Description']
y_text = df_IndSafety_F_Cpy['Accident Level']
```

In [344]:
```
# Encode labels in column 'Accident Level'.
#y_text = LabelEncoder().fit_transform(y_text_Normal)
```

In [345]:
```
# Divide our data into testing and training sets:
X_text_train, X_text_test, y_text_train, y_text_test = train_test_split(X_text, y_text,

print('X_text_train shape : ({0})'.format(X_text_train.shape[0]))
print('y_text_train shape : ({0},)'.format(y_text_train.shape[0]))
print('X_text_test shape : ({0})'.format(X_text_test.shape[0]))
print('y_text_test shape : ({0},)'.format(y_text_test.shape[0]))
```

```
X_text_train shape : (334)
y_text_train shape : (334,)
X_text_test shape : (84)
y_text_test shape : (84,)
```

In [346]:
```
# Convert both the training and test labels into one-hot encoded vectors:
y_text_train = to_categorical(y_text_train)
y_text_test = to_categorical(y_text_test)
```

In [347]:
```
# The first step in word embeddings is to convert the words into thier corre
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(X_text_train)

X_text_train = tokenizer.texts_to_sequences(X_text_train)
X_text_test = tokenizer.texts_to_sequences(X_text_test)
```

In [348]:
```
# Sentences can have different lengths, and therefore the sequences returned
# We need to pad the our sequences using the max length.
vocab_size = len(tokenizer.word_index) + 1
print("vocab_size:", vocab_size)

maxlen = 100

X_text_train = pad_sequences(X_text_train, padding='post', maxlen=maxlen)
X_text_test = pad_sequences(X_text_test, padding='post', maxlen=maxlen)
```

```
vocab_size: 2231
```

In [349]:
```
# We need to load the built-in GloVe word embeddings
embedding_size = 50
embeddings_dictionary = dict()

glove_file = open(r'glove.6B.50d.txt', encoding="utf8")

for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dimensions = np.asarray(records[1:], dtype='float32')
    embeddings_dictionary[word] = vector_dimensions

glove_file.close()

embedding_matrix = np.zeros((vocab_size, embedding_size))

for word, index in tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector

len(embeddings_dictionary.values())
```

Out[349]: 400000

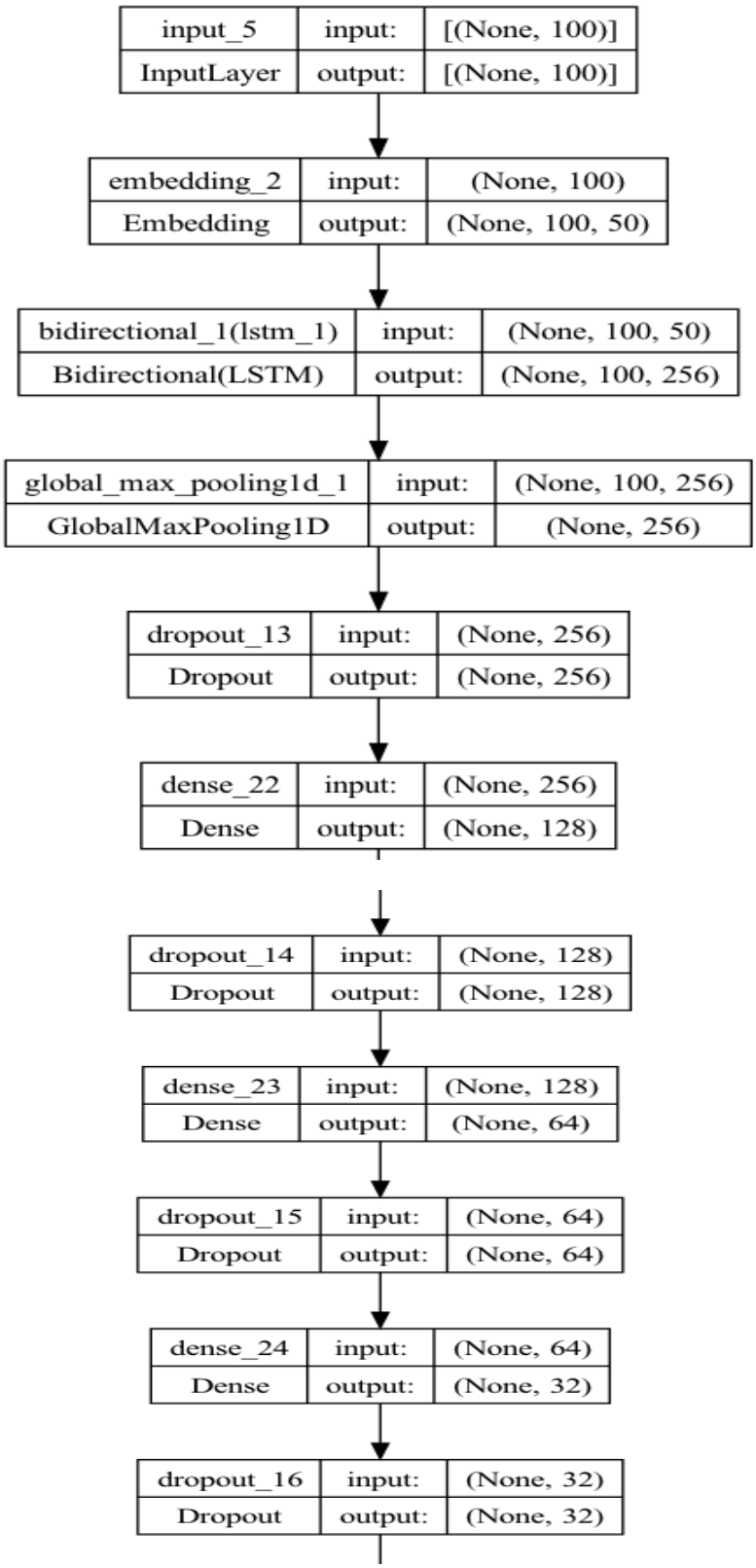
Following is the Model Summary

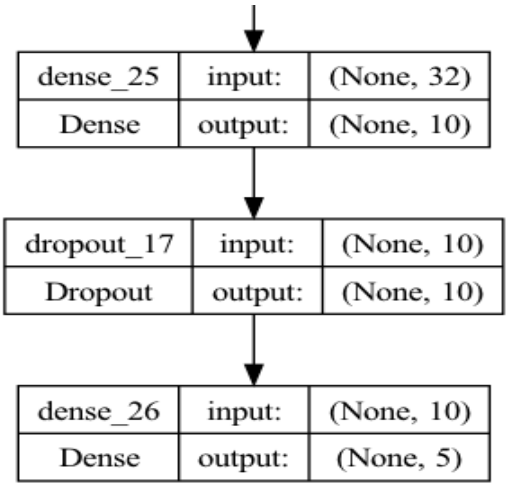
In [351]:
```
print(model.summary())
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_5 (InputLayer)        [(None, 100)]             0

 embedding_2 (Embedding)     (None, 100, 50)           111550

 bidirectional_1 (Bidirectio  (None, 100, 256)         183296
 nal)

 global_max_pooling1d_1 (Glo  (None, 256)              0
 balMaxPooling1D)

 dropout_13 (Dropout)        (None, 256)               0

 dense_22 (Dense)            (None, 128)               32896

 dropout_14 (Dropout)        (None, 128)               0

 dense_23 (Dense)            (None, 64)                8256

 dropout_15 (Dropout)        (None, 64)                0

 dense_24 (Dense)            (None, 32)                2080

 dropout_16 (Dropout)        (None, 32)                0

 dense_25 (Dense)            (None, 10)                330

 dropout_17 (Dropout)        (None, 10)                0

 dense_26 (Dense)            (None, 5)                 55

=================================================================
Total params: 338,463
Trainable params: 226,913
Non-trainable params: 111,550
_________________________________________________________________
```

Visualizing the Model Architecture in Plotting as follows

Out[354]:





Fitting the Above Model with 50 Epochs as follows

In [355]:
```
# fit the keras model on the dataset
training_history_lstm_text_input = model_lstm_text_input.fit(X_text_train, y_text_train,
                                                             epochs=50, batch_size=8, verbose=1,
                                                             validation_data=(X_text_test, y_text_test),
                                                             callbacks=[rlrp, metrics])

```

```
Epoch 1/50

Epoch 48/50
11/11 [==============================] - 1s 56ms/stepss: 0.
— train_f1: 0.739521 — train_precision: 0.739521 — train_recall 0.739521
42/42 [==============================] - 8s 185ms/step - loss: 0.9333 - acc: 0.7395 - val_loss: 0.9215 - val_acc: 0.7
381 - lr: 1.0000e-11
Epoch 49/50
11/11 [==============================] - 1s 81ms/stepss:
— train_f1: 0.739521 — train_precision: 0.739521 — train_recall 0.739521
42/42 [==============================] - 6s 139ms/step - loss: 0.9262 - acc: 0.7395 - val_loss: 0.9215 - val_acc: 0.7
381 - lr: 1.0000e-11
Epoch 50/50
11/11 [==============================] - 1s 79ms/stepss:
— train_f1: 0.739521 — train_precision: 0.739521 — train_recall 0.739521
42/42 [==============================] - 6s 139ms/step - loss: 0.9176 - acc: 0.7395 - val_loss: 0.9215 - val_acc: 0.7
381 - lr: 1.0000e-11
```

Training and Test Accuracy of the Model is as follows:

In [356]:
```
# evaluate the keras model
train_loss_lstm_text_input,train_accuracy_lstm_text_input = model_lstm_text_input.e
val_loss_lstm_text_input,test_accuracy_lstm_text_input = model_lstm_text_input.eval


print('Train accuracy: %.2f' % (train_accuracy_lstm_text_input*100))
print('Test accuracy: %.2f' % (test_accuracy_lstm_text_input *100))

print('Train loss: %.2f' % (train_loss_lstm_text_input*100))
print('Val loss: %.2f' % (val_loss_lstm_text_input*100))
```

```
42/42 [==============================] - 1s 23ms/step - loss: 0.9153 - acc: 0.7395
11/11 [==============================] - 0s 25ms/step - loss: 0.9215 - acc: 0.7381
Train accuracy: 73.95
Test accuracy: 73.81
Train loss: 91.53
Val loss: 92.15
```

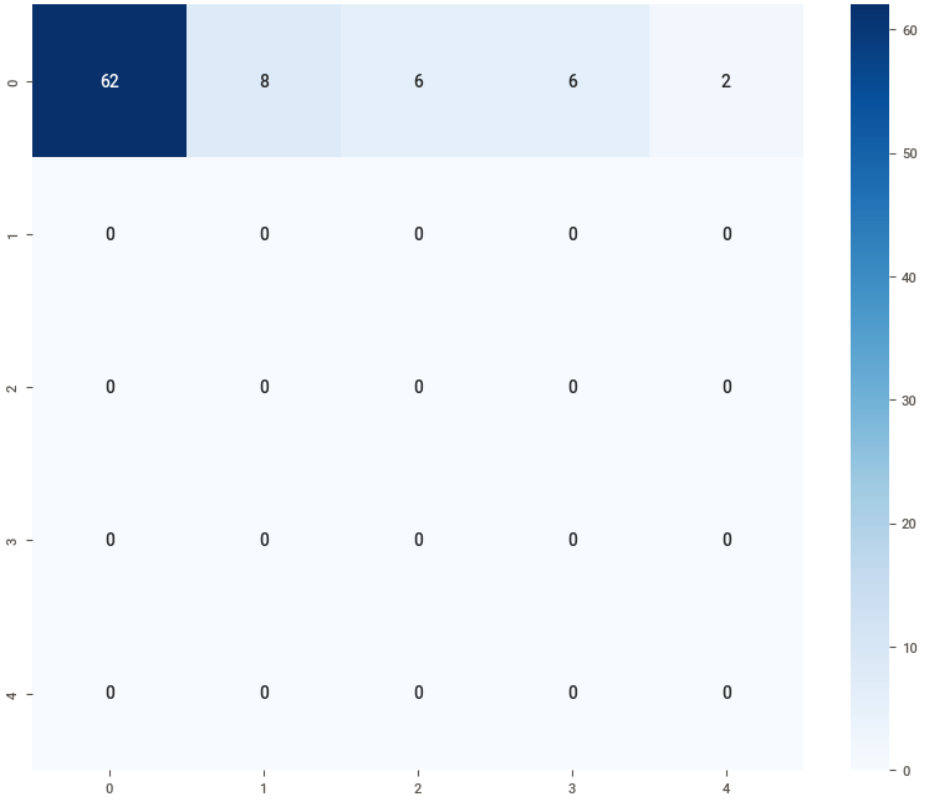
Classification Report and Confusion Matrix of the above trained model is as follows:

```
In [358]:
1 f1_score_lstm_text_input=f1_score(x, y_text_test, average='micro')
2
3 print('f1_score: %.2f' % (f1_score_lstm_text_input*100))
```

```
f1_score: 73.81
```

```
In [359]:
1 print(f'Classification Report:\n{classification_report(x, y_text_test)}')
```

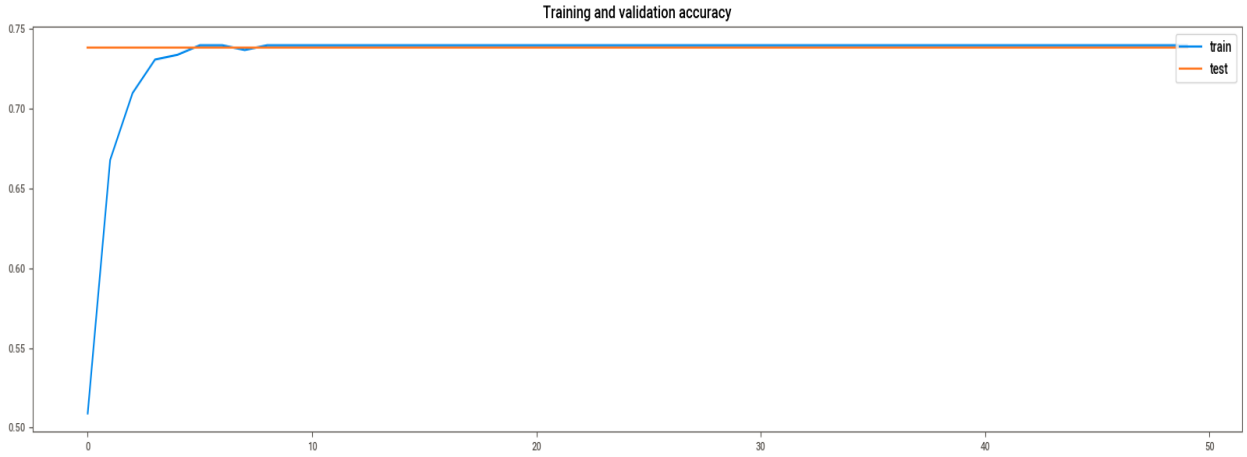
```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.74      0.85        84
           1       0.00      0.00      0.00         0
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0
           4       0.00      0.00      0.00         0

   micro avg       0.74      0.74      0.74        84
   macro avg       0.20      0.15      0.17        84
weighted avg       1.00      0.74      0.85        84
 samples avg       0.74      0.74      0.74        84
```



Loss Learning Curves of the above model is as follows:

Above one is good fit, it is identified by a training and validation loss that decreases to a point of stability with a minimal gap between the two final loss values. The loss of the model will almost always be lower on the training dataset than the validation dataset.

Following plot shows Training and Validation Accuracy



We could see it accuracy continually rise during training. As expected, we see the learning curves for accuracy on the test dataset plateau, indicating that the model has no longer overfit the training dataset and it is generalized model

Note: Surprisingly we observe that same f1-score = 73.81 % with accident description alone..

Creating a Model with Multiple Inputs

The first submodel will accept textual input in the form of accident description. This submodel will consist of an input shape layer, an embedding layer, and bidirectional LSTM layer of 128 neurons followed by max pool layer, drop out and dense layers. The second submodel will accept input in the form of meta information which consists of dense, batch norm and drop out layers.

The output from the dropout layer of the first submodel and the output from the batch norm layer of the second submodel will be concatenated together and will be used as concatenated input to another dense layer with 10 neurons. Finally, the output dense layer will have five neuorns corresponding to each accident level.

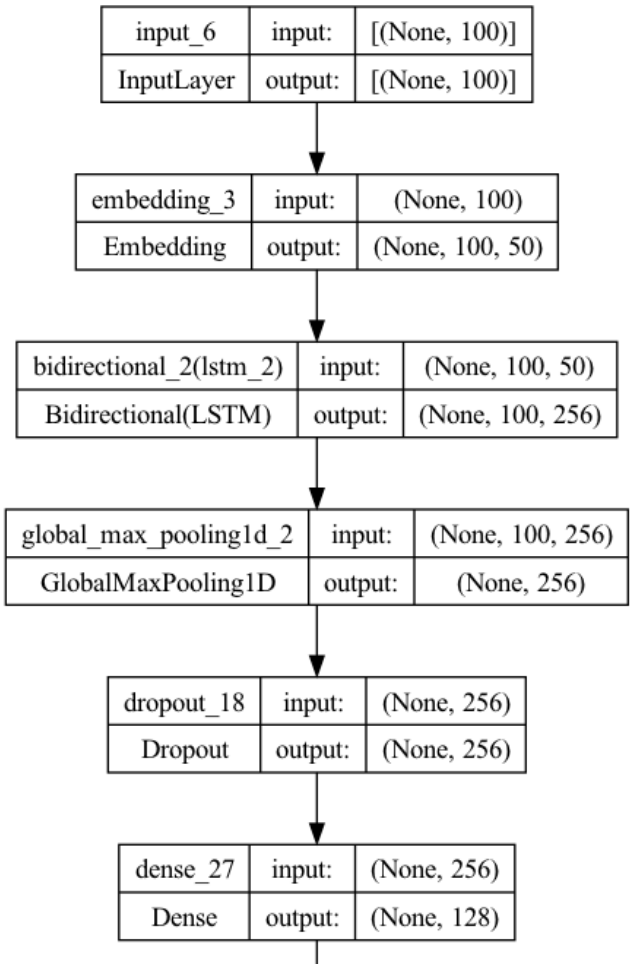
Following is the Model Summary

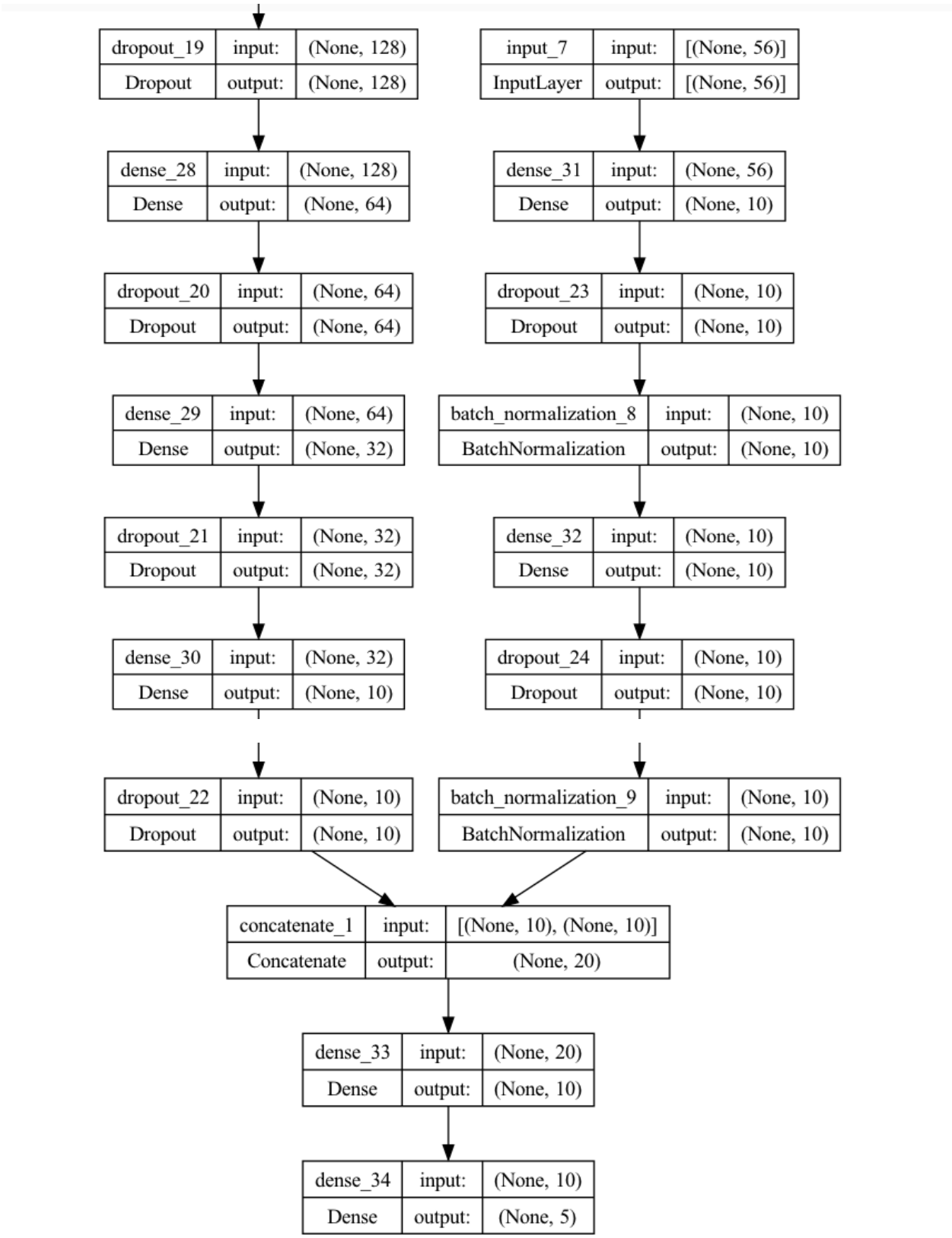
```
In [365]: 1 print(model.summary())
```

```
Model: "model_3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_6 (InputLayer)            [(None, 100)]        0           []

embedding_3 (Embedding)         (None, 100, 50)      111550      ['input_6[0][0]']

bidirectional_2 (Bidirectional  (None, 100, 256)     183296      ['embedding_3[0][0]']
)

global_max_pooling1d_2 (Global  (None, 256)          0           ['bidirectional_2[0][0]']
MaxPooling1D)

dropout_18 (Dropout)            (None, 256)          0           ['global_max_pooling1d_2[0][0]']

dense_27 (Dense)                (None, 128)          32896       ['dropout_18[0][0]']

dropout_19 (Dropout)            (None, 128)          0           ['dense_27[0][0]']

input_7 (InputLayer)            [(None, 56)]         0           []

dense_28 (Dense)                (None, 64)           8256        ['dropout_19[0][0]']

dense_31 (Dense)                (None, 10)           570         ['input_7[0][0]']

dropout_20 (Dropout)            (None, 64)           0           ['dense_28[0][0]']

dropout_23 (Dropout)            (None, 10)           0           ['dense_31[0][0]']

dense_29 (Dense)                (None, 32)           2080        ['dropout_20[0][0]']

batch_normalization_8 (BatchNo  (None, 10)           40          ['dropout_23[0][0]']
rmalization)

dropout_21 (Dropout)            (None, 32)           0           ['dense_29[0][0]']
```

```
dense_32 (Dense)              (None, 10)           110         ['batch_normalization_8[0][0]']

dense_30 (Dense)              (None, 10)           330         ['dropout_21[0][0]']

dropout_24 (Dropout)          (None, 10)           0           ['dense_32[0][0]']

dropout_22 (Dropout)          (None, 10)           0           ['dense_30[0][0]']

batch_normalization_9 (BatchNo  (None, 10)         40          ['dropout_24[0][0]']
rmalization)

concatenate_1 (Concatenate)   (None, 20)           0           ['dropout_22[0][0]',
                                                                 'batch_normalization_9[0][0]']

dense_33 (Dense)              (None, 10)           210         ['concatenate_1[0][0]']

dense_34 (Dense)              (None, 5)            55          ['dense_33[0][0]']

==================================================================================================
Total params: 339,433
Trainable params: 227,843
Non-trainable params: 111,590
__________________________________________________________________________________________________
```

Visualizing the Model Architecture in Plotting as follows




dropout_19 | input: | (None, 128)
Dropout | output: | (None, 128)
input_7 | input: | [(None, 56)]
InputLayer | output: | [(None, 56)]
dense_28 | input: | (None, 128)
Dense | output: | (None, 64)
dense_31 | input: | (None, 56)
Dense | output: | (None, 10)
dropout_20 | input: | (None, 64)
Dropout | output: | (None, 64)
dropout_23 | input: | (None, 10)
Dropout | output: | (None, 10)
dense_29 | input: | (None, 64)
Dense | output: | (None, 32)
batch_normalization_8 | input: | (None, 10)
BatchNormalization | output: | (None, 10)
dropout_21 | input: | (None, 32)
Dropout | output: | (None, 32)
dense_32 | input: | (None, 10)
Dense | output: | (None, 10)
dense_30 | input: | (None, 32)
Dense | output: | (None, 10)
dropout_24 | input: | (None, 10)
Dropout | output: | (None, 10)
dropout_22 | input: | (None, 10)
Dropout | output: | (None, 10)
batch_normalization_9 | input: | (None, 10)
BatchNormalization | output: | (None, 10)
concatenate_1 | input: | [(None, 10), (None, 10)]
Concatenate | output: | (None, 20)
dense_33 | input: | (None, 20)
Dense | output: | (None, 10)
dense_34 | input: | (None, 10)
Dense | output: | (None, 5)

Fitting the Above Model with 50 Epochs as follows

In [369]:
```
# fit the keras model on the dataset
training_history_lstm_mul_input = model_lstm_mul_input.fit([X_text_train, X_cat_train],
                                                           y_cat_train, epochs=50, batch_size=8,
                                                           verbose=1,
                                                           validation_data=([X_text_test, X_cat_test], y_cat_test),
                                                           callbacks=[rlrp, metrics])

```

```
Epoch 1/50


381 - lr: 1.0000e-30
Epoch 46/50
11/11 [==============================] - 1s 65ms/stepss:
— train_f1: 0.739521 — train_precision: 0.739521 — train_recall 0.739521
42/42 [==============================] - 5s 131ms/step - loss: 0.9049 - acc: 0.7395 - val_loss: 0.9213 - val_acc: 0.7
381 - lr: 1.0000e-30
Epoch 47/50
11/11 [==============================] - 1s 64ms/stepss:
— train_f1: 0.739521 — train_precision: 0.739521 — train_recall 0.739521
42/42 [==============================] - 5s 110ms/step - loss: 0.9031 - acc: 0.7395 - val_loss: 0.9204 - val_acc: 0.7
381 - lr: 1.0000e-34
Epoch 48/50
11/11 [==============================] - 1s 64ms/stepss:
— train_f1: 0.739521 — train_precision: 0.739521 — train_recall 0.739521
42/42 [==============================] - 5s 112ms/step - loss: 0.9041 - acc: 0.7395 - val_loss: 0.9208 - val_acc: 0.7
381 - lr: 1.0000e-34
Epoch 49/50
11/11 [==============================] - 1s 66ms/stepss:
— train_f1: 0.739521 — train_precision: 0.739521 — train_recall 0.739521
42/42 [==============================] - 5s 126ms/step - loss: 0.9162 - acc: 0.7395 - val_loss: 0.9215 - val_acc: 0.7
381 - lr: 1.0000e-34
Epoch 50/50
11/11 [==============================] - 1s 67ms/stepss:
— train_f1: 0.739521 — train_precision: 0.739521 — train_recall 0.739521
42/42 [==============================] - 5s 126ms/step - loss: 0.9064 - acc: 0.7395 - val_loss: 0.9226 - val_acc: 0.7
381 - lr: 1.0000e-34
```

Training and Test Accuracy of the Model is as follows:

In [370]:
```
# evaluate the keras model
train_loss_lstm_mul_input,train_accuracy_lstm_mul_input = model_lstm_mul_input.evalu
val_loss_lstm_mul_input,test_accuracy_lstm_mul_input = model_lstm_mul_input.evaluate


print('Train accuracy: %.2f' % (train_accuracy_lstm_mul_input*100))
print('Test accuracy: %.2f' % (test_accuracy_lstm_mul_input *100))

print('Train loss: %.2f' % (train_loss_lstm_mul_input*100))
print('Val loss: %.2f' % (val_loss_lstm_mul_input*100))
```

```
42/42 [==============================] - 1s 24ms/step - loss: 0.8865 - acc: 0.7395
11/11 [==============================] - 0s 30ms/step - loss: 0.9226 - acc: 0.7381
Train accuracy: 73.95
Test accuracy: 73.81
Train loss: 88.65
Val loss: 92.26
```

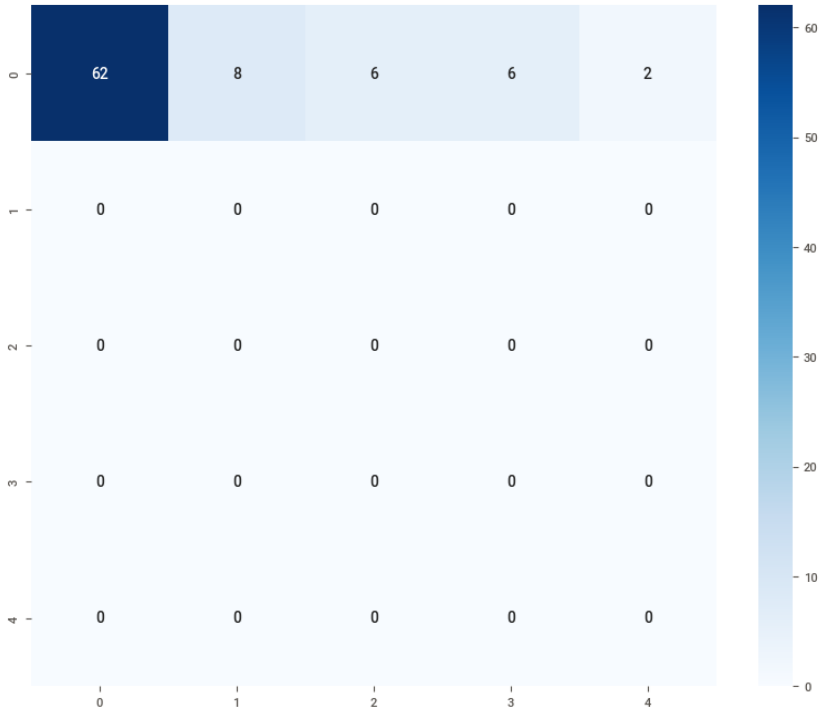
Classification Report and Confusion Matrix of the above trained model is as follows:

In [372]:
```
f1_score_lstm_mul_input=f1_score(x, y_cat_test, average='micro')

print('f1_score: %.2f' % (f1_score_lstm_mul_input*100))
```

```
f1_score: 73.81
```

```
In [373]: 1 print(f'Classification Report:\n{classification_report(x, y_cat_test)}')
```

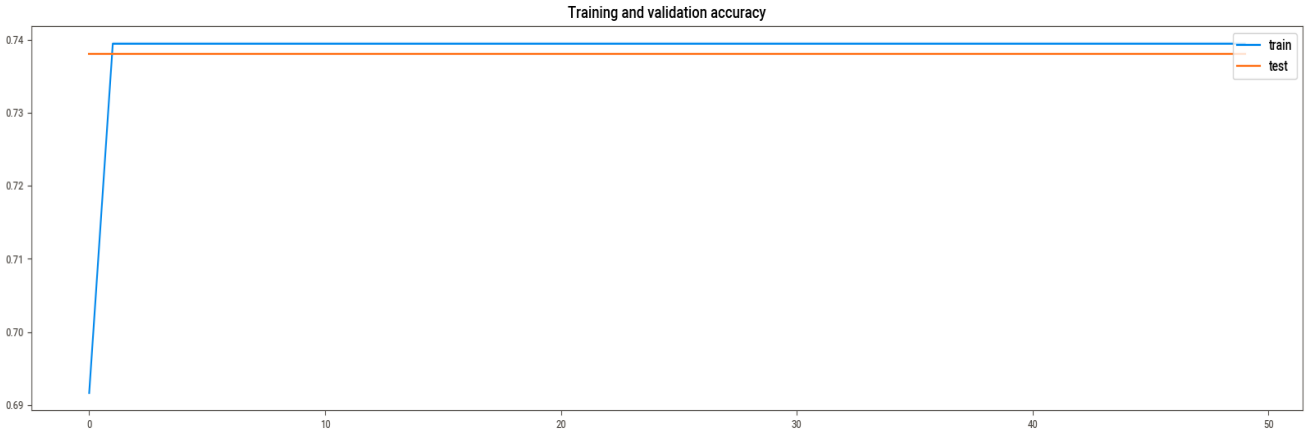
```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.74      0.85        84
           1       0.00      0.00      0.00         0
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0
           4       0.00      0.00      0.00         0

   micro avg       0.74      0.74      0.74        84
   macro avg       0.20      0.15      0.17        84
weighted avg       1.00      0.74      0.85        84
 samples avg       0.74      0.74      0.74        84
```



Loss Learning Curves of the above model is as follows:

Above one is good fit, it is identified by a training and validation loss that decreases to a point of stability with a minimal gap between the two final loss values. The loss of the model will almost always be lower on the training dataset than the validation dataset.

Following plot shows Training and Validation Accuracy



We could see it accuracy continually rise during training. As expected, we see the learning curves for accuracy on the test dataset plateau, indicating that the model has no longer overfit the training dataset and it is generalized model.

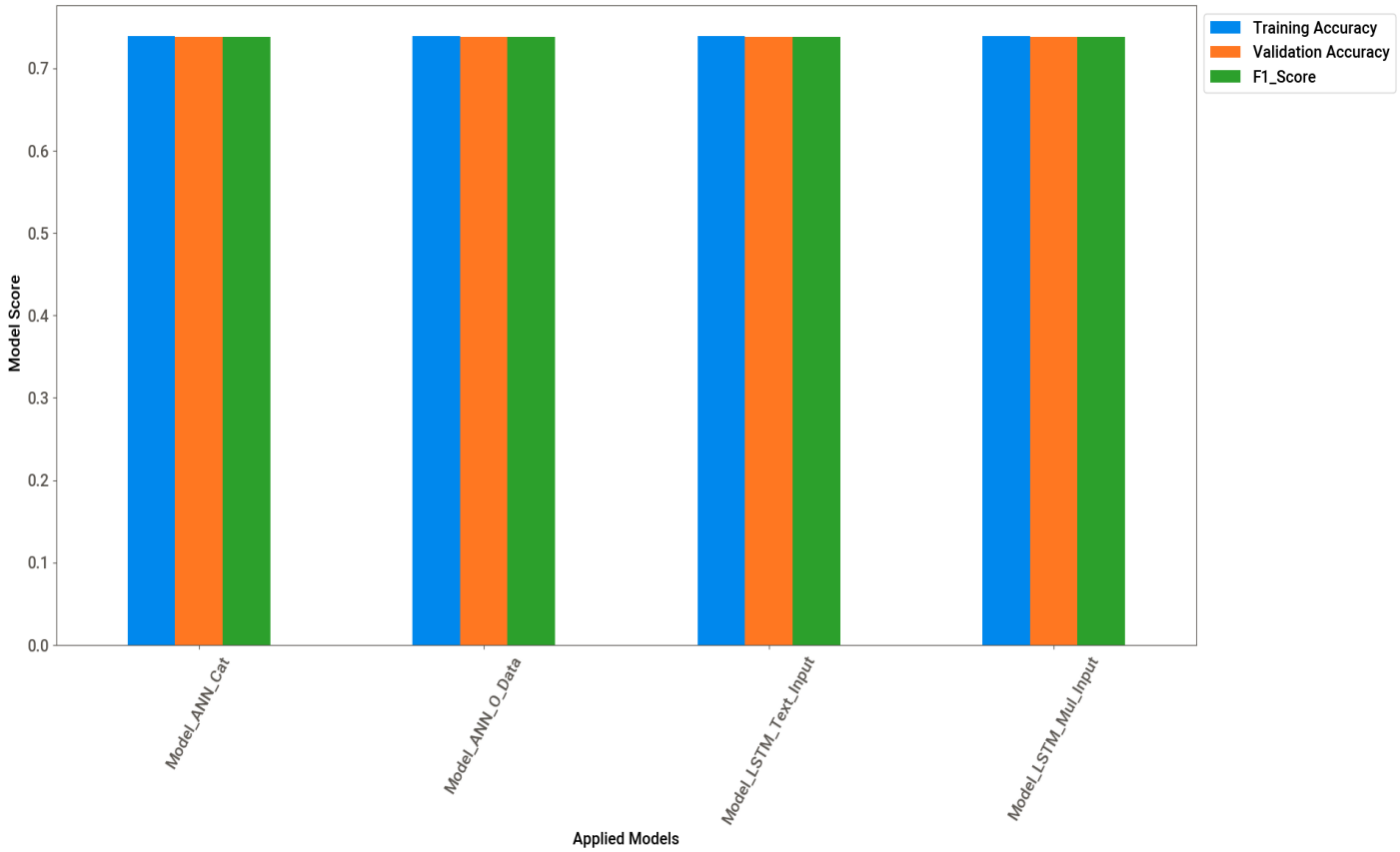
Comparison of Accuracy and F1-Score of All the Applied Models above.

```
In [379]:
1 df_Comparision_graph = Comparision_graph.set_index('Model')
2 df_Comparision_graph
```

Out[379]:

Model	Training Accuracy	Validation Accuracy	F1_Score
Model_ANN_Cat	0.739521	0.738095	0.738095
Model_ANN_O_Data	0.739521	0.738095	0.738095
Model_LSTM_Text_Input	0.739521	0.738095	0.738095
Model_LSTM_Mul_Input	0.739521	0.738095	0.738095

Graphical Representation:



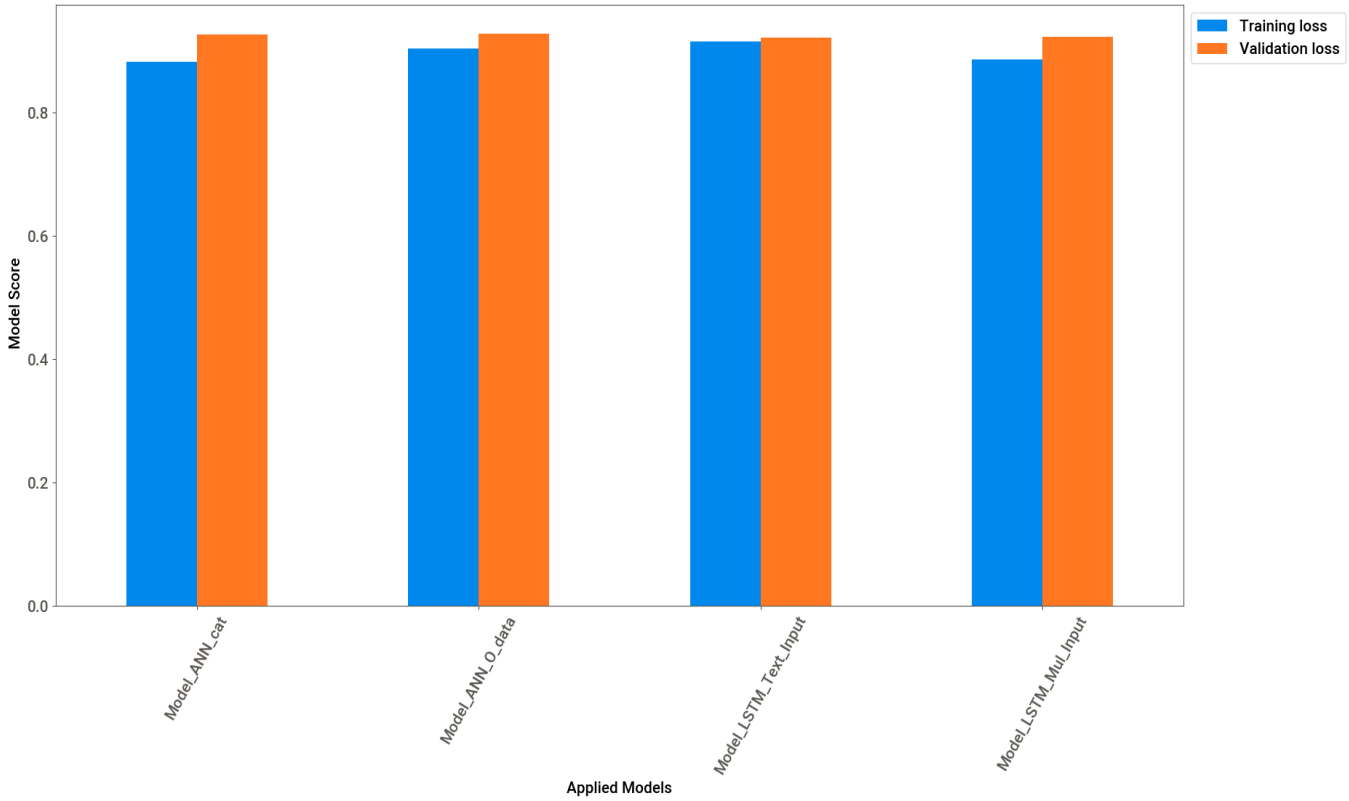
Comparison of Loss of All the Applied Models above.

```
In [382]:
1 df_Comparision_graph = Comparision_graph.set_index('Model')
2 df_Comparision_graph
```

Out[382]:

Model	Training loss	Validation loss
Model_ANN_cat	0.883129	0.927482
Model_ANN_O_data	0.904697	0.928642
Model_LSTM_Text_Input	0.915309	0.921521
Model_LSTM_Mul_Input	0.886474	0.922586

Graphical Representation :



From the above comparision graph it is found that model_lstm_mul_input is best fit for the given dataset and able to predict the accident level with a test accuracy of 73.81% and f1-score of 73.81%, hence we select model_lstm_mul_input for deployment.

Choosing the Best Performing Classifier and Choosing it.

Conclusion:

1. Able to predict the accident level with a test accuracy of 73.81% and f1-score of 73.81%
2. We have seven duplicate values in this dataset and dropped those duplicate values.
3. We have no outliers in this dataset.
4. We have no missing values in this dataset.
5. Extracted the day, month and year from Date column and created new features such as weekday, weekofyear and seasons.

6. Target variable – 'Accident Level' distribution is not equal (I: 309, II: 40, III: 31, IV: 30, V: 8).

7. Class imbalance issue is handled using below methods and found out that, for this particular dataset, with original data we have achieved the better results.

    a. Resampling techniques: Oversampling minority class
    b. SMOTE: Generate synthetic samples

8. By comparing the results from all ML methods with original data, we can select the best method as SVC classifier with f1-score 68.00% with TF-IDF Upsampled data.

9. Explored below options in Neural Networks.

    a. Convert Classification to Numerical problem: achieved a test accuracy of 67% which is a bad result.
    b. Multiclass classification - Target variable - One hot encoded: achieved a test accuracy of 73.81% and f1-score of 73.81% with original data + TF-IDF features from accident description column.
    c. Create a model with Text inputs (accident description alone) only: surprisingly achieved a test accuracy of 73.81% and f1-score of 73.81% with original data.
    d. Create a model with Categorical features only: achieved a test accuracy of 73.81% and f1-score of 72.28% with original data.
    e. Create a model with Multiple Inputs (concatenated the layers from text input model and categorical features input model): surprisingly achieved a test accuracy of 73.81% and f1-score of 73.81% with original data.

10. Finally bidirectional LSTM model can be considered to productionalized model and predict the accident level.

Pickle The Model

In [384]:
```
import pickle

# Save the trained model as a pickle string.
pickle.dump(model_lstm_mul_input, open('model_lstm_mul_input','wb'))

```

Milestone-3: Chatbot Interface

In this Python web-based project with source code, we are going to build a chatbot using deep learning and flask techniques. The chatbot will be trained on the dataset which contains categories (intents), pattern and responses. We use a special artificial neural network (ANN) to classify which category the user's message belongs to and then we will give a random response from the list of responses.

Chatbot using Flask, NLTK, Keras, Python, etc.

Flask:

Flask provides configuration and conventions, with sensible defaults, to get started. This section of the documentation explains the different parts of the Flask framework and how they can be used, customized, and extended. Beyond Flask itself, look for community-maintained extensions to add even more functionality

Flask is a web application framework written in Python. Armin Ronacher, who leads an international group of Python enthusiasts named Pocco, develops it. Flask is based on Werkzeug WSGI toolkit and Jinja2 template engine. Both are Pocco projects

Before you start proceeding with this tutorial, we are assuming that you have hands-on experience on HTML and Python. If you are not well aware of these concepts, then we will suggest you to go through our short tutorials on HTML and Python

The Dataset

The dataset we will be using is 'sample.json'. This is a JSON file that we will create with some usual patterns along with the data which we get from the given problem statement. This JSON File which we create contains the patterns we need to find and the responses we want to return to the user.

Prerequisites

The project requires you to have good knowledge of Python, Keras, and Natural language processing (NLTK), Flask. Along with them, we will use some helping modules which you can download using the python-pip command.

How to Make Chatbot in Python?

Now we are going to build the chatbot using Flask framework but first, let us see the file structure and the type of files we will be creating:

a. sample.json – The data file which has predefined patterns and responses.

b. Texts.pkl – This is a pickle file in which we store the words Python object using Nltk that contains a list of our vocabulary.

c. Labels.pkl – The classes or target variable of the pickle file contains the list of categories(Labels).

d. Final_keras_model_chatbot.h5 – This is the trained model that contains information about the model and has weights of the neurons.

e. app.py – This is the FLASK Python script in which we implemented web-based GUI for our chatbot. Users can easily interact with the bot.

f. index.html – This is the web based html file which is the design of the chatbot which accepts the input and send the input to app.py and get the response and displays the response.

g. Style.css – This file consists of the style of the display of the chatbot html page. HTML Page uses this css file to display the chatbot in an attractive manner according to the design which we give in css file.

Creating the sample.json file.

We are considering the DataFrame mentioned below.

```
In [385]: 1 df_IndSafety_Final
```

Out[385]:

ustry ector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Description	Year	Month_Year	Day_Name	Week_Of_Year	Quarter	Is_Holiday	Cleaned_Description
lining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...	2016	2016-01	Friday	53	1	1	remove drill rod jumbo maintenance supervisor ...
lining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...	2016	2016-01	Saturday	53	1	0	activation sodium sulphide pump pipe uncoupled...
lining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...	2016	2016-01	Wednesday	1	1	0	sub station milpo locate level collaborator ex...

[

```
{"tag": "greeting",
 "patterns": ["Hi there", "How are you", "Is anyone there?","Hey","Hola", "Hello", "Good day"],
 "responses": ["Hello, thanks for asking", "Good to see you again", "Hi there, how can I help?"],
 "context": [""]
},
{"tag": "goodbye",
 "patterns": ["Bye", "See you later", "Goodbye", "Nice chatting to you, bye", "Till next time"],
 "responses": ["See you!", "Have a nice day", "Bye! Come back again soon."],
 "context": [""]
},
{"tag": "thanks",
 "patterns": ["Thanks", "Thank you", "That's helpful", "Awesome, thanks", "Thanks for helping me"],
 "responses": ["Happy to help!", "Any time!", "My pleasure"],
 "context": [""]
},
{"tag": "noanswer",
 "patterns": ["No Answer","Can you please elaborate","More info please","Sorry, can't understand you", "Not sure I understand"],
 "responses": ["Accident level: from I to VI, it registers how severe was the accident where I means not severe but VI means very severe"],
 "context": [""]
},
{"tag": "options",
 "patterns": ["How you could help me?", "What you can do?", "What help you provide?", "How you can be helpful?", "What support is offered"],
 "responses": ["I can guide you through What type of Accident Level you have got into."],
 "context": [""]
}
]
```


We are considering the above some regular patterns to make the chatbot more user friendly. Along with the data present in the given dataset.

Below code will create the JSON file which consists of the description of the dataset along with the above some common patterns.

```
for acc in list(df_IndSafety_Final['Accident Level'].unique()):
    sent = []
    for i in df_IndSafety_Final[df_IndSafety_Final['Accident Level']==acc]['Cleaned_Description']:
        sent.append(i)
    acc_lev = ["Accident Level "+acc]
    list_of_dict.append({"tag" :acc,"patterns": sent,"responses": acc_lev,"context": [""]})
main_json_dict = {"intents":list_of_dict}
main_json_dict
with open("sample.json", "w") as outfile:
    json.dump(main_json_dict, outfile)
```

Final Dataset looks like below.

In [389]:
```
df_final = pd.DataFrame(columns=["tags","text"])
for i in range(pd.DataFrame(main_json_dict['intents'])[["tag","patterns"]].shape[0]):
    tag = pd.DataFrame(main_json_dict['intents'])["tag"][i]
    for j in pd.DataFrame(main_json_dict['intents'])["patterns"][i]:
        row= {'tags':tag,'text':j}
        df_final = df_final.append(row,ignore_index=True)
df_final
```

Out[389]:

	tags	text
0	greeting	Hi there
1	greeting	How are you
2	greeting	Is anyone there?
3	greeting	Hey
4	greeting	Hola
...	...	...
440	V	process loading drill carman pit level operato...
441	V	scoop head rpa cut point cro south unload visu...
442	V	activity change conveyor belt b feed primary m...
443	V	approximately hour change cable z power cell l...
444	V	perform shotcrete cast resane cruise nv p appr...

445 rows × 2 columns

Creating Texts.pkl and Labels.pkl Files:

```
words=[]
classes = []
documents = []
ignore_words = ['?', '!']
data_file = open('sample.json').read()
intents = json.loads(data_file)
lemmatizer=WordNetLemmatizer()

for intent in intents['intents']:
    for pattern in intent['patterns']:

        #tokenize each word
        w = nltk.word_tokenize(pattern)
        words.extend(w)
        #add documents in the corpus
        documents.append((w, intent['tag']))

        # add to our classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])
```

```
# lemmaztize and lower each word and remove duplicates
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]
words = sorted(list(set(words)))
# sort classes
classes = sorted(list(set(classes)))
# documents = combination between patterns and intents
print (len(documents), "documents")
# classes = intents
print (len(classes), "classes", classes)
# words = all words, vocabulary
print (len(words), "unique lemmatized words")


pickle.dump(words,open('texts.pkl','wb'))
pickle.dump(classes,open('labels.pkl','wb'))
```

```
445 documents
10 classes ['I', 'II', 'III', 'IV', 'V', 'goodbye', 'greeting', 'noanswer', 'options', 'thanks']
2497 unique lemmatized words
```

From above we can see that labels.pkl file consists of 10 Classes and texts.pkl consists of the 2497 unique lemmatized words which is pre-processed using NLP techniques.

Creating the final_keras_model_chatbot.h5 Model file:

We are building a sequential model to train the above data and then we save the model after the model is trained with the above data.

```
68 # Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd output laye
69 # equal to number of intents to predict output intent with softmax
70 model = Sequential()
71 model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
72 model.add(Dropout(0.5))
73 model.add(Dense(64, activation='relu'))
74 model.add(Dropout(0.5))
75 model.add(Dense(len(train_y[0]), activation='softmax'))
76
77 # Compile model. Stochastic gradient descent with Nesterov accelerated gradient gives good resu
78 sgd = SGD(learning_rate=0.01,  momentum=0.9, nesterov=True)
79 model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
80
81 #fitting and saving the model
82 hist = model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=5, verbose=1)
83
```

```
Epoch 196/200
89/89 [==============================] - 0s 5ms/step - loss: 0.0580 - accuracy: 0.9730
Epoch 197/200
89/89 [==============================] - 1s 6ms/step - loss: 0.0319 - accuracy: 0.9865
Epoch 198/200
89/89 [==============================] - 0s 5ms/step - loss: 0.0275 - accuracy: 0.9888
Epoch 199/200
89/89 [==============================] - 0s 5ms/step - loss: 0.0273 - accuracy: 0.9865
Epoch 200/200
89/89 [==============================] - 1s 6ms/step - loss: 0.0216 - accuracy: 0.9910
```

After Training the model we are saving the model with the name mentioned above.

In [391]:
```
#model created and saved to file to a disk
model.save('final_keras_model_chatbot.h5', hist)

print("model created")
```

```
model created
```

App.py – Flask Python Script:

Some snippets of the Flask Python Script which accepts the query from the html file and predicts the response and give the output to the html file.

```
import nltk
nltk.download('popular')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import pickle
import numpy as np

from keras.models import load_model
model = load_model('final_keras_model_chatbot.h5')
import json
import random
intents = json.loads(open('sample.json').read())
words = pickle.load(open('texts.pkl','rb'))
classes = pickle.load(open('labels.pkl','rb'))
```

```

from flask import Flask, render_template, request

app = Flask(__name__)
app.static_folder = 'static'

@app.route("/")
def home():
    return render_template("index.html")

@app.route("/get")
def get_bot_response():
    userText = request.args.get('msg')
    return chatbot_response(userText)


if __name__ == "__main__":
    app.run()
```

```
59
60 def getResponse(ints, intents_json):
61     tag = ints[0]['intent']
62     list_of_intents = intents_json['intents']
63     for i in list_of_intents:
64         if(i['tag']== tag):
65             result = random.choice(i['responses'])
66             break
67     return result
68
69 def chatbot_response(msg):
70     ints = predict_class(msg, model)
71     res = getResponse(ints, intents)
72     return res
```

Index.html File:

The index.html file is the webpage which is the web based chatbot which accepts the user query. Some sample snippets of html file is as follows.

The html file should be inside the templates folder.

```
  msgerChat.insertAdjacentHTML("beforeend", msgHTML);
  msgerChat.scrollTop += 500;
}

function botResponse(rawText) {

  // Bot Response
  $.get("/get", { msg: rawText }).done(function (data) {
    console.log(rawText);
    console.log(data);
    const msgText = data;
    appendMessage(BOT_NAME, BOT_IMG, "left", msgText);

  });

}


// Utils
function get(selector, root = document) {
  return root.querySelector(selector);
}

function formatDate(date) {
  const h = "0" + date.getHours();
  const m = "0" + date.getMinutes();

  return `${h.slice(-2)}:${m.slice(-2)}`;
}
```

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>NLP BASED CHATBOT INTERFACE FOR METALS and MINING INDUSTRY</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <link rel="stylesheet" href="{{ url_for('static', filename='styles/style.css') }}">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
</head>

<body>
  <!-- partial:index.partial.html -->
  <section class="msger">
    <header class="msger-header">
      <div class="msger-header-title">
        <i class="fas fa-bug"></i> NLP Chatbot <i class="fas fa-bug"></i>
      </div>
    </header>


  <main class="msger-chat">
    <div class="msg left-msg">
      <div class="msg-img" style="background-image: url(https://image.flaticon.com/icons/svg/327/327779.svg)">

      <div class="msg-bubble">
        <div class="msg-info">
          <div class="msg-info-name">NLP Chatbot</div>
          <div class="msg-info-time">12:45</div>
        </div>

        <div class="msg-text">
          Hi, Welcome to CHATBOT INTERFACE FOR METALS and MINING INDUSTRY! Go ahead and send me a message.
        </div>
      </div>
    </div>

  </main>
```

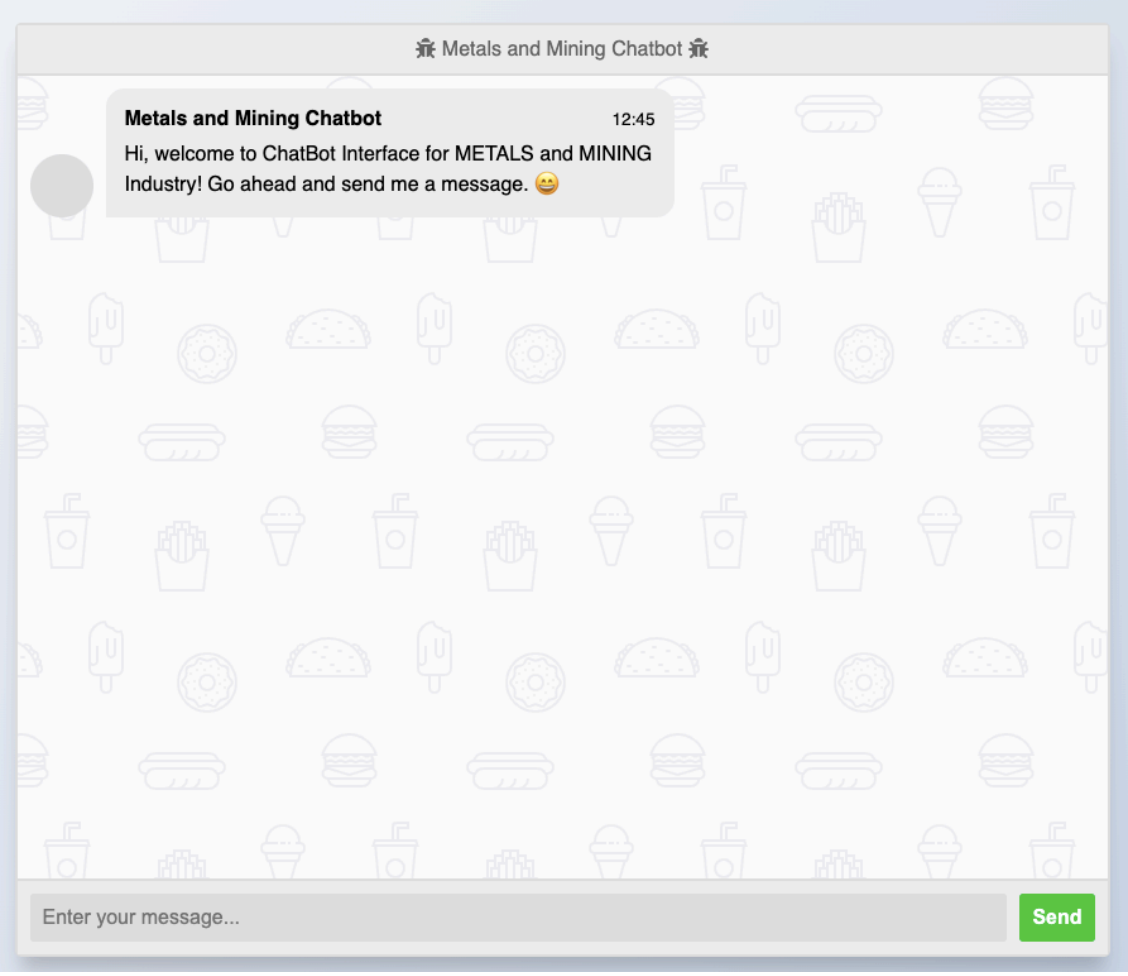
Style.css file : (Cascading Style Sheets File):

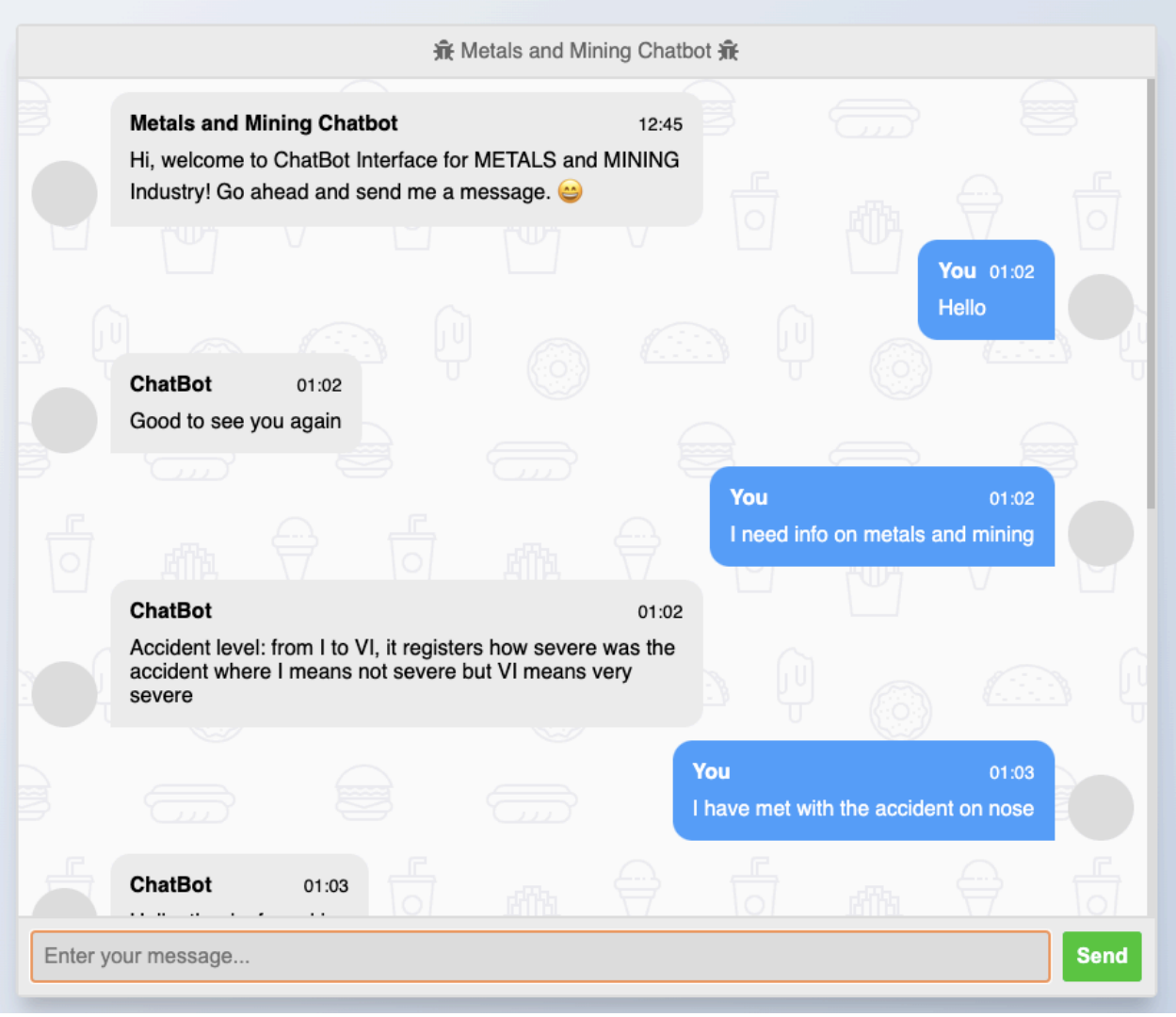
The style.css file should be inside the /static/styles/ folder. Some sample snippets of the style.css file is as follows.

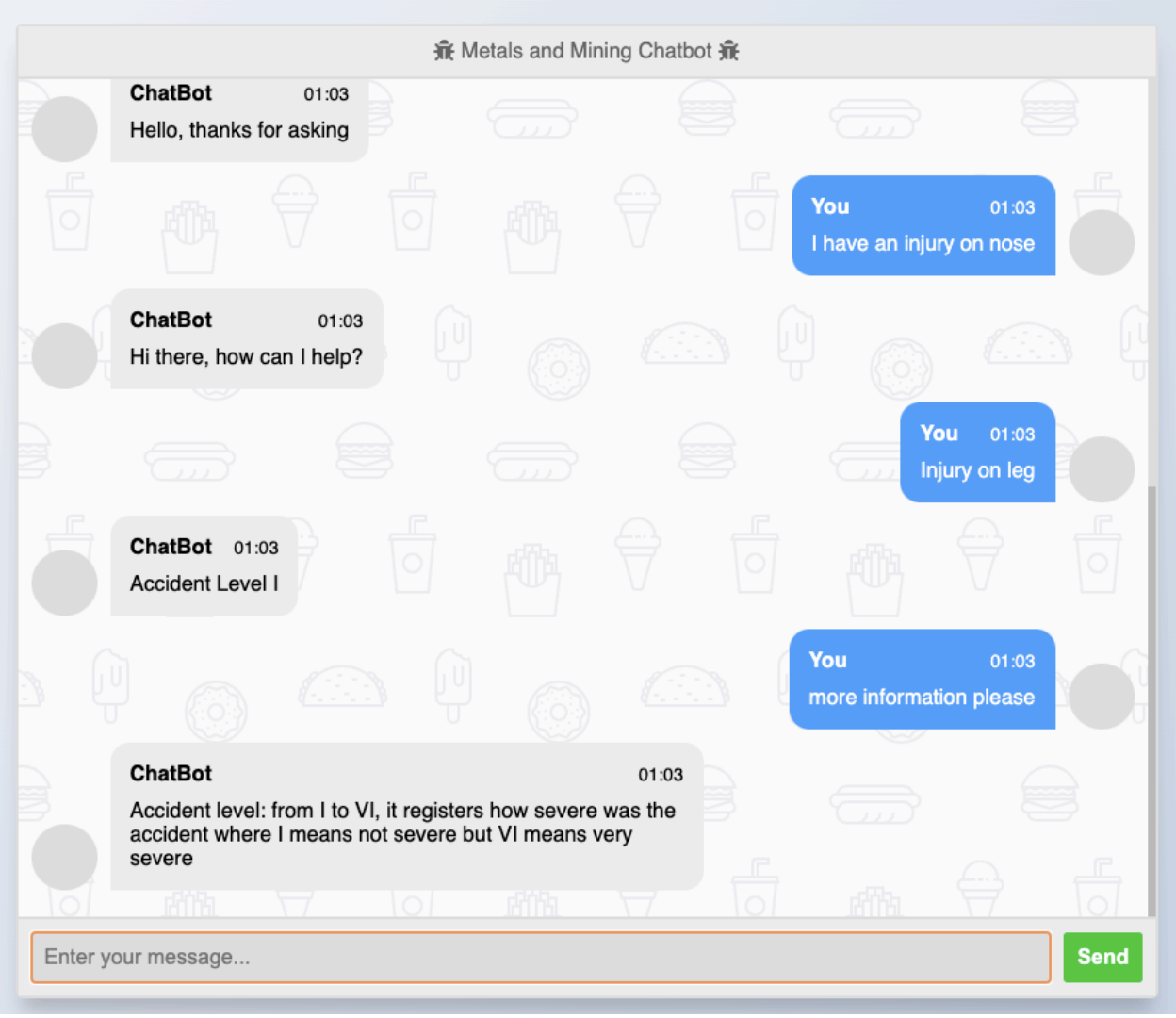
```
:root {
  --body-bg: linear-gradient(135deg, #f5f7fa 0%, #c3cfe2 100%);
  --msger-bg: #fff;
  --border: 2px solid #ddd;
  --left-msg-bg: #ececec;
  --right-msg-bg: #579ffb;
}
```

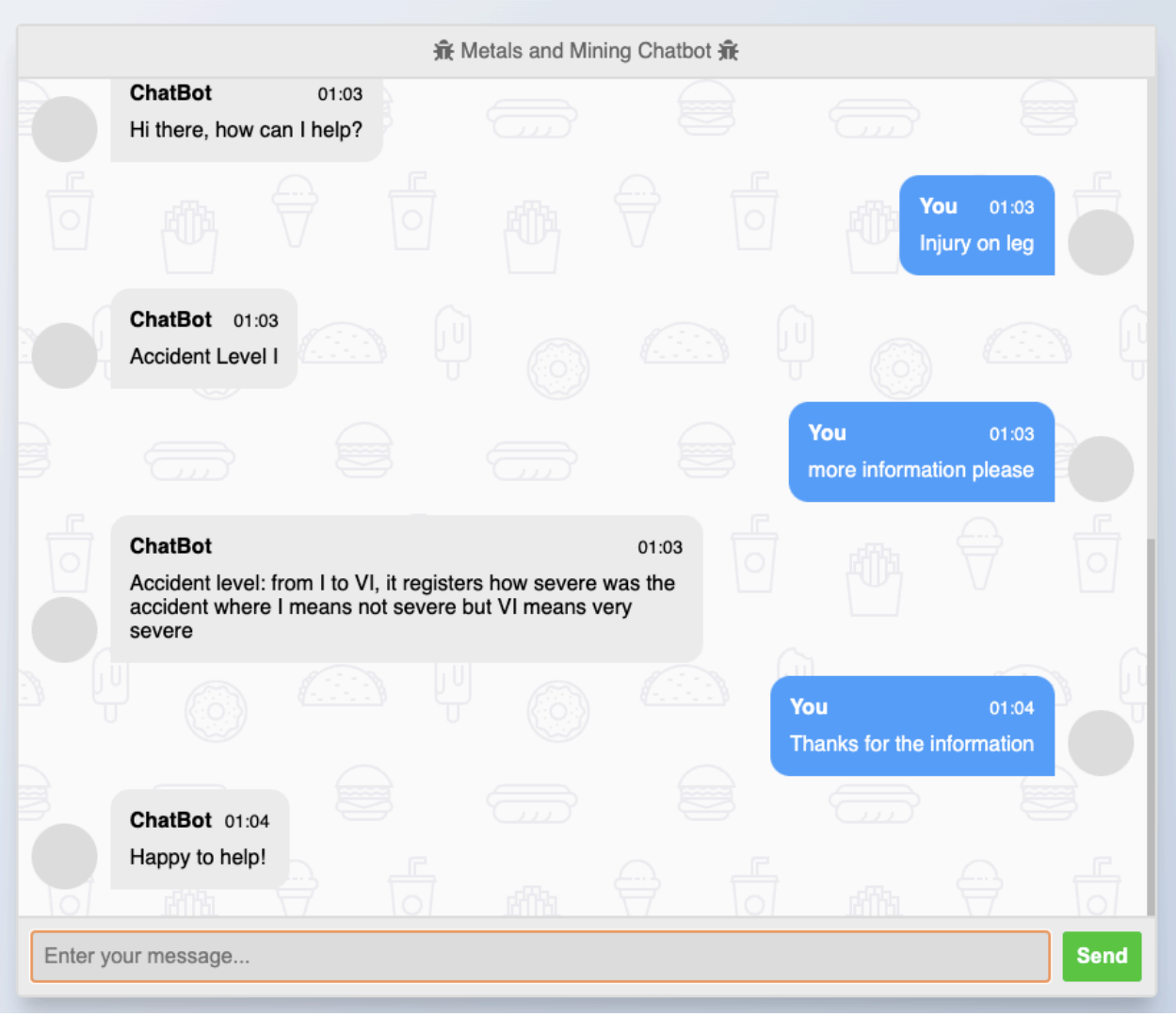
```
body {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  background-image: var(--body-bg);
  font-family: Helvetica, sans-serif;
}


}
.msg {
  display: flex;
  align-items: flex-end;
  margin-bottom: 10px;
}

.msg-img {
  width: 50px;
  height: 50px;
  margin-right: 10px;
  background: #ddd;
  background-repeat: no-repeat;
  background-position: center;
  background-size: cover;
  border-radius: 50%;
}
.msg-bubble {
  max-width: 450px;
  padding: 15px;
  border-radius: 15px;
  background: var(--left-msg-bg);
}
```

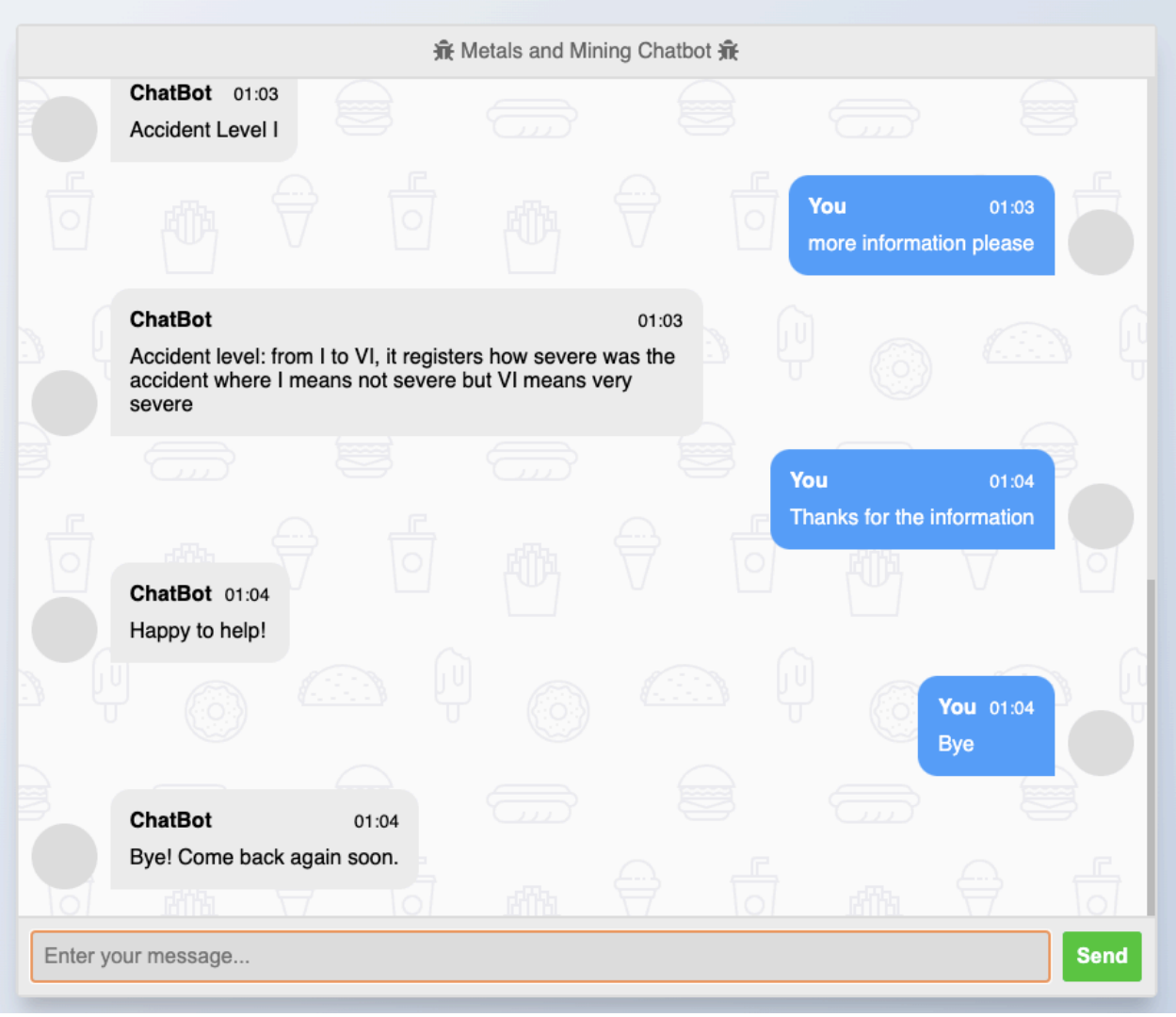
Pictures of the Web-based Chatbot:




Metals and Mining Chatbot
Metals and Mining Chatbot 12:45
Hi, welcome to ChatBot Interface for METALS and MINING Industry! Go ahead and send me a message.
You 01:02
Hello
ChatBot 01:02
Good to see you again
You 01:02
I need info on metals and mining
ChatBot 01:02
Accident level: from I to VI, it registers how severe was the accident where I means not severe but VI means very severe
You 01:03
I have met with the accident on nose
ChatBot 01:03
Enter your message...
Send


Metals and Mining Chatbot
ChatBot 01:03
Hello, thanks for asking
You 01:03
I have an injury on nose
ChatBot 01:03
Hi there, how can I help?
You 01:03
Injury on leg
ChatBot 01:03
Accident Level I
You 01:03
more information please
ChatBot 01:03
Accident level: from I to VI, it registers how severe was the accident where I means not severe but VI means very severe
Enter your message...
Send


Metals and Mining Chatbot
ChatBot 01:03
Hi there, how can I help?
You 01:03
Injury on leg
ChatBot 01:03
Accident Level I
You 01:03
more information please
ChatBot 01:03
Accident level: from I to VI, it registers how severe was the accident where I means not severe but VI means very severe
You 01:04
Thanks for the information
ChatBot 01:04
Happy to help!
Enter your message...
Send



END OF CAPSTONE PROJECT