INDUSTRIAL SAFETY-CHATBOT UTILITY

NLP based Industrial safety Chatbot

Submitted by

Chidanand Pujar

Mrinal Kalita

Thirumurugan S

Date:16/02/2024

Mentor: Jyant Mahara

Table of Contents

Problem Statement

There is a need for industries/Companies around the globe to understand why employees still suffer from injuries/accidents in plants. Sometimes they also die in such environment.

Based on the given description of the incident, it is required to identify the severity of the incident and highlight the safety risk associated with the incident description.

Business value proposition:

System which highlights the safety risk associated with the incident.

Modelling Objective

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

Considering description as independent variable and Potential accident level as target variable.

Dataset

Source of dataset

The database comes from one of the biggest industries in Brazil and in the world.

Dataset Information

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

Columns Description:

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

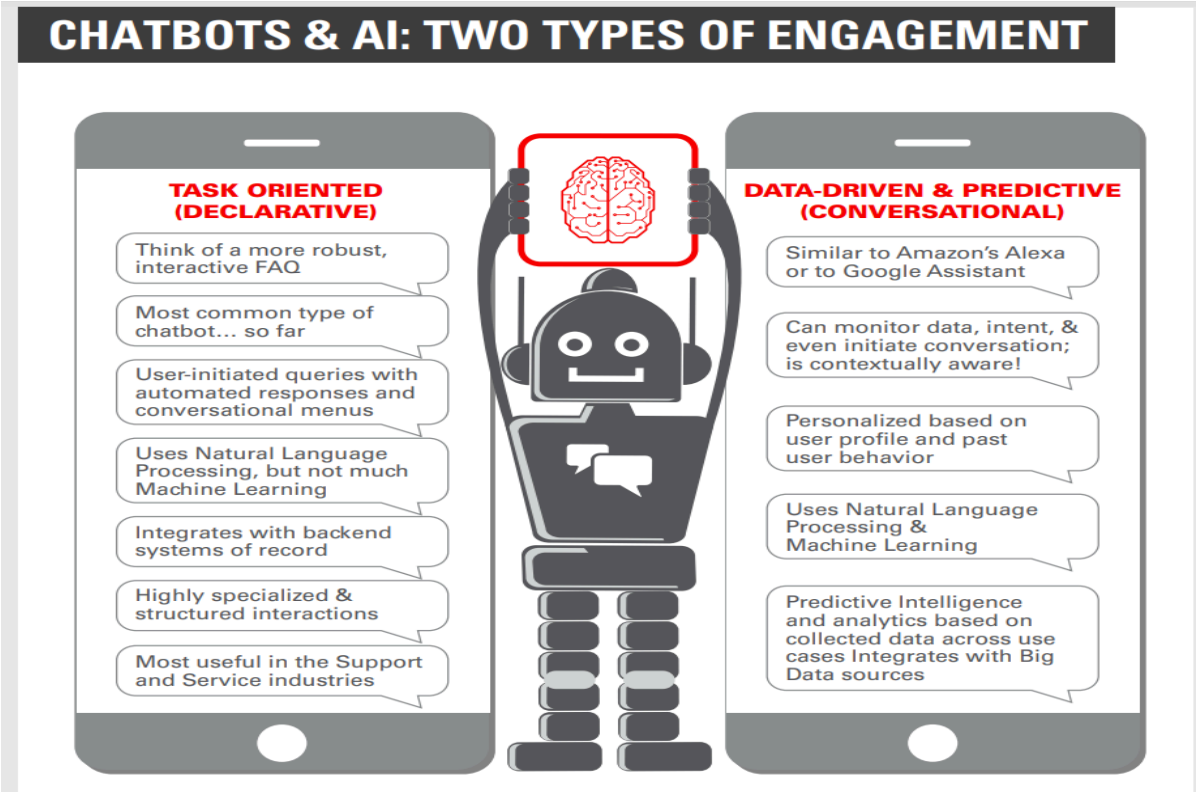
What is a chatbot?

At the most basic level, a chatbot is a computer program that simulates and processes human conversation (either written or spoken), allowing humans to interact with digital devices as if they were communicating with a real person. Chatbots can be as simple as rudimentary programs that answer a simple query with a single-line response, or as sophisticated as digital assistants that learn and evolve to deliver increasing levels of personalization as they gather and process information.

You've probably interacted with a chatbot whether you know it or not. For example, you're at your computer researching a product, and a window pops up on your screen asking if you need help. Or perhaps you're on your way to a concert and you use your smartphone to request a ride via chat. Or you might have used voice commands to order a coffee from your neighbourhood café and received a response telling you when your order will be ready and what it will cost. These are all examples of scenarios in which you could be encountering a chatbot.

How to do chatbots work?

Driven by AI, automated rules, natural-language processing (NLP), and machine learning (ML), chatbots process data to deliver responses to requests of all kinds.



There are two main types of chatbots.

- **Task-oriented (declarative) chatbots** are single-purpose programs that focus on performing one function. Using rules, NLP, and very little ML, they generate automated but conversational responses to user inquiries. Interactions with these chatbots are highly specific and structured and are most applicable to support and service functions—think robust, interactive FAQs. Task-oriented chatbots can handle common questions, such as queries about hours of business or simple transactions that don't involve a variety of variables. Though they do use NLP so end users can experience them in a conversational way, their capabilities are fairly basic. These are currently the most commonly used chatbots.
- **Data-driven and predictive (conversational) chatbots** are often referred to as virtual assistants or digital assistants, and they are much more sophisticated, interactive, and personalized than task-oriented chatbots. These chatbots are contextually aware and leverage natural-language understanding (NLU), NLP, and ML to learn as they go. They apply predictive intelligence and analytics to enable personalization based on user profiles and past user behaviour. Digital assistants can learn a user's preferences over time, provide recommendations, and even anticipate needs. In addition to monitoring data and intent, they can initiate conversations. Apple's Siri and Amazon's Alexa are examples of consumer-oriented, data-driven, predictive chatbots.

Advanced digital assistants are also able to connect several single-purpose chatbots under one umbrella, pull disparate information from each of them, and then combine this information to perform a task while still maintaining context—so the chatbot doesn't become "confused."

Why were chatbots created?

Digitization is transforming society into a "mobile-first" population. As messaging applications grow in popularity, chatbots are increasingly playing an important role in this mobility-driven transformation. Intelligent conversational chatbots are often interfaces for mobile applications and are changing the way businesses and customers interact.

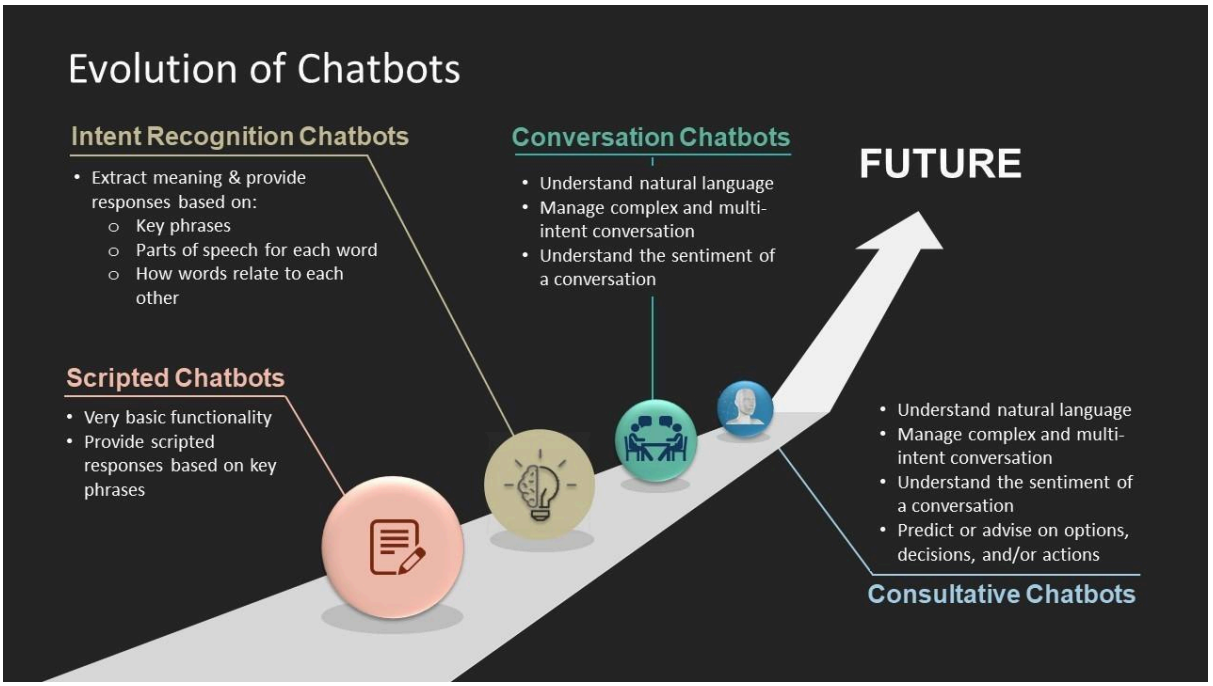
Chatbots allow businesses to connect with customers in a personal way without the expense of human representatives. For example, many of the questions or issues customers have been common and easily answered. That's why companies create FAQs and troubleshooting guides. Chatbots provide a personal alternative to a written FAQ or guide and can even triage questions, including handing off a customer issue to a live person if the issue becomes too complex for the chatbot to resolve. Chatbots have become popular as a time and money saver for businesses and an added convenience for customers.

Evolution of chatbots

The origin of the chatbot arguably lies with Alan Turing's 1950s vision of intelligent machines. Artificial intelligence, the foundation for chatbots, has progressed since that time to include superintelligent supercomputers such as IBM Watson.

The original chatbot was the phone tree, which led phone-in customers on an often cumbersome and frustrating path of selecting one option after another to wind their way through an automated customer service model. Enhancements in technology and the growing sophistication of AI, ML, and NLP evolved this model into pop-up, live, onscreen chats. And the evolutionary journey has continued.

With today's digital assistants, businesses can scale AI to provide much more convenient and effective interactions between companies and customers—directly from customers' digital devices.



Both the benefits and the limitations of chatbots reside within the AI and the data that drive them.

AI considerations: AI is very good at automating mundane and repetitive processes. When AI is incorporated into a chatbot for these types of tasks, the chatbot usually functions well. However, if a demand is made on a chatbot that extends beyond its capabilities or makes its task more complicated, the chatbot might struggle—and that has negative consequences for businesses and customers. There are questions and issues that chatbots simply may not be able to answer or resolve—for example, complex service issues that have a large number of variables.
Developers can work around these limitations by adding a contingency to their chatbot application that routes the user to another resource (such as a live agent) or prompts a customer for a different question or issue. Some chatbots can move seamlessly through transitions between chatbot, live agent, and back again. As AI technology and implementation continue to evolve, chatbots and digital assistants will become more seamlessly integrated into our everyday experience.

Data considerations: All chatbots use data, which is accessed from a variety of sources. As long as the data is high quality and the chatbot is developed correctly, the data will be a chatbot enabler. However, if the data quality is poor, it will limit the chatbot's functionality. And even if the data quality is good, if the chatbot's ML training wasn't modelled properly or is unsupervised, the chatbot can perform poorly—or unexpectedly, at the very least.
In other words, your chatbot is only as good as the AI and data you build into it.

Milestone1

Step1: Import the data

Download and install the unidecode -to automatically convert all Unicode characters to their nearest pure ASCII equivalent.

```
!pip install unidecode
import unidecode
```

Download and install the wordcloud -to use for text analysis

```
!pip install wordcloud
from wordcloud import WordCloud
```

Import necessary libraries to be use in this Project

```
import numpy as np
import pandas as pd
import tensorflow as tf

!pip install unidecode
import unidecode

!pip install wordcloud
from wordcloud import WordCloud

import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords

nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer, SnowballStemmer

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,classification_report

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier
from sklearn.metrics import recall_score, precision_score, f1_score

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Read from the .xlsx the following

- Shape of the data
- Info about the columns-Column name, Datatype

```
[ ]  #Checking the shape
     df.shape
```

```
(425, 11)
```

Dataset has 425 records and 11 columns.

```
[ ]  #Checking the info about columns
     df.info()
```

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

Read the first 5 rows of the class information data frame to understand the structure.

```
#Checking the first 5 records
df.head()
```

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

Data Collection Summary:

- There are 425 rows and 11 columns in the dataset
- Unnamed:0 column is of numeric type and remaining all the columns are of type object
- **Categorical columns:** Countries, Local, Industry Sector, Accident Level, Potential Accident Level, Genre Employee or Third Party, Critical Risk, Description
- **Date column:** Data
- Description is of type text, considered as Input variables or Predictors.
- Accident Level or Potential Accident Level considered as target variable.

Step2: Data Cleansing

We will perform the following in the step of data cleansing

- Dropping of an unnamed column from the dataset
- Checking and removing the duplicate records

```
[ ] #Dropping the Unnamed columns
    df = df.drop('Unnamed: 0', axis=1)
```

```
#Checking the duplicate values
df.duplicated().sum()
```

```
7
```

There are 7 duplicate records.

```
[ ] #Checking the uplicate records
    df[df.duplicated() == True]
```

	Data	Countries	Local	Industry Sector	Accident Level	Potential Accident Level	Genre	Employee or Third Party	Critical Risk	Description
77	2016-04-01	Country_01	Local_01	Mining	I	V	Male	Third Party (Remote)	Others	In circumstances that two workers of the Abrat...
262	2016-12-01	Country_01	Local_03	Mining	I	IV	Male	Employee	Others	During the activity of chuteo of ore in hopper...
303	2017-01-21	Country_02	Local_02	Mining	I	I	Male	Third Party (Remote)	Others	Employees engaged in the removal of material f...
345	2017-03-02	Country_03	Local_10	Others	I	I	Male	Third Party	Venomous Animals	On 02/03/17 during the soil sampling in the re...
346	2017-03-02	Country_03	Local_10	Others	I	I	Male	Third Party	Venomous Animals	On 02/03/17 during the soil sampling in the re...
355	2017-03-15	Country_03	Local_10	Others	I	I	Male	Third Party	Venomous Animals	Team of the VMS Project performed soil collect...
397	2017-05-23	Country_01	Local_04	Mining	I	IV	Male	Third Party	Projection of fragments	In moments when the 02 collaborators carried o...

```
[ ] #Removing the duplicate records
    df.drop_duplicates(inplace=True)
```

- Renaming the column names
- Check for missing values in the dataset

```
[ ] #Renaming the column names
    df.rename(columns={'Data' : 'Date', 'Countries':'Country','Genre':'Gender','Employee or Third Party':'Employee Type'},inplace= True)
```

```
[ ] #Checking for the missing values
    df.isnull().sum()
```

```
Date                      0
Country                   0
Local                     0
Industry Sector           0
Accident Level            0
Potential Accident Level  0
Gender                    0
Employee Type             0
Critical Risk             0
Description               0
dtype: int64
```

There are no missing values in the dataset.

- Deriving new columns from date column-year, month, day & day name.
- After the above Read the first 5 rows to understand the structure after data cleansing

```
[ ] #Deriving new columns from Date column - Year, Month, day, Dayname
    df['Year'] = df['Date'].apply(lambda x:x.year)
    df['Day'] = df['Date'].apply(lambda x:x.day)
    df['Month'] = df['Date'].apply(lambda x:x.month)
    df['Day_name'] = df['Date'].apply(lambda x:x.day_name())
```

```
df.head()
```

	Date	Country	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee Type	Critical Risk	Description	Year	Day	Month	Day_name
0	2016-01-01	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...	2016	1	1	Friday
1	2016-01-02	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...	2016	2	1	Saturday
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	6	1	Wednesday
3	2016-01-08	Country_01	Local_04	Mining	I	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 C...	2016	8	1	Friday
4	2016-01-10	Country_01	Local_04	Mining	IV	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances t...	2016	10	1	Sunday

Data Cleansing Summary:

- Removed Unnamed 0 column from the dataset
- Renamed following columns to give meaningful names data to Date Countries to Country Genre to Gender Employee or Third Party to Employee type
- There were 7 duplicate entries, removed them from dataset
- From Date column, it is observed the accidents entries from 2016-01-01 00:00:00 to 2017-07-09 00:00:00
- There are 3 country types country_001, country_002, country_003 and highest number of incidents reported in country_001 and lowest incidents are reported in country_003
- There are 12 local cities where manufacturing plants are located local_01 to local_12 and highest number of accidents are reported in local_03 and lowest number of accidents are reported in local_09 and local_11
- There are only 3 Industry Sectors and highest number of accidents are reported from Mining industry sector and lowest number of accidents are reported from others
- There are only 5 Accident Levels, ranging from lowest severity I to highest severity V and highest number of accidents recorded are of severity Low I and lowest number of accidents recorded are of severity high V.
- There are only 6 Potential Accident Levels, ranging from lowest severity I to highest severity VI and highest number Potential accidents are reported with severity low I and Lowest number of Potential accidents are reported with severity high VI.
- There are 2 Genders male and female and highest number of accidents are recorded for male Gender and lowest of accidents are recorded for female gender
- There are 3 employee types Third Party, Employee and Third Party (Remote) and highest of accidents are recorded for Third Party and lowest number of accidents are recorded for Third Party (Remote).
- There are quite a lot of Critical Risks reported and highest number of accidents are recorded of type others Critical Risk.
- No missing values in the dataset

- After Data Cleansing, we have now 418 rows and 10 columns

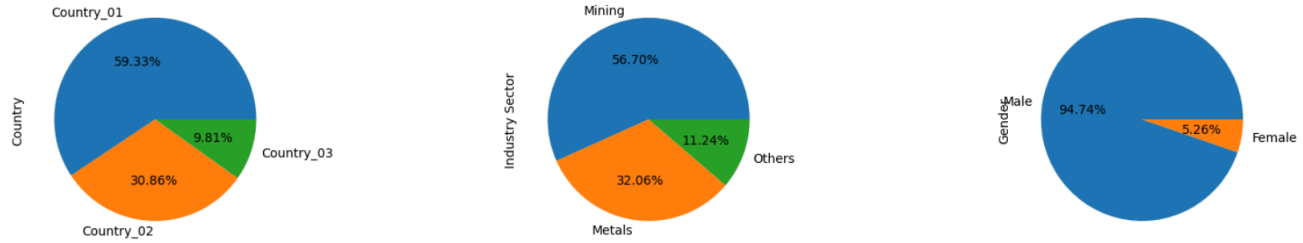
Exploratory Data Analysis (EDA):

- Doing a Univariate analysis to check accident distribution in country, local & industry sector column

Univariate Analysis

```
[ ] #Checking the accident distribution in Country, Local and industry sector column
    plt.figure(figsize=(20,8))
    plt.subplot(2,3,1)
    df['Country'].value_counts().plot(kind='pie',autopct='%.2f%%')
    plt.subplot(2,3,2)
    df['Industry Sector'].value_counts().plot(kind='pie',autopct='%.2f%%')
    plt.subplot(2,3,3)
    df['Gender'].value_counts().plot(kind='pie',autopct='%.2f%%')
```

```
<Axes: ylabel='Gender'>
```



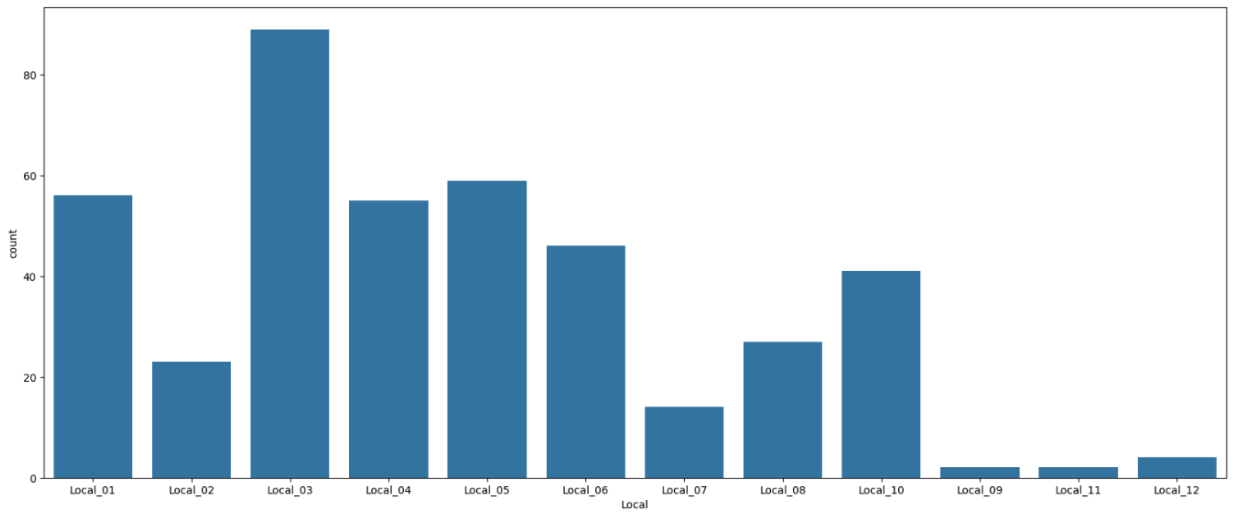
Inference from the univariate analysis:

a. Majority of the accident happens in Country_01.
b. Mining industry has highest accident.
c. 94.74% of the accident happens in male.

- Checking the distribution of local.

```
#Checking the distribution of Local
plt.figure(figsize=(20,8))
sns.countplot(data = df, x = 'Local')
```

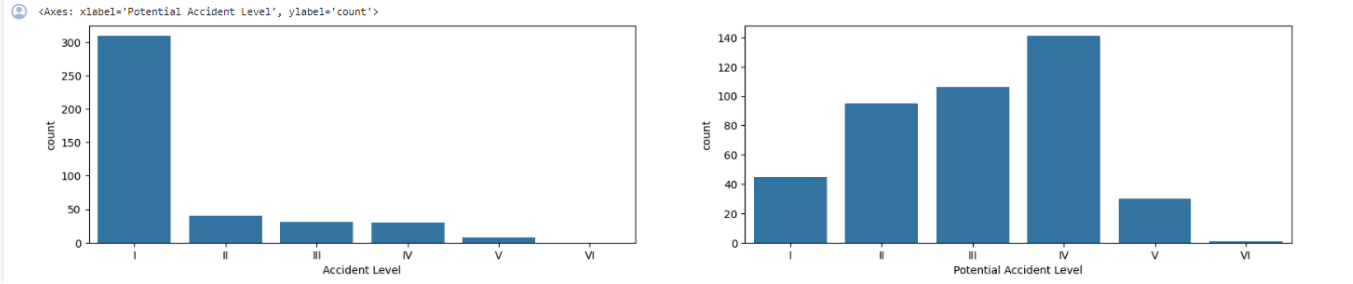
```
<Axes: xlabel='Local', ylabel='count'>
```



- Checking the distribution of target variable.

```
#Checking the distribution of Target variable
plt.figure(figsize=(20,8))
plt.subplot(2,2,1)
sns.countplot(data = df, x = 'Accident Level',order=['I','II','III','IV','V','VI'])
plt.subplot(2,2,2)
sns.countplot(data = df, x = 'Potential Accident Level',order=['I','II','III','IV','V','VI'])
```

```
<Axes: xlabel='Potential Accident Level', ylabel='count'>
```



Inference from the checking of local and target variable:

a. Local-03 has the highest accident
b. Majority of the accident is of level 1.
c. No of accident decreases as severity increases.
d. There are very few records for accident level VI and potential accident level VI. So, we can merge accident level VI and potential accident level VI with V.

Replacing Accident Level VI with V

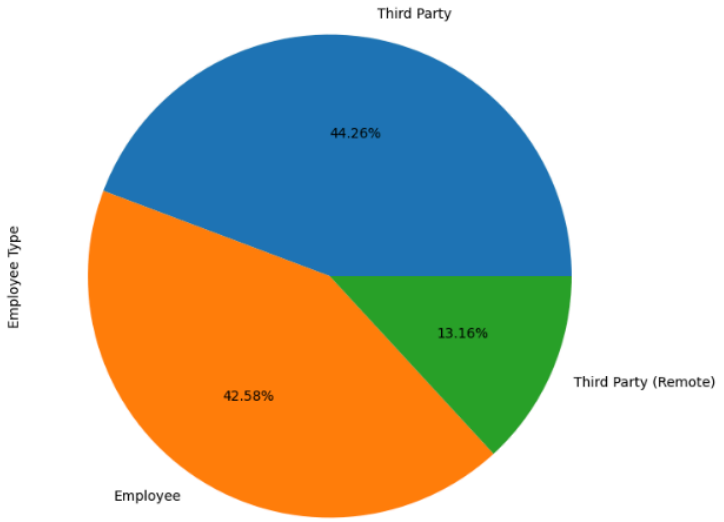
```
#Replacing Accident Level VI with V
df['Accident Level'] = df['Accident Level'].replace('VI','V')
df['Potential Accident Level'] = df['Potential Accident Level'].replace('VI','V')
```

We would like to check the various distributions

- Checking the accident distribution by Employee Type

```
#Checking the accident distribution Employee Type
plt.figure(figsize=(10,8))
df['Employee Type'].value_counts().plot(kind='pie',autopct='%.2f%%')
```

```
<Axes: ylabel='Employee Type'>
```

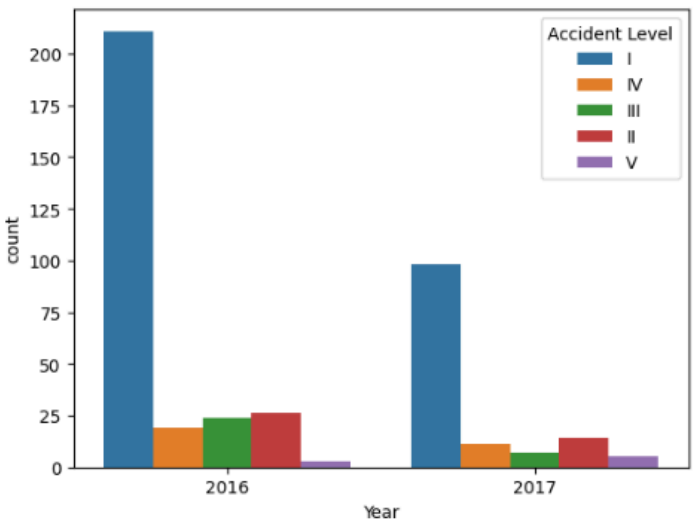


a. 44.26% accidents happen in Third party.
b. 42.585 accidents happen in Employee.
c. 13.16% accidents happen in Third Party (Remote).

- Checking the accident level distribution by year

```
#Checking the distribution of Year
sns.countplot(data = df, x = 'Year',hue='Accident Level')
```

```
<Axes: xlabel='Year', ylabel='count'>
```

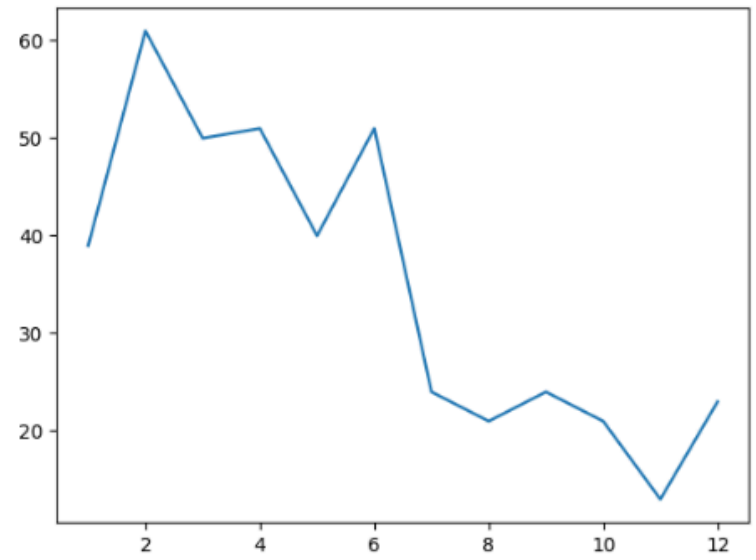


a. Year 2016 had more accident than 2017

- Checking the accident distribution with respect to Month

```
#Checking the distribution of Month
plt.plot(df['Month'].value_counts().sort_index())
```

```
[<matplotlib.lines.Line2D at 0x7ad4a64a00d0>]
```

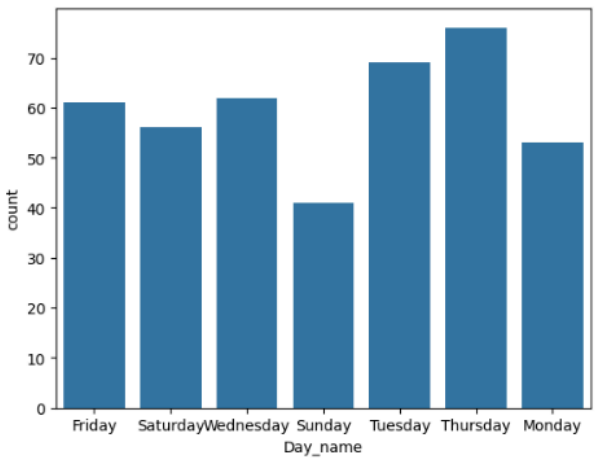


a. Highest accident happened in February.
b. November had lowest no of accident.

- Checking the distribution of by Weekdays

```
#Checking the distribution of Day_name
sns.countplot(data = df, x = 'Day_name')
```

```
<Axes: xlabel='Day_name', ylabel='count'>
```



EDA Summary:

- There are 3 country types country_001, country_002, country_003 and highest number of incidents reported in country_001 and lowest incidents are reported in country_003
- There are 12 local cities where manufacturing plants are located local_01 to local_12 and highest number of accidents are reported in local_03 and lowest number of accidents are reported in local_09 and local_11

- There are only 3 Industry Sectors and highest number of accidents are reported from Mining industry sector and lowest number of accidents are reported from others
- There are only 5 Accident Levels, ranging from lowest severity I to highest severity V and highest number of accidents recorded are of severity Low I and lowest number of accidents recorded are of severity high V.
- There are only 6 Potential Accident Levels, ranging from lowest severity I to highest severity VI and highest number Potential accidents are reported with severity low I and Lowest number of Potential accidents are reported with severity high VI.
- There are 2 Genders male and female and highest number of accidents are recorded for male Gender and lowest of accidents are recorded for female gender
- There are 3 employee types Third Party, Employee and Third Party (Remote) and highest number of accidents are recorded for Third Party and lowest number of accidents are recorded for Third Party (Remote).
- Most of the Critical Risks are classified as others
- Highest number of accidents are reported in the year 2016 and lowest in the year 2017

Step3: Data Preprocessing

- Setting up Feature and target variable

```
#Feature and Target variable
x = df['Description']
y = df['Accident Level']
```

- Removing accented characters, special characters and lowercasing all the letters

Removing Accented Chracters

```
[ ]  def rmv_uni(sen):
       wrds = sen.split()
       new_sen = [unidecode.unidecode(w) for w in wrds]
       new_sen = ' '.join(new_sen)

       return new_sen
```

```
     x = x.apply(lambda x: rmv_uni(x))
```

Removing Special Characters

```
[ ]  def rmv_schar(s):
       wrds = s.split()
       new_text = [w for w in wrds if w.isalnum()]
       new_text = ' '.join(new_text)

       return new_text
```

```
[ ]  x = x.apply(lambda x: rmv_schar(x))
```

Lowercasing All The Letters

```
[ ]  def lower_case(s):
       s = s.lower()
       return s
```

```
[ ]  x = x.apply(lambda x: lower_case(x))
```

```
[ ]  x
```

```
0      while removing the drill rod of the jumbo 08 f...
1      during the activation of a sodium sulphide the...
2      in the milpo located at level when the collabo...
3      being approximately in the 1880 the personnel ...
4      approximately at in circumstances that the mec...
                           ...
420    being approximately when lifting the kelly hq ...
421    the collaborator moved from the infrastructure...
422    during the environmental monitoring activity i...
423    the employee performed the activity of strippi...
424    at when the assistant cleaned the floor of mod...
Name: Description, Length: 418, dtype: object
```

- Generating wordcloud

Wordcloud

```
all_texts = ' '.join(text for text in x)
```

```
#Generate wordcloud
all_texts = ' '.join(text for text in x)
print('Total cumber words after removing stopwords:', len(all_texts))
wrd_cld = WordCloud(max_font_size = 40, max_words=300, background_color="white")
cloud = wrd_cld.generate(all_texts)
plt.figure(figsize=(10,10))
plt.imshow(cloud)
plt.axis("off")
plt.show()
```

```
Total cumber words after removing stopwords: 131089
```

- Removing stop words from wordcloud

Removing Stopwords

```
def rmv_stpwrds(sen):
  wrds = sen.split()
  new_text = [w for w in wrds if w not in stopwords.words('english')]
  new_text = ' '.join(new_text)

  return new_text
```

```
x= x.apply(lambda x: rmv_stpwrds(x))
```

Wordcloud after Removing Stopwords

```
#Generate wordcloud
all_texts = ' '.join(text for text in x)
print('Total cumber words after removing stopwords:', len(all_texts))
wrd_cld = WordCloud(max_font_size = 40, max_words=300, background_color="white")
cloud = wrd_cld.generate(all_texts)
plt.figure(figsize=(10,10))
plt.imshow(cloud)
plt.axis("off")
plt.show()
```

```
Total cumber words after removing stopwords: 83076
```

Step 4: Data preparation - Cleansed data in .xlsx or .csv file

- Lemmatizing the words and removing extra spaces from the words

Lemmatizing the Words

```
[ ] lemmatizer = WordNetLemmatizer()

    def stem(sen):
      wrds = sen.split()
      new_text = [lemmatizer.lemmatize(w) for w in wrds]
      new_text = ' '.join(new_text)

      return new_text
```

```
[ ] x = x.apply(lambda x: stem(x))
```

Stripping the Extra Spaces

```
[ ] def strp(sen):
      sen = sen.strip()
      return sen
```

```
[ ] x = x.apply(lambda x: strp(x))
```

```
[ ] #Adding preprocessed description in the dataframe
    df['Cleaned_description'] = x
```

```
[ ] df.head()
```

	Date	Country	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee Type	Critical Risk	Description	Year	Day	Month	Day_name	Cleaned_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...	2016	1	1	Friday	removing drill rod jumbo 08 supervisor proceed...
1	2016-01-02	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...	2016	2	1	Saturday	activation sodium sulphide piping uncoupled su...
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	6	1	Wednesday	milpo located level collaborator excavation wo...
3	2016-01-08	Country_01	Local_04	Mining	I	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 C...	2016	8	1	Friday	approximately 1880 personnel begin task unlock...
4	2016-01-10	Country_01	Local_04	Mining	IV	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances t...	2016	10	1	Sunday	approximately circumstance mechanic anthony ed...

- Finally, we are going to output the cleansed data into .csv file for further use

```
[ ] #Saving preprocessed data into csv file
    df.to_csv('Preprocessed_Dataset_chatbot.csv')
```

- In the next step we are going to prepare data using TF-IDF Vectorizer and then split the data into train and test data

```
#Vectorization using tfidf
vec = TfidfVectorizer(max_features = 300)
x_vec = vec.fit_transform(x).toarray()
```

```
[ ] #Splitting data into train and test
    x_train, x_test, y_train, y_test = train_test_split(x_vec,y,train_size=0.8,random_state=1)
```

Step 5: Model Building using basic Machine learning classifiers

- Here we are going to Create function to trained the model

```
#Creating function to trained the model
def train_test(models, x_train,y_train,x_test,y_test):

  for name, model in models.items():
    print("\nModel Name:", name)
    model.fit(x_train,y_train)
    y_train_pred = model.predict(x_train)
    y_test_pred = model.predict(x_test)

    print('\nTraining Accuracy: ',model.score(x_train,y_train))
    print('Testing Accuracy: ',model.score(x_test,y_test))


    print('\nConfusion matrix for training set:\n')
    model_cm_train = confusion_matrix(y_train, y_train_pred)
    sns.heatmap(model_cm_train, annot=True,  fmt='.2f', xticklabels = ["I", "II", "III", "IV", "V"] , yticklabels = ["I", "II", "III", "IV", "V"] )
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.show()

    print('\nConfusion matrix for testing set:\n')
    model_cm_test = confusion_matrix(y_test, y_test_pred)
    sns.heatmap(model_cm_test, annot=True,  fmt='.2f', xticklabels = ["I", "II", "III", "IV", "V"] , yticklabels = ["I", "II", "III", "IV", "V"] )
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.show()
    print('**************************************************************************')
    print('\nClassification report for Training set:\n')
    print(classification_report(y_train,y_train_pred))

    print('\nClassification report for Testing set:\n')
    print(classification_report(y_test,y_test_pred))
    print('**************************************************************************')
```

- Next is to function for Hyperparameter tuning

```
#Function for Hyperparameter Tunning
def hyperparameter_tunning(model,x_train,y_train,params):
  cv = StratifiedKFold(n_splits=10)
  grid_search = GridSearchCV(estimator=model,param_grid=params)
  result = grid_search.fit(x_train, y_train)
  print("Best Score: " , result.best_score_)
  print("Best Parameter:",result.best_params_)
  means = result.cv_results_['mean_test_score']
  stds = result.cv_results_['std_test_score']
  params = result.cv_results_['params']
  for mean, stdev, param in zip(means, stds, params):
    if param == result.best_params_:
      print("95% Confidence interval range: ({0:.4f} %, {1:.4f} %)".format(mean-(2*stdev), mean+(2*stdev)))
  return result.best_params_
```

- Next step is to train a basic classifier like

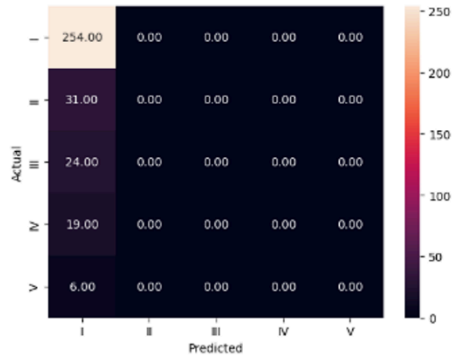
1. Naive Bayes without Sampling
2. KNearest Neighbour without Sampling
3. Logistic Regression without Sampling
4. SVM without Sampling
5. Decision Tree without Sampling
6. Random Forest without Sampling
7. Ada Boost without Sampling
8. Gradient Boost without Sampling

```
[ ] #Defining the models
    models = dict({'Naive Bayes without Sampling':MultinomialNB(),'KNearest Neighbor without Sampling':KNeighborsClassifier(),'Logistic Regression without Sampling':LogisticRegression(),
                   'SVM without Sampling':SVC(),'Decision Tree without Sampling':DecisionTreeClassifier(),'Random Forest without Sampling':RandomForestClassifier(),
                   'Ada Boost without Sampling':AdaBoostClassifier(),'Gradient Boost without Sampling':GradientBoostingClassifier()})
```

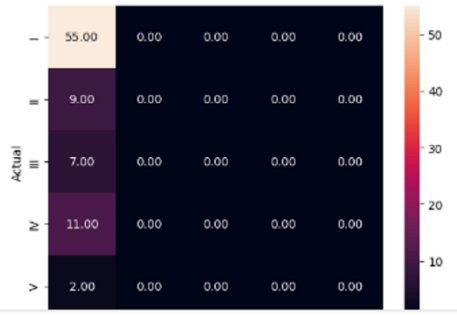
- Training all models

Naive Bayes without Sampling

```
Model Name: Naive Bayes without Sampling

Training Accuracy:  0.7604790419161677
Testing Accuracy:  0.6547619047619048

Confusion matrix for training set:
```



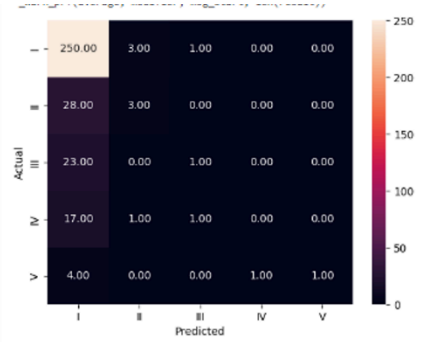

```
Confusion matrix for testing set:
```



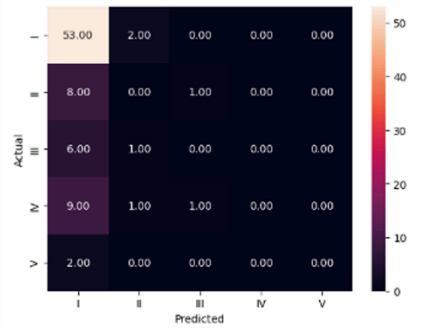

```
**********************************************************************

Classification report for Training set:

              precision    recall  f1-score   support

           I       0.76      1.00      0.86       254
          II       0.00      0.00      0.00        31
         III       0.00      0.00      0.00        24
          IV       0.00      0.00      0.00        19
           V       0.00      0.00      0.00         6

    accuracy                           0.76       334
   macro avg       0.15      0.20      0.17       334
weighted avg       0.58      0.76      0.66       334


Classification report for Testing set:

              precision    recall  f1-score   support

           I       0.65      1.00      0.79        55
          II       0.00      0.00      0.00         9
         III       0.00      0.00      0.00         7
          IV       0.00      0.00      0.00        11
           V       0.00      0.00      0.00         2

    accuracy                           0.65        84
   macro avg       0.13      0.20      0.16        84
weighted avg       0.43      0.65      0.52        84

**********************************************************************
```

KNearest Neighbour without Sampling

```
Model Name: KNearest Neighbor without Sampling

Training Accuracy:  0.7634730538922155
Testing Accuracy:  0.6309523809523809
```



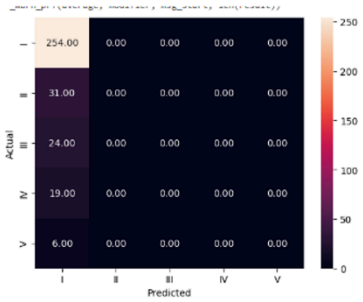
Confusion matrix for testing set:



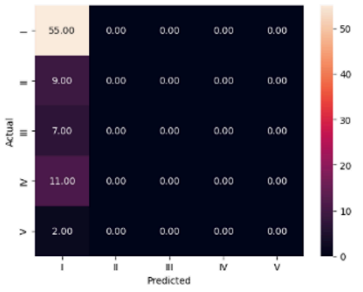

```
**********************************************************************

Classification report for Training set:

              precision    recall  f1-score   support

           I       0.78      0.98      0.87       254
          II       0.43      0.10      0.16        31
         III       0.33      0.04      0.07        24
          IV       0.00      0.00      0.00        19
           V       1.00      0.17      0.29         6

    accuracy                           0.76       334
   macro avg       0.51      0.26      0.28       334
weighted avg       0.67      0.76      0.69       334


Classification report for Testing set:

              precision    recall  f1-score   support

           I       0.68      0.96      0.80        55
          II       0.00      0.00      0.00         9
         III       0.00      0.00      0.00         7
          IV       0.00      0.00      0.00        11
           V       0.00      0.00      0.00         2

    accuracy                           0.63        84
   macro avg       0.14      0.19      0.16        84
weighted avg       0.44      0.63      0.52        84

**********************************************************************
```

Logistic Regression without Sampling

```
**********************************************************************

Model Name: Logistic Regression without Sampling

Training Accuracy:  0.7604790419161677
Testing Accuracy:  0.6547619047619048

Confusion matrix for training set:
```



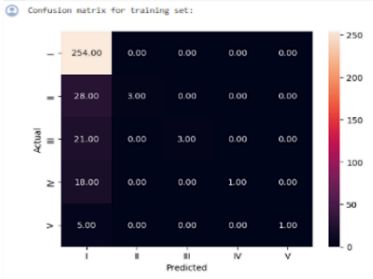
Confusion matrix for testing set:



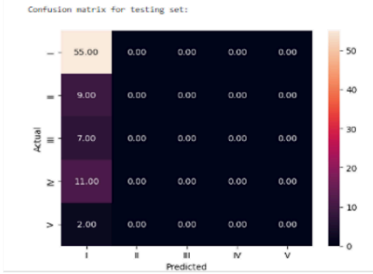

```
**********************************************************************

Classification report for Training set:

              precision    recall  f1-score   support

           I       0.76      1.00      0.86       254
          II       0.00      0.00      0.00        31
         III       0.00      0.00      0.00        24
          IV       0.00      0.00      0.00        19
           V       0.00      0.00      0.00         6

    accuracy                           0.76       334
   macro avg       0.15      0.20      0.17       334
weighted avg       0.58      0.76      0.66       334


Classification report for Testing set:

              precision    recall  f1-score   support

           I       0.65      1.00      0.79        55
          II       0.00      0.00      0.00         9
         III       0.00      0.00      0.00         7
          IV       0.00      0.00      0.00        11
           V       0.00      0.00      0.00         2

    accuracy                           0.65        84
   macro avg       0.13      0.20      0.16        84
weighted avg       0.43      0.65      0.52        84

**********************************************************************
```

SVM without Sampling

```
**********************************************************************

Model Name: SVM without Sampling

Training Accuracy:  0.7844311377245509
Testing Accuracy:  0.6547619047619048

Confusion matrix for training set:
```



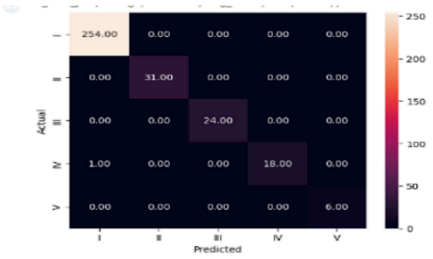

```
**********************************************************************

Classification report for Training set:

              precision    recall  f1-score   support

           I       0.78      1.00      0.88       254
          II       1.00      0.10      0.18        31
         III       1.00      0.12      0.22        24
          IV       1.00      0.05      0.10        19
           V       1.00      0.17      0.29         6

    accuracy                           0.78       334
   macro avg       0.96      0.29      0.33       334
weighted avg       0.83      0.78      0.71       334
```




```
Classification report for Testing set:

              precision    recall  f1-score   support

           I       0.65      1.00      0.79        55
          II       0.00      0.00      0.00         9
         III       0.00      0.00      0.00         7
          IV       0.00      0.00      0.00        11
           V       0.00      0.00      0.00         2

    accuracy                           0.65        84
   macro avg       0.13      0.20      0.16        84
weighted avg       0.43      0.65      0.52        84

**********************************************************************
```

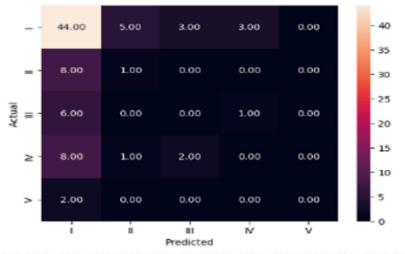
Decision Tree without Sampling

```
**********************************************************************

Model Name: Decision Tree without Sampling

Training Accuracy:  0.9970059880239521
Testing Accuracy:  0.5357142857142857

Confusion matrix for training set:
```




```
**********************************************************************

Classification report for Training set:

              precision    recall  f1-score   support

           I       1.00      1.00      1.00       254
          II       1.00      1.00      1.00        31
         III       1.00      1.00      1.00        24
          IV       1.00      0.95      0.97        19
           V       1.00      1.00      1.00         6

    accuracy                           1.00       334
   macro avg       1.00      0.99      0.99       334
weighted avg       1.00      1.00      1.00       334
```

```
Confusion matrix for testing set:
```



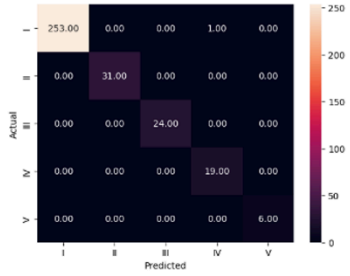

```
Classification report for Testing set:

              precision    recall  f1-score   support

           I       0.65      0.80      0.72        55
          II       0.14      0.11      0.12         9
         III       0.00      0.00      0.00         7
          IV       0.00      0.00      0.00        11
           V       0.00      0.00      0.00         2

    accuracy                           0.54        84
   macro avg       0.16      0.18      0.17        84
weighted avg       0.44      0.54      0.48        84

**********************************************************************
```

Random Forest without Sampling

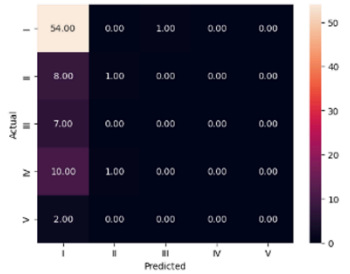
Model Name: Random Forest without Sampling

Training Accuracy: 0.9970059880239521
Testing Accuracy: 0.6547619047619048

Confusion matrix for training set:



Confusion matrix for testing set:



****

Classification report for Training set:

	precision	recall	f1-score	support
I	1.00	1.00	1.00	254
II	1.00	1.00	1.00	31
III	1.00	1.00	1.00	24
IV	0.95	1.00	0.97	19
V	1.00	1.00	1.00	6
accuracy			1.00	334
macro avg	0.99	1.00	0.99	334
weighted avg	1.00	1.00	1.00	334

Classification report for Testing set:

	precision	recall	f1-score	support
I	0.67	0.98	0.79	55
II	0.50	0.11	0.18	9
III	0.00	0.00	0.00	7
IV	0.00	0.00	0.00	11
V	0.00	0.00	0.00	2
accuracy			0.65	84
macro avg	0.23	0.22	0.20	84
weighted avg	0.49	0.65	0.54	84

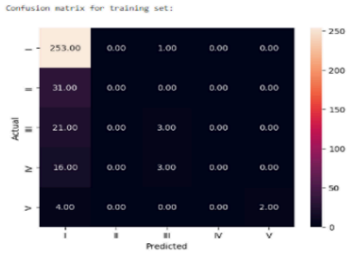
****

Ada Boost without Sampling

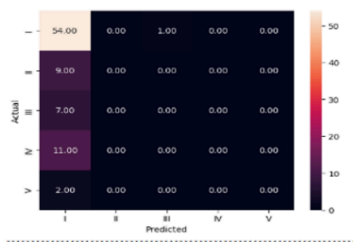
Model Name: Ada Boost without Sampling

Training Accuracy: 0.7724550898203593
Testing Accuracy: 0.6428571428571429

Confusion matrix for training set:



Confusion matrix for testing set:



****

Classification report for Training set:

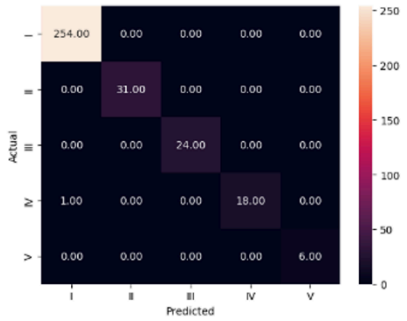
	precision	recall	f1-score	support
I	0.78	1.00	0.87	254
II	0.00	0.00	0.00	31
III	0.43	0.12	0.19	24
IV	0.00	0.00	0.00	19
V	1.00	0.33	0.50	6
accuracy			0.77	334
macro avg	0.44	0.29	0.31	334
weighted avg	0.64	0.77	0.69	334

Classification report for Testing set:

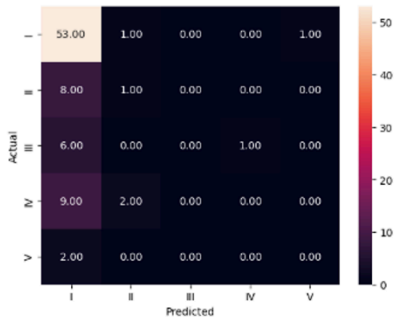
	precision	recall	f1-score	support
I	0.65	0.98	0.78	55
II	0.00	0.00	0.00	9
III	0.00	0.00	0.00	7
IV	0.00	0.00	0.00	11
V	0.00	0.00	0.00	2
accuracy			0.64	84
macro avg	0.13	0.20	0.16	84
weighted avg	0.43	0.64	0.51	84

Gradient Boost without Sampling

```
**********************************************************************

Model Name: Gradient Boost without Sampling

Training Accuracy:  0.9970059880239521
Testing Accuracy:  0.6428571428571429

Confusion matrix for training set:
```




```
Confusion matrix for testing set:
```




```
**********************************************************************

Classification report for Training set:

              precision    recall  f1-score   support

           I       1.00      1.00      1.00       254
          II       1.00      1.00      1.00        31
         III       1.00      1.00      1.00        24
          IV       1.00      0.95      0.97        19
           V       1.00      1.00      1.00         6

    accuracy                           1.00       334
   macro avg       1.00      0.99      0.99       334
weighted avg       1.00      1.00      1.00       334


Classification report for Testing set:

              precision    recall  f1-score   support

           I       0.68      0.96      0.80        55
          II       0.25      0.11      0.15         9
         III       0.00      0.00      0.00         7
          IV       0.00      0.00      0.00        11
           V       0.00      0.00      0.00         2

    accuracy                           0.64        84
   macro avg       0.19      0.21      0.19        84
weighted avg       0.47      0.64      0.54        84

**********************************************************************
```

- Next is to comparing all the models (without sampling)

```
# Function to get the metrics score
def get_metrics_score(model):

    score_list = []

    pred_train = model.predict(x_train)
    pred_test = model.predict(x_test)

    train_acc = round(model.score(x_train,y_train)*100,2)
    test_acc = round(model.score(x_test,y_test)*100,2)

    train_recall = round(recall_score(y_train,pred_train,average='weighted')*100,2)
    test_recall = round(recall_score(y_test,pred_test,average='weighted')*100,2)

    train_precision = round(precision_score(y_train,pred_train,average='weighted')*100,2)
    test_precision = round(precision_score(y_test,pred_test,average='weighted')*100,2)

    train_f1 = round(f1_score(y_train,pred_train,average='weighted')*100,2)
    test_f1 = round(f1_score(y_test,pred_test,average='weighted')*100,2)

    score_list.append([train_acc,test_acc,train_recall,test_recall,train_precision,test_precision,train_f1,test_f1])

    return score_list
```

```
[ ]
#Creating empty list to add train and test results
methods = []
acc_train = []
acc_test = []
recall_train = []
recall_test = []
precision_train = []
precision_test = []
f1_train = []
f1_test = []
#looping through the models
for name,model in models.items():
  methods.append(name)
  j = get_metrics_score(model)
  acc_train.append(j[0][0])
  acc_test.append(j[0][1])
  recall_train.append(j[0][2])
  recall_test.append(j[0][3])
  precision_train.append(j[0][4])
  precision_test.append(j[0][5])
  f1_train.append(j[0][6])
  f1_test.append(j[0][7])
```

```
#Creating Dataframe
comparison_frame = pd.DataFrame({'Methods':methods,
                                 'Train_Accuracy': acc_train,'Test_Accuracy': acc_test,
                                 'Train_Recall':recall_train,'Test_Recall':recall_test,
                                 'Train_Precision':precision_train,'Test_Precision':precision_test,
                                 'Train_F1':f1_train,
                                 'Test_F1':f1_test  })

#Sorting models in decreasing order of test accuracy
comparison_frame.sort_values(by='Test_F1',ascending=False)
```

	Methods	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision	Train_F1	Test_F1
5	Random Forest without Sampling	99.70	65.48	99.70	65.48	99.72	49.01	99.70	53.94
7	Gradient Boost without Sampling	99.70	64.29	99.70	64.29	99.70	47.17	99.70	53.83
1	KNearest Neighbor without Sampling	78.35	63.10	78.35	63.10	67.21	44.49	68.52	52.18
0	Naive Bayes without Sampling	78.05	65.48	78.05	65.48	57.83	42.87	65.70	51.82
2	Logistic Regression without Sampling	78.05	65.48	78.05	65.48	57.83	42.87	65.70	51.82
3	SVM without Sampling	78.44	65.48	78.44	65.48	83.20	42.87	70.92	51.82
6	Ada Boost without Sampling	77.25	64.29	77.25	64.29	64.08	42.60	68.75	51.24
4	Decision Tree without Sampling	99.70	53.57	99.70	53.57	99.70	43.90	99.70	48.18

Random Forest model have highest f1 score for test data, although all the models are overfitting.

Inference from the comparing all the models is Random Forest model have highest f1 score for test data and all the models are overfitting.

- After the above inference we will tune the models

Naive Bayes

```
[ ] params_nb = {'alpha':np.logspace(0,-9, num=10)}
    nb_param = hyperparameter_tunning(models['Naive Bayes without Sampling'],x_train,y_train,params_nb)
```

```
Best Score:  0.7604703753957486
Best Parameter: {'alpha': 1.0}
95% Confidence interval range: (0.7576 %, 0.7634 %)
```

KNearest Neighbor

```
[ ] params_knn = {'leaf_size':list(range(1,100)),'n_neighbors':list(range(3, 16, 2)),'weights' : ['uniform', 'distance'],'metric' : ['euclidean', 'manhattan']}
    knn_param = hyperparameter_tunning(models['KNearest Neighbor without Sampling'],x_train,y_train,params_knn)
```

```
Best Score:  0.7604703753957486
Best Parameter: {'leaf_size': 1, 'metric': 'euclidean', 'n_neighbors': 13, 'weights': 'uniform'}
95% Confidence interval range: (0.7576 %, 0.7634 %)
```

Logistic Regression

```
[ ] params_lr = {
        'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
        'penalty': ['l2'],
        'solver': ['newton-cg', 'lbfgs', 'sag', 'saga']
    }
    lr_param = hyperparameter_tunning(models['Logistic Regression without Sampling'],x_train,y_train,params_lr)
```

SVM

```
[ ] params_svm = {'C': [0.1, 1, 10, 100, 1000],
                  'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                  'kernel': ['rbf']}
    svm_param = hyperparameter_tunning(models['SVM without Sampling'],x_train,y_train,params_svm)
```

```
Best Score:  0.7604703753957486
Best Parameter: {'C': 0.1, 'gamma': 1, 'kernel': 'rbf'}
95% Confidence interval range: (0.7576 %, 0.7634 %)
```

Decision Tree

```
[ ] params_dt = {'max_depth': [2, 3, 5, 7, 10,15, 20],
        'min_samples_leaf': [5, 10, 20, 50, 100, 200, 300],
        'criterion': ["gini", "entropy"]}
    dt_param = hyperparameter_tunning(models['Decision Tree without Sampling'],x_train,y_train,params_dt)
```

```
Best Score:  0.7604703753957486
Best Parameter: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 20}
95% Confidence interval range: (0.7576 %, 0.7634 %)
```

Random Forest

```
[ ] params_rf = {'n_estimators': [150,200,250],
        'max_features':[0.2,0.3,0.4,0.5,0.6],
        'max_samples': [0.3,0.4,0.5,0.6]}
    rf_param = hyperparameter_tunning(models['Random Forest without Sampling'],x_train,y_train,params_rf)
```

```
Best Score:  0.7604703753957486
Best Parameter: {'max_features': 0.2, 'max_samples': 0.3, 'n_estimators': 150}
95% Confidence interval range: (0.7576 %, 0.7634 %)
```

Ada Boost

```
[ ] params_ada = {'n_estimators': [50, 100,150,200,250],'algorithm' : ['SAMME','SAMME.R'],
                  'learning_rate':[0.0001, 0.01, 0.1, 1.0, 1.1, 1.2]}
    ada_param = hyperparameter_tunning(models['Ada Boost without Sampling'],x_train,y_train,params_ada)
```

```
Best Score:  0.7604703753957486
Best Parameter: {'algorithm': 'SAMME', 'learning_rate': 0.01, 'n_estimators': 100}
95% Confidence interval range: (0.7576 %, 0.7634 %)
```

+ Code + Text

Gradient Boost

```
[ ] params_gb = {'n_estimators': [100, 150,200],'max_depth':np.arange(3,7,1),
                 'max_features':np.arange(0.2,0.6,0.1)
                }
    gb_param = hyperparameter_tunning(models['Gradient Boost without Sampling'],x_train,y_train,params_gb)
```

```
Best Score:  0.7126639529624603
Best Parameter: {'max_depth': 5, 'max_features': 0.2, 'n_estimators': 150}
95% Confidence interval range: (0.6307 %, 0.7947 %)
```

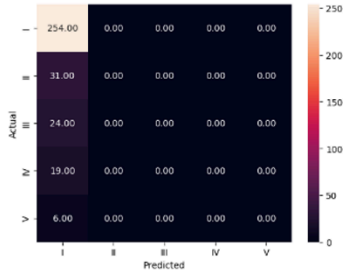
- Training all tuned models

Naive Bayes without Sampling-Tuned

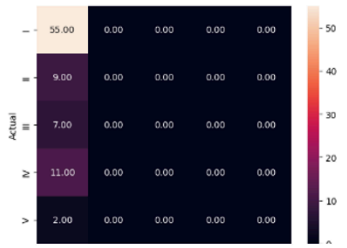
Model Name: Tunned Naive Bayes without Sampling

Training Accuracy: 0.7604790419161677
Testing Accuracy: 0.6547619047619048

Confusion matrix for training set:



Confusion matrix for testing set:



****

Classification report for Training set:

	precision	recall	f1-score	support
I	0.76	1.00	0.86	254
II	0.00	0.00	0.00	31
III	0.00	0.00	0.00	24
IV	0.00	0.00	0.00	19
V	0.00	0.00	0.00	6
accuracy			0.76	334
macro avg	0.15	0.20	0.17	334
weighted avg	0.58	0.76	0.66	334

Classification report for Testing set:

	precision	recall	f1-score	support
I	0.65	1.00	0.79	55
II	0.00	0.00	0.00	9
III	0.00	0.00	0.00	7
IV	0.00	0.00	0.00	11
V	0.00	0.00	0.00	2
accuracy			0.65	84
macro avg	0.13	0.20	0.16	84
weighted avg	0.43	0.65	0.52	84

****

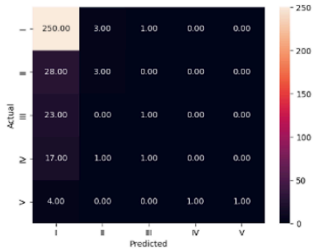
KNearest Neighbour without Sampling-Tuned

****

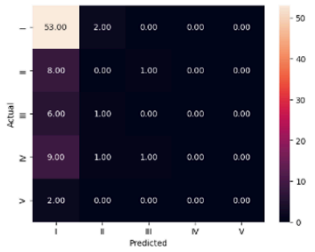
Model Name: Tunned KNearest Neighbor without Sampling

Training Accuracy: 0.7634730538922155
Testing Accuracy: 0.6309523809523809

Confusion matrix for training set:



Confusion matrix for testing set:



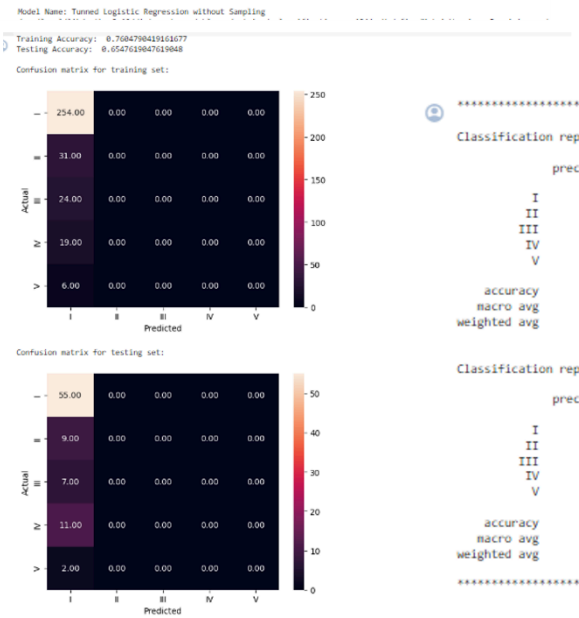
Classification report for Training set:

	precision	recall	f1-score	support
I	0.76	1.00	0.87	254
II	1.00	0.03	0.06	31
III	0.00	0.00	0.00	24
IV	0.00	0.00	0.00	19
V	0.00	0.00	0.00	6
accuracy			0.76	334
macro avg	0.35	0.21	0.19	334
weighted avg	0.67	0.76	0.66	334

Classification report for Testing set:

	precision	recall	f1-score	support
I	0.65	1.00	0.79	55
II	0.00	0.00	0.00	9
III	0.00	0.00	0.00	7
IV	0.00	0.00	0.00	11
V	0.00	0.00	0.00	2
accuracy			0.65	84
macro avg	0.13	0.20	0.16	84
weighted avg	0.43	0.65	0.52	84

Logistic Regression without Sampling-Tuned



****

Classification report for Training set:

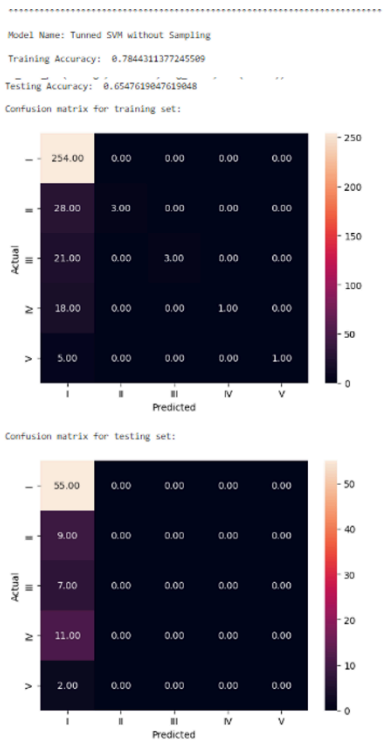
	precision	recall	f1-score	support
I	0.76	1.00	0.86	254
II	0.00	0.00	0.00	31
III	0.00	0.00	0.00	24
IV	0.00	0.00	0.00	19
V	0.00	0.00	0.00	6
accuracy			0.76	334
macro avg	0.15	0.20	0.17	334
weighted avg	0.58	0.76	0.66	334

Classification report for Testing set:

	precision	recall	f1-score	support
I	0.65	1.00	0.79	55
II	0.00	0.00	0.00	9
III	0.00	0.00	0.00	7
IV	0.00	0.00	0.00	11
V	0.00	0.00	0.00	2
accuracy			0.65	84
macro avg	0.13	0.20	0.16	84
weighted avg	0.43	0.65	0.52	84

****

SVM without Sampling-Tuned



Classification report for Training set:

	precision	recall	f1-score	support
I	0.76	1.00	0.86	254
II	0.00	0.00	0.00	31
III	0.00	0.00	0.00	24
IV	0.00	0.00	0.00	19
V	0.00	0.00	0.00	6
accuracy			0.76	334
macro avg	0.15	0.20	0.17	334
weighted avg	0.58	0.76	0.66	334

Classification report for Testing set:

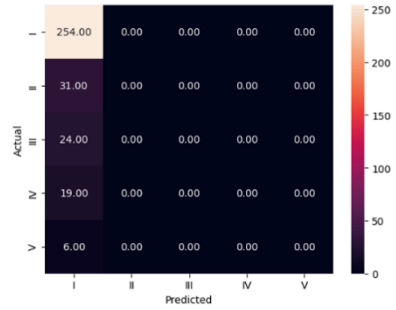
	precision	recall	f1-score	support
I	0.65	1.00	0.79	55
II	0.00	0.00	0.00	9
III	0.00	0.00	0.00	7
IV	0.00	0.00	0.00	11
V	0.00	0.00	0.00	2
accuracy			0.65	84
macro avg	0.13	0.20	0.16	84
weighted avg	0.43	0.65	0.52	84

Decision Tree without Sampling-Tuned

Model Name: Tunned Decision Tree without Sampling

Training Accuracy: 0.7604790419161677
Testing Accuracy: 0.6547619047619048

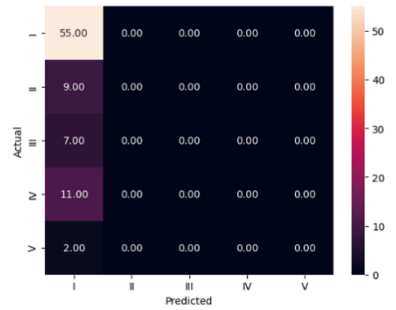
Confusion matrix for training set:



Classification report for Training set:

	precision	recall	f1-score	support
I	0.76	1.00	0.86	254
II	0.00	0.00	0.00	31
III	0.00	0.00	0.00	24
IV	0.00	0.00	0.00	19
V	0.00	0.00	0.00	6
accuracy			0.76	334
macro avg	0.15	0.20	0.17	334
weighted avg	0.58	0.76	0.66	334

Confusion matrix for testing set:



Classification report for Testing set:

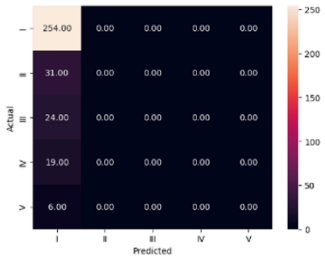
	precision	recall	f1-score	support
I	0.65	1.00	0.79	55
II	0.00	0.00	0.00	9
III	0.00	0.00	0.00	7
IV	0.00	0.00	0.00	11
V	0.00	0.00	0.00	2
accuracy			0.65	84
macro avg	0.13	0.20	0.16	84
weighted avg	0.43	0.65	0.52	84

Random Forest without Sampling-Tuned

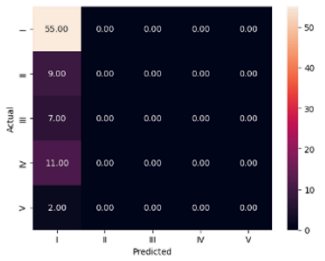
Model Name: Tunned Random Forest without Sampling

Training Accuracy: 0.7604790419161677
Testing Accuracy: 0.6547619047619048

Confusion matrix for training set:



Confusion matrix for testing set:



****

Classification report for Training set:

	precision	recall	f1-score	support
I	0.76	1.00	0.86	254
II	0.00	0.00	0.00	31
III	0.00	0.00	0.00	24
IV	0.00	0.00	0.00	19
V	0.00	0.00	0.00	6
accuracy			0.76	334
macro avg	0.15	0.20	0.17	334
weighted avg	0.58	0.76	0.66	334

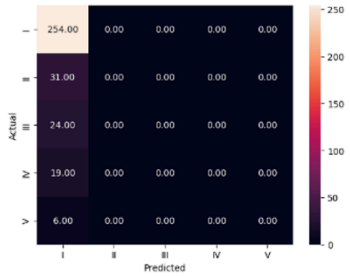
Classification report for Testing set:

	precision	recall	f1-score	support
I	0.65	1.00	0.79	55
II	0.00	0.00	0.00	9
III	0.00	0.00	0.00	7
IV	0.00	0.00	0.00	11
V	0.00	0.00	0.00	2
accuracy			0.65	84
macro avg	0.13	0.20	0.16	84
weighted avg	0.43	0.65	0.52	84

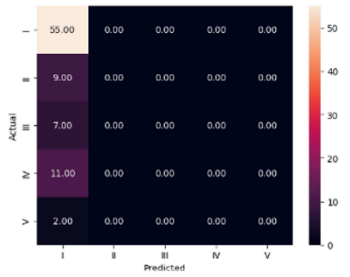
****

Ada Boost without Sampling-Tuned

```
Model Name: Tunned Ada Boost without Sampling

Training Accuracy:  0.7604790419161677
Testing Accuracy:  0.6547619047619048

Confusion matrix for training set:
```



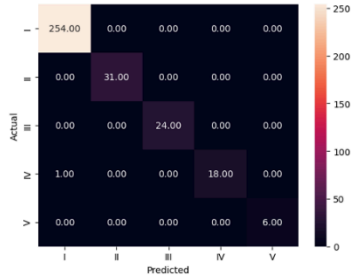

```
Confusion matrix for testing set:
```



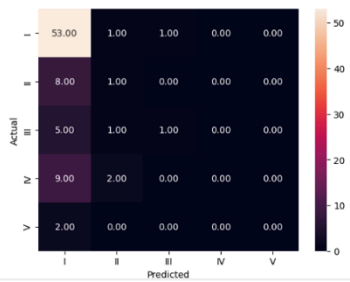

```
**********************************************************************

Classification report for Training set:

              precision    recall  f1-score   support

           I       0.76      1.00      0.86       254
          II       0.00      0.00      0.00        31
         III       0.00      0.00      0.00        24
          IV       0.00      0.00      0.00        19
           V       0.00      0.00      0.00         6

    accuracy                           0.76       334
   macro avg       0.15      0.20      0.17       334
weighted avg       0.58      0.76      0.66       334


Classification report for Testing set:

              precision    recall  f1-score   support

           I       0.65      1.00      0.79        55
          II       0.00      0.00      0.00         9
         III       0.00      0.00      0.00         7
          IV       0.00      0.00      0.00        11
           V       0.00      0.00      0.00         2

    accuracy                           0.65        84
   macro avg       0.13      0.20      0.16        84
weighted avg       0.43      0.65      0.52        84

**********************************************************************
```

Gradient Boost without Sampling-Tuned

```
**********************************************************************

Model Name: Tunned Gradient Boost without Sampling

Training Accuracy:  0.9970059880239521
Testing Accuracy:  0.6547619047619048

Confusion matrix for training set:
```




```
Confusion matrix for testing set:
```




```
**********************************************************************

Classification report for Training set:

              precision    recall  f1-score   support

           I       1.00      1.00      1.00       254
          II       1.00      1.00      1.00        31
         III       1.00      1.00      1.00        24
          IV       1.00      0.95      0.97        19
           V       1.00      1.00      1.00         6

    accuracy                           1.00       334
   macro avg       1.00      0.99      0.99       334
weighted avg       1.00      1.00      1.00       334


Classification report for Testing set:

              precision    recall  f1-score   support

           I       0.69      0.96      0.80        55
          II       0.20      0.11      0.14         9
         III       0.50      0.14      0.22         7
          IV       0.00      0.00      0.00        11
           V       0.00      0.00      0.00         2

    accuracy                           0.65        84
   macro avg       0.28      0.24      0.23        84
weighted avg       0.51      0.65      0.56        84
```

- Comparing All the Tuned Models

```
#Creating empty list to add train and test results
methods = []
acc_train = []
acc_test = []
recall_train = []
recall_test = []
precision_train = []
precision_test = []
f1_train = []
f1_test = []
#looping through the models
for name,model in models_tunned.items():
  methods.append(name)
  j = get_metrics_score(model)
  acc_train.append((j[0][0]))
  acc_test.append((j[0][1]))
  recall_train.append((j[0][2]))
  recall_test.append((j[0][3]))
  precision_train.append((j[0][4]))
  precision_test.append((j[0][5]))
  f1_train.append((j[0][6]))
  f1_test.append((j[0][7]))
```

- Creating a data frame

```
#Creating Dataframe
comparison_frame = pd.DataFrame({'Methods':methods,
                                 'Train_Accuracy': acc_train,'Test_Accuracy': acc_test,
                                 'Train_Recall':recall_train,'Test_Recall':recall_test,
                                 'Train_Precision':precision_train,'Test_Precision':precision_test,
                                 'Train_F1':f1_train,
                                 'Test_F1':f1_test  })

#Sorting models in decreasing order of test accuracy
comparison_frame.sort_values(by='Test_F1',ascending=False)
```

	Methods	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision	Train_F1	Test_F1
7	Tunned Gradient Boost without Sampling	99.70	65.48	99.70	65.48	99.70	51.38	99.70	55.96
0	Tunned Naive Bayes without Sampling	76.05	65.48	76.05	65.48	57.83	42.87	65.70	51.82
1	Tunned KNearest Neighbor without Sampling	76.35	65.48	76.35	65.48	67.29	42.87	66.39	51.82
2	Tunned Logistic Regression without Sampling	76.05	65.48	76.05	65.48	57.83	42.87	65.70	51.82
3	Tunned SVM without Sampling	76.05	65.48	76.05	65.48	57.83	42.87	65.70	51.82
4	Tunned Decision Tree without Sampling	76.05	65.48	76.05	65.48	57.83	42.87	65.70	51.82
5	Tunned Random Forest without Sampling	76.05	65.48	76.05	65.48	57.83	42.87	65.70	51.82
6	Tunned Ada Boost without Sampling	76.05	65.48	76.05	65.48	57.83	42.87	65.70	51.82

Summary of Tuning and comparing the Models

- The performance of the model is improved after tuning the model
- We need to try sampling the data as it is highly imbalanced

Model training after sampling

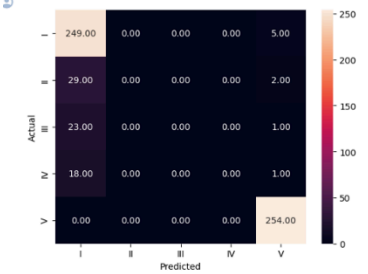
- Defining the models with sampling

```
#Defining the models
models_sampling = dict({'Naive Bayes with Sampling':MultinomialNB(),'KNearest Neighbor with Sampling':KNeighborsClassifier(),'Logistic Regression with Sampling':LogisticRegression(),
                'SVM with Sampling':SVC(),'Decision Tree with Sampling':DecisionTreeClassifier(),'Random Forest with Sampling':RandomForestClassifier(),
                'Ada Boost with Sampling':AdaBoostClassifier(),'Gradient Boost with Sampling':GradientBoostingClassifier()})
```

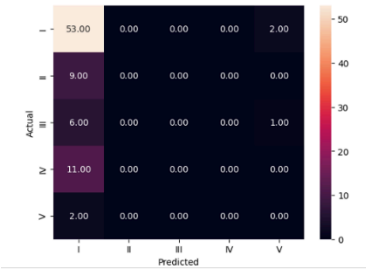
```
#Training all the moddels
train_test(models_sampling,x_train_sample,y_train_sample,x_test,y_test)
```

Naive Bayes with Sampling

```
Model Name: Naive Bayes with Sampling

Training Accuracy:  0.8642611683848798
Testing Accuracy:  0.6309523809523809

Confusion matrix for training set:
```



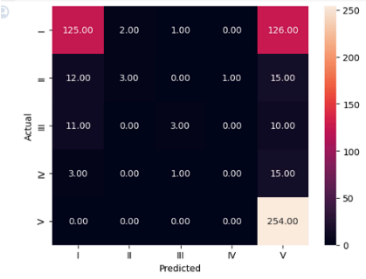

```
Confusion matrix for testing set:
```



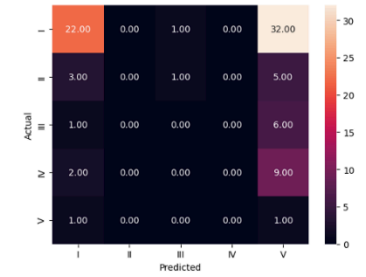

```
Predicted
**********************************************************************

Classification report for Training set:

              precision    recall  f1-score   support

           I       0.78      0.98      0.87       254
          II       0.00      0.00      0.00        31
         III       0.00      0.00      0.00        24
          IV       0.00      0.00      0.00        19
           V       0.97      1.00      0.98       254

    accuracy                           0.86       582
   macro avg       0.35      0.40      0.37       582
weighted avg       0.76      0.86      0.81       582


Classification report for Testing set:

              precision    recall  f1-score   support

           I       0.65      0.96      0.78        55
          II       0.00      0.00      0.00         9
         III       0.00      0.00      0.00         7
          IV       0.00      0.00      0.00        11
           V       0.00      0.00      0.00         2

    accuracy                           0.63        84
   macro avg       0.13      0.19      0.16        84
weighted avg       0.43      0.63      0.51        84

**********************************************************************
```

KNearest Neighbour with Sampling

```
Model Name: KNearest Neighbor with Sampling

Training Accuracy:  0.6615120274914089
Testing Accuracy:  0.27380952380952384

Confusion matrix for training set:
```



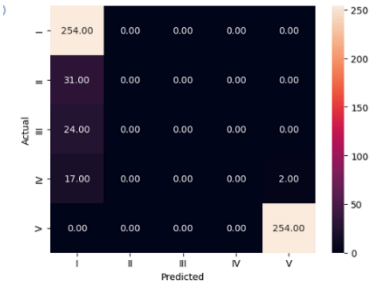

```
Confusion matrix for testing set:
```



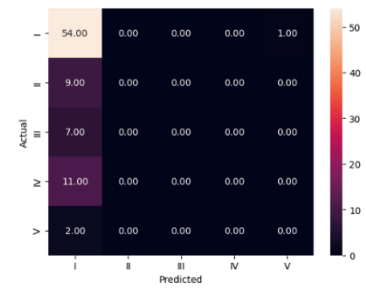

```
**********************************************************************

Classification report for Training set:

              precision    recall  f1-score   support

           I       0.83      0.49      0.62       254
          II       0.60      0.10      0.17        31
         III       0.60      0.12      0.21        24
          IV       0.00      0.00      0.00        19
           V       0.60      1.00      0.75       254

    accuracy                           0.66       582
   macro avg       0.53      0.34      0.35       582
weighted avg       0.68      0.66      0.62       582


Classification report for Testing set:

              precision    recall  f1-score   support

           I       0.76      0.40      0.52        55
          II       0.00      0.00      0.00         9
         III       0.00      0.00      0.00         7
          IV       0.00      0.00      0.00        11
           V       0.02      0.50      0.04         2

    accuracy                           0.27        84
   macro avg       0.16      0.18      0.11        84
weighted avg       0.50      0.27      0.34        84

**********************************************************************
```

Logistic Regression with Sampling

```
Model Name: Logistic Regression with Sampling

Training Accuracy:  0.872852233676976
Testing Accuracy:  0.6428571428571429

Confusion matrix for training set:
```




```
Confusion matrix for testing set:
```



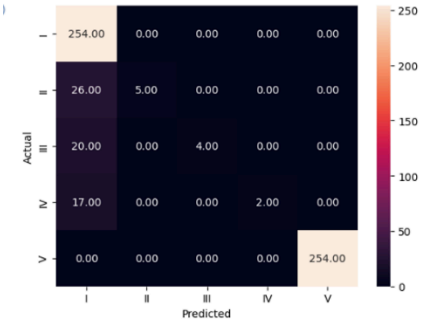

```
**********************************************************************

Classification report for Training set:

              precision    recall  f1-score   support

           I       0.78      1.00      0.88       254
          II       0.00      0.00      0.00        31
         III       0.00      0.00      0.00        24
          IV       0.00      0.00      0.00        19
           V       0.99      1.00      1.00       254

    accuracy                           0.87       582
   macro avg       0.35      0.40      0.37       582
weighted avg       0.77      0.87      0.82       582


Classification report for Testing set:

              precision    recall  f1-score   support

           I       0.65      0.98      0.78        55
          II       0.00      0.00      0.00         9
         III       0.00      0.00      0.00         7
          IV       0.00      0.00      0.00        11
           V       0.00      0.00      0.00         2

    accuracy                           0.64        84
   macro avg       0.13      0.20      0.16        84
weighted avg       0.43      0.64      0.51        84

**********************************************************************
```

SVM with Sampling

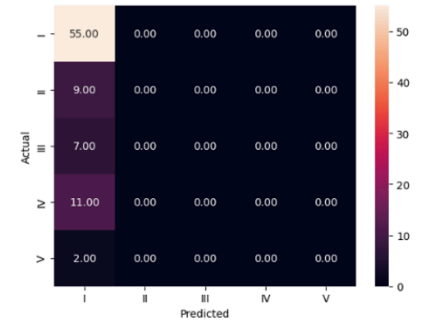
Model Name: SVM with Sampling

Training Accuracy: 0.8917525773195877
Testing Accuracy: 0.6547619047619048

Confusion matrix for training set:



Confusion matrix for testing set:



****

Classification report for Training set:

	precision	recall	f1-score	support
I	0.80	1.00	0.89	254
II	1.00	0.16	0.28	31
III	1.00	0.17	0.29	24
IV	1.00	0.11	0.19	19
V	1.00	1.00	1.00	254
accuracy			0.89	582
macro avg	0.96	0.49	0.53	582
weighted avg	0.91	0.89	0.86	582

Classification report for Testing set:

	precision	recall	f1-score	support
I	0.65	1.00	0.79	55
II	0.00	0.00	0.00	9
III	0.00	0.00	0.00	7
IV	0.00	0.00	0.00	11
V	0.00	0.00	0.00	2
accuracy			0.65	84
macro avg	0.13	0.20	0.16	84
weighted avg	0.43	0.65	0.52	84

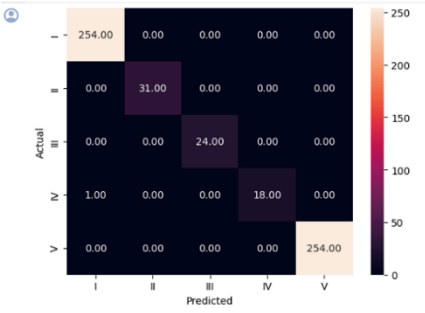
****

Decision Tree with Sampling

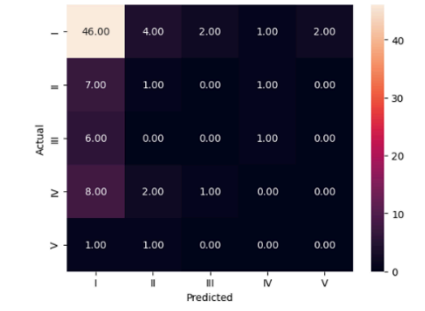
Model Name: Decision Tree with Sampling

Training Accuracy: 0.9982817869415808
Testing Accuracy: 0.5595238095238095

Confusion matrix for training set:



Confusion matrix for testing set:



****

Classification report for Training set:

	precision	recall	f1-score	support
I	1.00	1.00	1.00	254
II	1.00	1.00	1.00	31
III	1.00	1.00	1.00	24
IV	1.00	0.95	0.97	19
V	1.00	1.00	1.00	254
accuracy			1.00	582
macro avg	1.00	0.99	0.99	582
weighted avg	1.00	1.00	1.00	582

Classification report for Testing set:

	precision	recall	f1-score	support
I	0.68	0.84	0.75	55
II	0.12	0.11	0.12	9
III	0.00	0.00	0.00	7
IV	0.00	0.00	0.00	11
V	0.00	0.00	0.00	2
accuracy			0.56	84
macro avg	0.16	0.19	0.17	84
weighted avg	0.46	0.56	0.50	84

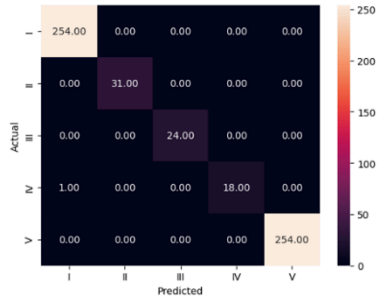
****

Random Forest with Sampling

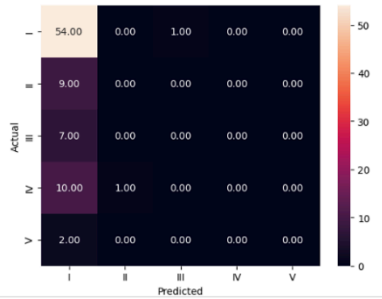
Model Name: Random Forest with Sampling

Training Accuracy: 0.9982817869415808
Testing Accuracy: 0.6428571428571429

Confusion matrix for training set:



Confusion matrix for testing set:



****

Classification report for Training set:

	precision	recall	f1-score	support
I	1.00	1.00	1.00	254
II	1.00	1.00	1.00	31
III	1.00	1.00	1.00	24
IV	1.00	0.95	0.97	19
V	1.00	1.00	1.00	254
accuracy			1.00	582
macro avg	1.00	0.99	0.99	582
weighted avg	1.00	1.00	1.00	582

Classification report for Testing set:

	precision	recall	f1-score	support
I	0.66	0.98	0.79	55
II	0.00	0.00	0.00	9
III	0.00	0.00	0.00	7
IV	0.00	0.00	0.00	11
V	0.00	0.00	0.00	2
accuracy			0.64	84
macro avg	0.13	0.20	0.16	84
weighted avg	0.43	0.64	0.52	84

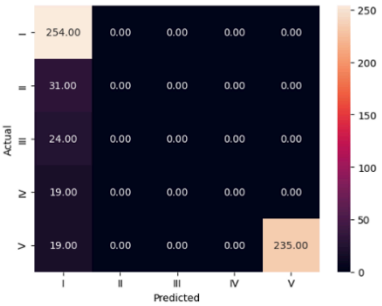
****

Ada Boost with Sampling

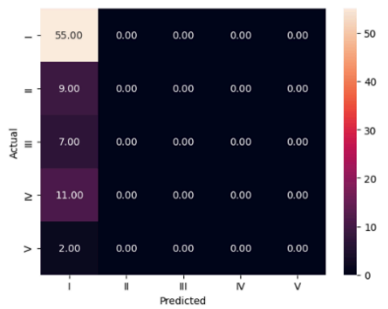
Model Name: Ada Boost with Sampling

Training Accuracy: 0.8402061855670103
Testing Accuracy: 0.6547619047619048

Confusion matrix for training set:



Confusion matrix for testing set:



Predicted

****

Classification report for Training set:

	precision	recall	f1-score	support
I	0.73	1.00	0.85	254
II	0.00	0.00	0.00	31
III	0.00	0.00	0.00	24
IV	0.00	0.00	0.00	19
V	1.00	0.93	0.96	254
accuracy			0.84	582
macro avg	0.35	0.39	0.36	582
weighted avg	0.76	0.84	0.79	582

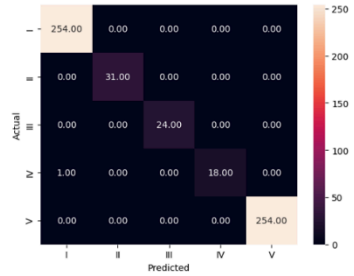
Classification report for Testing set:

	precision	recall	f1-score	support
I	0.65	1.00	0.79	55
II	0.00	0.00	0.00	9
III	0.00	0.00	0.00	7
IV	0.00	0.00	0.00	11
V	0.00	0.00	0.00	2
accuracy			0.65	84
macro avg	0.13	0.20	0.16	84
weighted avg	0.43	0.65	0.52	84

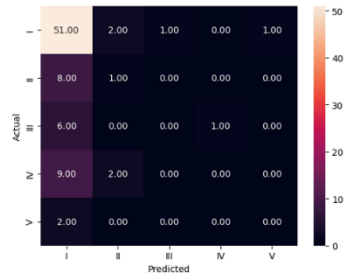
****

Gradient Boost with Sampling

```
**********************************************************************

Model Name: Gradient Boost with Sampling

Training Accuracy:  0.9982817869415808
Testing Accuracy:  0.6190476190476191

Confusion matrix for training set:
```



Confusion matrix for testing set:




```
**********************************************************************

Classification report for Training set:
              precision    recall  f1-score   support

           I       1.00      1.00      1.00       254
          II       1.00      1.00      1.00        31
         III       1.00      1.00      1.00        24
          IV       1.00      0.95      0.97        19
           V       1.00      1.00      1.00       254

    accuracy                           1.00       582
   macro avg       1.00      0.99      0.99       582
weighted avg       1.00      1.00      1.00       582


Classification report for Testing set:
              precision    recall  f1-score   support

           I       0.67      0.93      0.78        55
          II       0.20      0.11      0.14         9
         III       0.00      0.00      0.00         7
          IV       0.00      0.00      0.00        11
           V       0.00      0.00      0.00         2

    accuracy                           0.62        84
   macro avg       0.17      0.21      0.18        84
weighted avg       0.46      0.62      0.53        84

**********************************************************************
```

- Comparing All the Models with sampling

```
# Function to get the metrics score
def get_metrics_score_sample(model):

  score_list = []

  pred_train = model.predict(x_train_sample)
[ ]
  pred_test = model.predict(x_test)

  train_acc = round(model.score(x_train_sample,y_train_sample)*100,2)
  test_acc = round(model.score(x_test,y_test)*100,2)

  train_recall = round(recall_score(y_train_sample,pred_train,average='weighted')*100,2)
  test_recall = round(recall_score(y_test,pred_test,average='weighted')*100,2)

  train_precision = round(precision_score(y_train_sample,pred_train,average='weighted')*100,2)
  test_precision = round(precision_score(y_test,pred_test,average='weighted')*100,2)

  train_f1 = round(f1_score(y_train_sample,pred_train,average='weighted')*100,2)
  test_f1 = round(f1_score(y_test,pred_test,average='weighted')*100,2)

  score_list.append([train_acc,test_acc,train_recall,test_recall,train_precision,test_precision,train_f1,test_f1])

  return score_list
```

```
[ ] #Creating empty list to add train and test results
  methods = []
  acc_train = []
  acc_test = []
  recall_train = []
  recall_test = []
  precision_train = []
  precision_test = []
  f1_train = []
  f1_test = []
  #looping through the models
  for name,model in models_sampling.items():
    methods.append(name)
    j = get_metrics_score_sample(model)
    acc_train.append((j[0][0]))
    acc_test.append((j[0][1]))
    recall_train.append((j[0][2]))
    recall_test.append((j[0][3]))
    precision_train.append((j[0][4]))
    precision_test.append((j[0][5]))
    f1_train.append((j[0][6]))
    f1_test.append((j[0][7]))
```

- Creating a data frame

```
#Creating Dataframe
comparison_frame = pd.DataFrame({'Methods':methods,
                                 'Train_Accuracy': acc_train,'Test_Accuracy': acc_test,
                                 'Train_Recall':recall_train,'Test_Recall':recall_test,
                                 'Train_Precision':precision_train,'Test_Precision':precision_test,
                                 'Train_F1':f1_train,
                                 'Test_F1':f1_test  })

#Sorting models in decreasing order of test accuracy
comparison_frame.sort_values(by='Test_F1',ascending=False)
```

	Methods	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision	Train_F1	Test_F1
7	Gradient Boost with Sampling	99.83	61.90	99.83	61.90	99.83	46.08	99.83	52.51
3	SVM with Sampling	89.18	65.48	89.18	65.48	91.33	42.87	85.75	51.82
6	Ada Boost with Sampling	84.02	65.48	84.02	65.48	75.59	42.87	78.84	51.82
5	Random Forest with Sampling	99.83	64.29	99.83	64.29	99.83	43.12	99.83	51.62
2	Logistic Regression with Sampling	87.29	64.29	87.29	64.29	77.31	42.60	81.70	51.24
0	Naive Bayes with Sampling	86.43	63.10	86.43	63.10	76.22	42.84	80.81	51.03
4	Decision Tree with Sampling	99.83	55.95	99.83	55.95	99.83	45.63	99.83	50.23
1	KNearest Neighbor with Sampling	66.15	27.38	66.15	27.38	68.19	49.72	61.57	34.38

- Tuning the models with sampling

Naive Bayes

```
params_nb = {'alpha':np.logspace(0,-9, num=10)}
nb_param = hyperparameter_tunning(models_sampling['Naive Bayes with Sampling'],x_train_sample,y_train_sample,params_nb)
```

```
Best Score:  0.8608458591217213
Best Parameter: {'alpha': 0.1}
95% Confidence interval range: (0.8338 %, 0.8879 %)
```

KNearest Neighbor

```
params_knn = {'leaf_size':list(range(1,100)),'n_neighbors':list(range(3, 16, 2)),'weights' : ['uniform', 'distance'],'metric' : ['euclidean', 'manhattan']}
knn_param = hyperparameter_tunning(models_sampling['KNearest Neighbor with Sampling'],x_train_sample,y_train_sample,params_knn)
```

```
Best Score:  0.8625552608311228
Best Parameter: {'leaf_size': 1, 'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'uniform'}
95% Confidence interval range: (0.8441 %, 0.8810 %)
```

Logistic Regression

```
params_lr = {
        'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
        'penalty': ['l2'],
        'solver': ['newton-cg', 'lbfgs', 'sag', 'saga']
    }
lr_param = hyperparameter_tunning(models_sampling['Logistic Regression with Sampling'],x_train_sample,y_train_sample,params_lr)
```

SVM

```
[ ]  params_svm = {'C': [0.1, 1, 10, 100, 1000],
                   'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                   'kernel': ['rbf']}
     svm_param = hyperparameter_tunning(models_sampling['SVM with Sampling'],x_train_sample,y_train_sample,params_svm)
```

```
Best Score:  0.8728558797524315
Best Parameter: {'C': 0.1, 'gamma': 1, 'kernel': 'rbf'}
95% Confidence interval range: (0.8663 %, 0.8794 %)
```

Decision Tree

```
[ ]  params_dt = {'max_depth': [2, 3, 5, 7, 10,15, 20],
         'min_samples_leaf': [5, 10, 20, 50, 100, 200, 300],
         'criterion': ["gini", "entropy"]}
     dt_param = hyperparameter_tunning(models_sampling['Decision Tree with Sampling'],x_train_sample,y_train_sample,params_dt)
```

```
Best Score:  0.8642941349837902
Best Parameter: {'criterion': 'entropy', 'max_depth': 3, 'min_samples_leaf': 10}
95% Confidence interval range: (0.8376 %, 0.8910 %)
```

Random Forest

```
[ ]  params_rf = {'n_estimators': [150,200,250],
         'max_features':[0.2,0.3,0.4,0.5,0.6],
         'max_samples': [0.3,0.4,0.5,0.6]}
     rf_param = hyperparameter_tunning(models_sampling['Random Forest with Sampling'],x_train_sample,y_train_sample,params_rf)
```

```
Best Score:  0.8728558797524315
Best Parameter: {'max_features': 0.2, 'max_samples': 0.3, 'n_estimators': 150}
95% Confidence interval range: (0.8663 %, 0.8794 %)
```

Ada Boost

```
[ ]  params_ada = {'n_estimators': [50, 100,150,200,250],'algorithm' : ['SAMME','SAMME.R'],
                           'learning_rate':[0.0001, 0.01, 0.1, 1.0, 1.1, 1.2]}
     ada_param = hyperparameter_tunning(models_sampling['Ada Boost with Sampling'],x_train_sample,y_train_sample,params_ada)
```

```
Best Score:  0.8677276746242264
Best Parameter: {'algorithm': 'SAMME', 'learning_rate': 0.1, 'n_estimators': 100}
95% Confidence interval range: (0.8512 %, 0.8843 %)
```

Gradient Boost

```
[ ]  params_gb = {'n_estimators': [100, 150,200],'max_depth':np.arange(3,7,1),
                           'max_features':np.arange(0.2,0.6,0.1)
                          }
     gb_param = hyperparameter_tunning(models_sampling['Gradient Boost with Sampling'],x_train_sample,y_train_sample,params_gb)
```

```
Best Score:  0.8453728264073093
Best Parameter: {'max_depth': 4, 'max_features': 0.3000000000000004, 'n_estimators': 100}
95% Confidence interval range: (0.7899 %, 0.9008 %)
```

- Training models with sampling

```
[ ]  #Defining the models
     models_sampling_tunned = dict({'Tunned Naive Bayes with Sampling':MultinomialNB(**nb_param),'Tunned KNearest Neighbor with Sampling':KNeighborsClassifier(**knn_param),'Tunned Logistic Regression with Sampling':LogisticRegression(**lr_param),
                 'Tunned SVM with Sampling':SVC(**svm_param),'Tunned Decision Tree with Sampling':DecisionTreeClassifier(**dt_param),'Tunned Random Forest with Sampling':RandomForestClassifier(**rf_param),
                 'Tunned Ada Boost with Sampling':AdaBoostClassifier(**ada_param),'Tunned Gradient Boost with Sampling':GradientBoostingClassifier(**gb_param)})
```

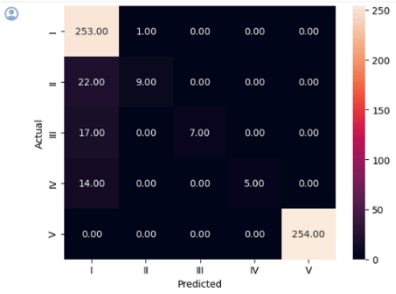
```
     #Training the model
     train_test(models_sampling_tunned,x_train_sample,y_train_sample,x_test,y_test)
```

Naive Bayes with Sampling-Tuned

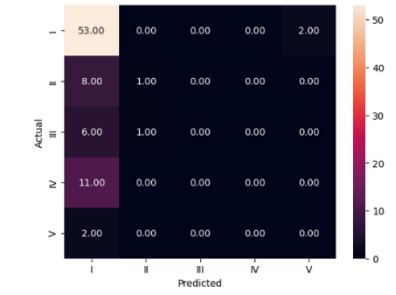
Model Name: Tunned Naive Bayes with Sampling

Training Accuracy: 0.9072164948453608
Testing Accuracy: 0.6428571428571429

Confusion matrix for training set:



Confusion matrix for testing set:



****

Classification report for Training set:

	precision	recall	f1-score	support
I	0.83	1.00	0.90	254
II	0.90	0.29	0.44	31
III	1.00	0.29	0.45	24
IV	1.00	0.26	0.42	19
V	1.00	1.00	1.00	254
accuracy			0.91	582
macro avg	0.95	0.57	0.64	582
weighted avg	0.92	0.91	0.89	582

Classification report for Testing set:

	precision	recall	f1-score	support
I	0.66	0.96	0.79	55
II	0.50	0.11	0.18	9
III	0.00	0.00	0.00	7
IV	0.00	0.00	0.00	11
V	0.00	0.00	0.00	2
accuracy			0.64	84
macro avg	0.23	0.21	0.19	84
weighted avg	0.49	0.64	0.53	84

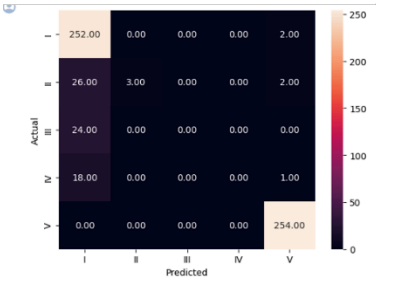
****

KNearest Neighbour with Sampling-Tuned

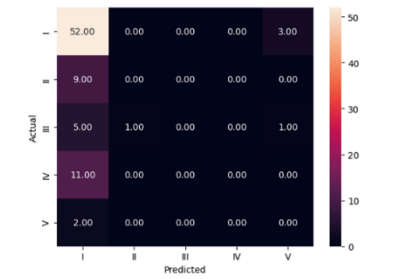
Model Name: Tunned KNearest Neighbor with Sampling

Training Accuracy: 0.8745704467353952
Testing Accuracy: 0.6190476190476191

Confusion matrix for training set:



Confusion matrix for testing set:



****

Classification report for Training set:

	precision	recall	f1-score	support
I	0.79	0.99	0.88	254
II	1.00	0.10	0.18	31
III	0.00	0.00	0.00	24
IV	0.00	0.00	0.00	19
V	0.98	1.00	0.99	254
accuracy			0.87	582
macro avg	0.55	0.42	0.41	582
weighted avg	0.82	0.87	0.82	582

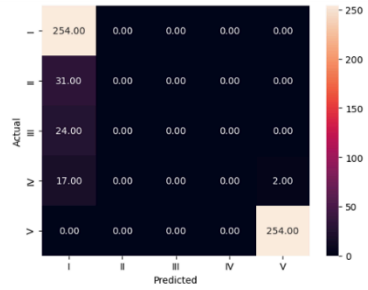
Classification report for Testing set:

	precision	recall	f1-score	support
I	0.66	0.95	0.78	55
II	0.00	0.00	0.00	9
III	0.00	0.00	0.00	7
IV	0.00	0.00	0.00	11
V	0.00	0.00	0.00	2
accuracy			0.62	84
macro avg	0.13	0.19	0.16	84
weighted avg	0.43	0.62	0.51	84

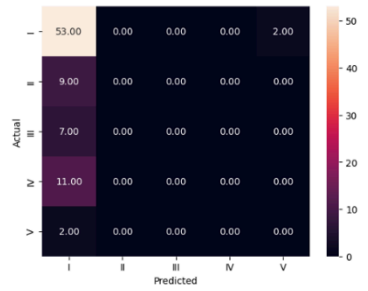
****

Logistic Regression with Sampling-Tuned

```
Model Name: Tunned Logistic Regression with Sampling

Training Accuracy:  0.872852233676976
Testing Accuracy:  0.6309523809523809

Confusion matrix for training set:
```



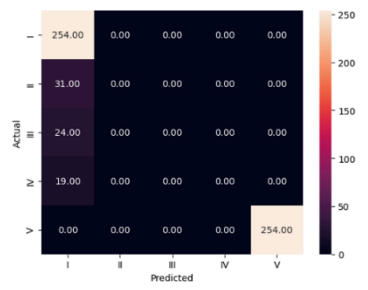

```
Confusion matrix for testing set:
```



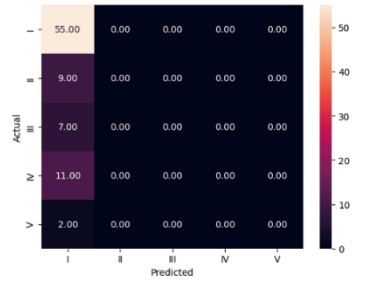

```
**********************************************************************

Classification report for Training set:

              precision    recall  f1-score   support

           I       0.78      1.00      0.88       254
          II       0.00      0.00      0.00        31
         III       0.00      0.00      0.00        24
          IV       0.00      0.00      0.00        19
           V       0.99      1.00      1.00       254

    accuracy                           0.87       582
   macro avg       0.35      0.40      0.37       582
weighted avg       0.77      0.87      0.82       582


Classification report for Testing set:

              precision    recall  f1-score   support

           I       0.65      0.96      0.77        55
          II       0.00      0.00      0.00         9
         III       0.00      0.00      0.00         7
          IV       0.00      0.00      0.00        11
           V       0.00      0.00      0.00         2

    accuracy                           0.63        84
   macro avg       0.13      0.19      0.15        84
weighted avg       0.42      0.63      0.51        84

**********************************************************************
```

SVM with Sampling-Tuned

```
Model Name: Tunned SVM with Sampling

Training Accuracy:  0.872852233676976
Testing Accuracy:  0.6547619047619048

Confusion matrix for training set:
```



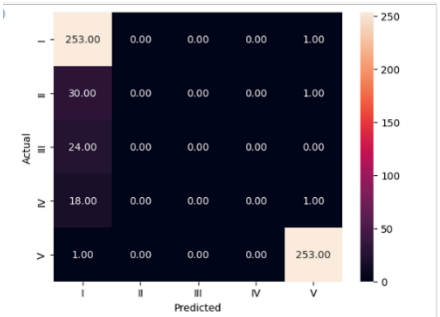

```
Confusion matrix for testing set:
```



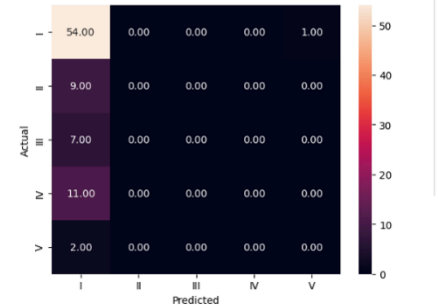

```
**********************************************************************

Classification report for Training set:

              precision    recall  f1-score   support

           I       0.77      1.00      0.87       254
          II       0.00      0.00      0.00        31
         III       0.00      0.00      0.00        24
          IV       0.00      0.00      0.00        19
           V       1.00      1.00      1.00       254

    accuracy                           0.87       582
   macro avg       0.35      0.40      0.37       582
weighted avg       0.77      0.87      0.82       582


Classification report for Testing set:

              precision    recall  f1-score   support

           I       0.65      1.00      0.79        55
          II       0.00      0.00      0.00         9
         III       0.00      0.00      0.00         7
          IV       0.00      0.00      0.00        11
           V       0.00      0.00      0.00         2

    accuracy                           0.65        84
   macro avg       0.13      0.20      0.16        84
weighted avg       0.43      0.65      0.52        84

**********************************************************************
```

Decision Tree with Sampling-Tuned

```
Model Name: Tunned Decision Tree with Sampling

Training Accuracy:  0.8694158075601375
Testing Accuracy:  0.6428571428571429

Confusion matrix for training set:
```



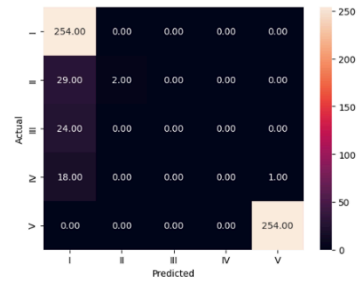

```
Confusion matrix for testing set:
```



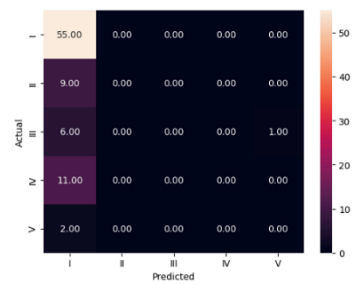

```
**********************************************************************

Classification report for Training set:

              precision    recall  f1-score   support

           I       0.78      1.00      0.87       254
          II       0.00      0.00      0.00        31
         III       0.00      0.00      0.00        24
          IV       0.00      0.00      0.00        19
           V       0.99      1.00      0.99       254

    accuracy                           0.87       582
   macro avg       0.35      0.40      0.37       582
weighted avg       0.77      0.87      0.81       582


Classification report for Testing set:

              precision    recall  f1-score   support

           I       0.65      0.98      0.78        55
          II       0.00      0.00      0.00         9
         III       0.00      0.00      0.00         7
          IV       0.00      0.00      0.00        11
           V       0.00      0.00      0.00         2

    accuracy                           0.64        84
   macro avg       0.13      0.20      0.16        84
weighted avg       0.43      0.64      0.51        84

**********************************************************************
```

Random Forest with Sampling-Tuned

```
**********************************************************************

Model Name: Tunned Random Forest with Sampling

Training Accuracy:  0.8762886597938144
Testing Accuracy:  0.6547619047619048

Confusion matrix for training set:
```




```
Confusion matrix for testing set:
```




```
**********************************************************************

Classification report for Training set:

              precision    recall  f1-score   support

           I       0.78      1.00      0.88       254
          II       1.00      0.06      0.12        31
         III       0.00      0.00      0.00        24
          IV       0.00      0.00      0.00        19
           V       1.00      1.00      1.00       254

    accuracy                           0.88       582
   macro avg       0.56      0.41      0.40       582
weighted avg       0.83      0.88      0.82       582


Classification report for Testing set:

              precision    recall  f1-score   support

           I       0.66      1.00      0.80        55
          II       0.00      0.00      0.00         9
         III       0.00      0.00      0.00         7
          IV       0.00      0.00      0.00        11
           V       0.00      0.00      0.00         2

    accuracy                           0.65        84
   macro avg       0.13      0.20      0.16        84
weighted avg       0.43      0.65      0.52        84

**********************************************************************
```

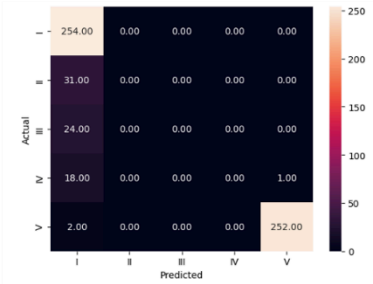
Ada Boost with Sampling-Tuned

****

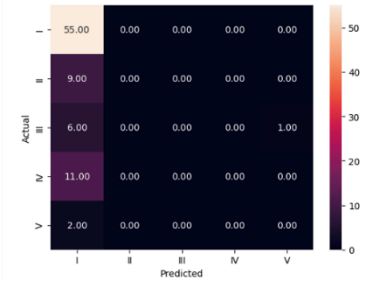
Model Name: Tunned Ada Boost with Sampling

Training Accuracy: 0.8694158075601375
Testing Accuracy: 0.6547619047619048

Confusion matrix for training set:



Confusion matrix for testing set:



****

Classification report for Training set:

	precision	recall	f1-score	support
I	0.77	1.00	0.87	254
II	0.00	0.00	0.00	31
III	0.00	0.00	0.00	24
IV	0.00	0.00	0.00	19
V	1.00	0.99	0.99	254
accuracy			0.87	582
macro avg	0.35	0.40	0.37	582
weighted avg	0.77	0.87	0.81	582

Classification report for Testing set:

	precision	recall	f1-score	support
I	0.66	1.00	0.80	55
II	0.00	0.00	0.00	9
III	0.00	0.00	0.00	7
IV	0.00	0.00	0.00	11
V	0.00	0.00	0.00	2
accuracy			0.65	84
macro avg	0.13	0.20	0.16	84
weighted avg	0.43	0.65	0.52	84

****

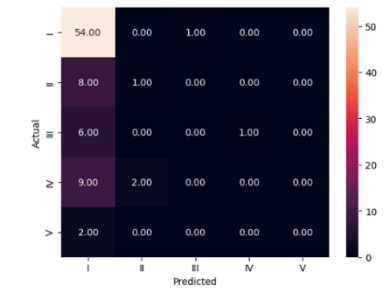
Gradient Boost with Sampling-Tuned

Model Name: Tunned Gradient Boost with Sampling

Training Accuracy: 0.9982817869415808
Testing Accuracy: 0.6547619047619048

Confusion matrix for training set:

Confusion matrix for testing set:



****

Classification report for Training set:

	precision	recall	f1-score	support
I	1.00	1.00	1.00	254
II	1.00	1.00	1.00	31
III	1.00	1.00	1.00	24
IV	1.00	0.95	0.97	19
V	1.00	1.00	1.00	254
accuracy			1.00	582
macro avg	1.00	0.99	0.99	582
weighted avg	1.00	1.00	1.00	582

Classification report for Testing set:

	precision	recall	f1-score	support
I	0.68	0.98	0.81	55
II	0.33	0.11	0.17	9
III	0.00	0.00	0.00	7
IV	0.00	0.00	0.00	11
V	0.00	0.00	0.00	2
accuracy			0.65	84
macro avg	0.20	0.22	0.19	84
weighted avg	0.48	0.65	0.55	84

****

- Comparing models with sampling

```
#Creating empty list to add train and test results
methods = []
acc_train = []
acc_test = []
recall_train = []
recall_test = []
precision_train = []
precision_test = []
f1_train = []
f1_test = []
#looping through the models
for name,model in models_sampling_tunned.items():
  methods.append(name)
  j = get_metrics_score_sample(model)
  acc_train.append((j[0][0]))
  acc_test.append((j[0][1]))
  recall_train.append((j[0][2]))
  recall_test.append((j[0][3]))
  precision_train.append((j[0][4]))
  precision_test.append((j[0][5]))
  f1_train.append((j[0][6]))
  f1_test.append((j[0][7]))
```

- Creating a data frame

```
[ ] #Creating Dataframe
    comparison_frame = pd.DataFrame({'Methods':methods,
                                     'Train_Accuracy': acc_train,'Test_Accuracy': acc_test,
                                     'Train_Recall':recall_train,'Test_Recall':recall_test,
                                     'Train_Precision':precision_train,'Test_Precision':precision_test,
                                     'Train_F1':f1_train,
                                     'Test_F1':f1_test  })

    #Sorting models in decreasing order of test accuracy
    comparison_frame.sort_values(by='Test_F1',ascending=False)
```

	Methods	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision	Train_F1	Test_F1
7	Tunned Gradient Boost with Sampling	99.83	65.48	99.83	65.48	99.83	48.33	99.83	54.56
0	Tunned Naive Bayes with Sampling	90.72	64.29	90.72	64.29	91.91	48.74	88.64	53.36
5	Tunned Random Forest with Sampling	87.63	65.48	87.63	65.48	82.91	43.39	82.49	52.19
6	Tunned Ada Boost with Sampling	86.94	65.48	86.94	65.48	77.16	43.39	81.41	52.19
3	Tunned SVM with Sampling	87.29	65.48	87.29	65.48	77.44	42.87	81.74	51.82
4	Tunned Decision Tree with Sampling	86.94	64.29	86.94	64.29	77.00	42.60	81.37	51.24
1	Tunned KNearest Neighbor with Sampling	87.46	61.90	87.46	61.90	82.50	43.10	82.48	50.82
2	Tunned Logistic Regression with Sampling	87.29	63.10	87.29	63.10	77.31	42.32	81.70	50.66

Conclusion of Milestone 1

- Gradient Boosting Model performs better which has a test f1 score of 55.92%. Also, after tuning the model f1 score improved to 55.96%.
- Since the distribution of target variable is highly unbalanced, so trained the models after oversampling of data, achieved f1 score of 52.51%, which further improved to 54.56% after tunning after tunning.
- After comparing all the models, we can conclude that gradient boosting algorithm without oversampling performs better which has f1 score of 55.96%.
- Further to improve the model performance and reduce overfitting, we can build neural networks using RNN or LSTM.

Milestone2

Step1: Design, train and test Neural networks classifiers

- Import necessary libraries to be use in this Project

```
from tensorflow.keras.regularizers import l1, l2, l1_l2
from keras.constraints import unit_norm
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences



import imblearn
from imblearn.over_sampling import SMOTE
```

- Vectorization using tfidf

```
[ ] #Vectorization using tfidf
    vec = TfidfVectorizer()
    x_vec = vec.fit_transform(x).toarray()
```

- One hot encoding of target variables

```
[ ] #One hot encoding of target variables
    y = pd.get_dummies(y).values
```

- Splitting data into Train and Test set

```
[ ] #Splitting data into train and test set
    x_train, x_test, y_train, y_test = train_test_split(x_vec,y,train_size=0.8,random_state=1)
```

- Resampling the minority class

```
[ ] # Resampling the minority class
    sm = SMOTE(sampling_strategy='minority', random_state=42)
    # Fit the model to generate the data.
    x_train_sample,y_train_sample = sm.fit_resample(x_train, y_train)
```

Building Neural Networks 1

- Building Neural Network

```
#building Neural nEtworkd
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(units=15,activation='relu',kernel_initializer='he_uniform'))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(units=20,activation='relu'))
model.add(tf.keras.layers.Dropout(0.4))
model.add(tf.keras.layers.Dense(units=12,activation='relu'))
model.add(tf.keras.layers.Dropout(0.4))
model.add(tf.keras.layers.Dense(units=10,activation='relu'))
model.add(tf.keras.layers.Dense(units=5,activation='softmax'))
```

- Compiling the model

```
#Compiling the model
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['Accuracy'])
#model.summary()
```

- Training the neural network

```
#Training the Neural Network
history = model.fit(x_train,y_train,batch_size=12,epochs=100,validation_data=(x_test,y_test))
```

```
Epoch 1/100
28/28 [==============================] - 2s 16ms/step - loss: 1.5456 - Accuracy: 0.7365 - val_loss: 1.4098 - val_Accuracy: 0.6548
Epoch 2/100
28/28 [==============================] - 0s 4ms/step - loss: 1.3526 - Accuracy: 0.7605 - val_loss: 1.3149 - val_Accuracy: 0.6548
Epoch 3/100
28/28 [==============================] - 0s 6ms/step - loss: 1.1187 - Accuracy: 0.7605 - val_loss: 1.1754 - val_Accuracy: 0.6548
Epoch 4/100
28/28 [==============================] - 0s 8ms/step - loss: 0.9671 - Accuracy: 0.7605 - val_loss: 1.1610 - val_Accuracy: 0.6548
Epoch 5/100
28/28 [==============================] - 0s 7ms/step - loss: 0.9216 - Accuracy: 0.7605 - val_loss: 1.1466 - val_Accuracy: 0.6548
Epoch 6/100
28/28 [==============================] - 0s 8ms/step - loss: 0.8436 - Accuracy: 0.7605 - val_loss: 1.1542 - val_Accuracy: 0.6548
Epoch 7/100
28/28 [==============================] - 0s 10ms/step - loss: 0.7938 - Accuracy: 0.7605 - val_loss: 1.1407 - val_Accuracy: 0.6548
Epoch 8/100
28/28 [==============================] - 0s 6ms/step - loss: 0.7745 - Accuracy: 0.7605 - val_loss: 1.1119 - val_Accuracy: 0.6548
Epoch 9/100
28/28 [==============================] - 0s 10ms/step - loss: 0.7055 - Accuracy: 0.7605 - val_loss: 1.1206 - val_Accuracy: 0.6548
Epoch 10/100
28/28 [==============================] - 0s 11ms/step - loss: 0.6633 - Accuracy: 0.7605 - val_loss: 1.1343 - val_Accuracy: 0.6548
Epoch 11/100
28/28 [==============================] - 0s 11ms/step - loss: 0.6395 - Accuracy: 0.7605 - val_loss: 1.1253 - val_Accuracy: 0.6548
Epoch 12/100
28/28 [==============================] - 0s 10ms/step - loss: 0.5718 - Accuracy: 0.7605 - val_loss: 1.1969 - val_Accuracy: 0.6548
Epoch 13/100
28/28 [==============================] - 0s 9ms/step - loss: 0.5603 - Accuracy: 0.7605 - val_loss: 1.1474 - val_Accuracy: 0.6548
Epoch 14/100
28/28 [==============================] - 0s 10ms/step - loss: 0.5099 - Accuracy: 0.7605 - val_loss: 1.2372 - val_Accuracy: 0.6548
Epoch 15/100
28/28 [==============================] - 0s 8ms/step - loss: 0.4833 - Accuracy: 0.7605 - val_loss: 1.2763 - val_Accuracy: 0.6548
Epoch 16/100
28/28 [==============================] - 0s 9ms/step - loss: 0.4684 - Accuracy: 0.7605 - val_loss: 1.4091 - val_Accuracy: 0.6548
Epoch 17/100
28/28 [==============================] - 0s 10ms/step - loss: 0.4539 - Accuracy: 0.7605 - val_loss: 1.5271 - val_Accuracy: 0.6548
Epoch 18/100
28/28 [==============================] - 0s 10ms/step - loss: 0.4432 - Accuracy: 0.7605 - val_loss: 1.5162 - val_Accuracy: 0.6548
Epoch 19/100
28/28 [==============================] - 0s 8ms/step - loss: 0.4311 - Accuracy: 0.7605 - val_loss: 1.6589 - val_Accuracy: 0.6548
Epoch 20/100
28/28 [==============================] - 0s 7ms/step - loss: 0.4236 - Accuracy: 0.7605 - val_loss: 1.7063 - val_Accuracy: 0.6548
Epoch 21/100
28/28 [==============================] - 0s 8ms/step - loss: 0.4161 - Accuracy: 0.7725 - val_loss: 1.8869 - val_Accuracy: 0.6429
Epoch 22/100
28/28 [==============================] - 0s 11ms/step - loss: 0.3996 - Accuracy: 0.8114 - val_loss: 1.9701 - val_Accuracy: 0.6429
Epoch 23/100
28/28 [==============================] - 0s 7ms/step - loss: 0.4066 - Accuracy: 0.8174 - val_loss: 2.1080 - val_Accuracy: 0.6429
Epoch 24/100
28/28 [==============================] - 0s 7ms/step - loss: 0.4079 - Accuracy: 0.8323 - val_loss: 2.1343 - val_Accuracy: 0.6429
Epoch 25/100
28/28 [==============================] - 0s 5ms/step - loss: 0.3885 - Accuracy: 0.8383 - val_loss: 2.2060 - val_Accuracy: 0.6429
Epoch 26/100
28/28 [==============================] - 0s 6ms/step - loss: 0.3917 - Accuracy: 0.8383 - val_loss: 2.2862 - val_Accuracy: 0.6429
```

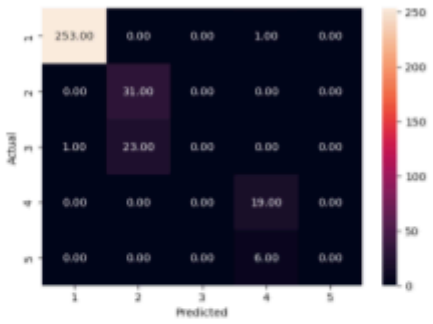
```
Epoch 86/100
28/28 [==============================] - 0s 4ms/step - loss: 0.2972 - Accuracy: 0.8772 - val_loss: 4.9151 - val_Accuracy: 0.6429
Epoch 87/100
28/28 [==============================] - 0s 4ms/step - loss: 0.2986 - Accuracy: 0.8772 - val_loss: 5.1287 - val_Accuracy: 0.6429
Epoch 88/100
28/28 [==============================] - 0s 4ms/step - loss: 0.2884 - Accuracy: 0.8743 - val_loss: 5.5341 - val_Accuracy: 0.6429
Epoch 27/100
28/28 [==============================] - 0s 5ms/step - loss: 0.3905 - Accuracy: 0.8383 - val_loss: 2.3324 - val_Accuracy: 0.6548
Epoch 28/100
28/28 [==============================] - 0s 5ms/step - loss: 0.3926 - Accuracy: 0.8413 - val_loss: 2.5077 - val_Accuracy: 0.6429
Epoch 29/100
28/28 [==============================] - 0s 5ms/step - loss: 0.3822 - Accuracy: 0.8473 - val_loss: 2.5197 - val_Accuracy: 0.6429
Epoch 30/100
28/28 [==============================] - 0s 6ms/step - loss: 0.3797 - Accuracy: 0.8443 - val_loss: 2.6467 - val_Accuracy: 0.6429
Epoch 31/100
28/28 [==============================] - 0s 7ms/step - loss: 0.3715 - Accuracy: 0.8473 - val_loss: 2.6609 - val_Accuracy: 0.6548
Epoch 32/100
28/28 [==============================] - 0s 6ms/step - loss: 0.3747 - Accuracy: 0.8473 - val_loss: 2.7793 - val_Accuracy: 0.6548
Epoch 33/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3727 - Accuracy: 0.8473 - val_loss: 2.8356 - val_Accuracy: 0.6548
Epoch 34/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3657 - Accuracy: 0.8473 - val_loss: 2.9843 - val_Accuracy: 0.6548
Epoch 35/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3686 - Accuracy: 0.8443 - val_loss: 3.1416 - val_Accuracy: 0.6429
Epoch 36/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3649 - Accuracy: 0.8443 - val_loss: 3.1313 - val_Accuracy: 0.6548
Epoch 37/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3726 - Accuracy: 0.8413 - val_loss: 3.2449 - val_Accuracy: 0.6548
Epoch 38/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3689 - Accuracy: 0.8443 - val_loss: 3.4246 - val_Accuracy: 0.6429
Epoch 39/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3636 - Accuracy: 0.8473 - val_loss: 3.6068 - val_Accuracy: 0.6548
Epoch 40/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3644 - Accuracy: 0.8473 - val_loss: 3.5881 - val_Accuracy: 0.6548
Epoch 41/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3647 - Accuracy: 0.8473 - val_loss: 3.4966 - val_Accuracy: 0.6548
Epoch 42/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3494 - Accuracy: 0.8503 - val_loss: 3.6650 - val_Accuracy: 0.6548
Epoch 43/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3565 - Accuracy: 0.8443 - val_loss: 3.7815 - val_Accuracy: 0.6548
Epoch 44/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3564 - Accuracy: 0.8443 - val_loss: 3.8042 - val_Accuracy: 0.6548
Epoch 45/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3658 - Accuracy: 0.8413 - val_loss: 3.7865 - val_Accuracy: 0.6429
Epoch 46/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3484 - Accuracy: 0.8473 - val_loss: 3.7940 - val_Accuracy: 0.6429
Epoch 47/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3528 - Accuracy: 0.8443 - val_loss: 3.7855 - val_Accuracy: 0.6667
Epoch 48/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3546 - Accuracy: 0.8443 - val_loss: 3.8991 - val_Accuracy: 0.6429
Epoch 49/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3506 - Accuracy: 0.8443 - val_loss: 4.1091 - val_Accuracy: 0.6429
Epoch 50/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3436 - Accuracy: 0.8443 - val_loss: 4.0325 - val_Accuracy: 0.6429
Epoch 51/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3562 - Accuracy: 0.8383 - val_loss: 4.1683 - val_Accuracy: 0.6429
Epoch 52/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3410 - Accuracy: 0.8473 - val_loss: 4.2924 - val_Accuracy: 0.6429
Epoch 52/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3410 - Accuracy: 0.8473 - val_loss: 4.2924 - val_Accuracy: 0.6429
Epoch 53/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3504 - Accuracy: 0.8443 - val_loss: 4.3504 - val_Accuracy: 0.6429
Epoch 54/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3609 - Accuracy: 0.8473 - val_loss: 3.9489 - val_Accuracy: 0.6667
Epoch 55/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3421 - Accuracy: 0.8473 - val_loss: 3.9655 - val_Accuracy: 0.6667
Epoch 56/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3443 - Accuracy: 0.8443 - val_loss: 3.8999 - val_Accuracy: 0.6667
Epoch 57/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3452 - Accuracy: 0.8443 - val_loss: 4.0050 - val_Accuracy: 0.6548
Epoch 58/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3456 - Accuracy: 0.8443 - val_loss: 4.1897 - val_Accuracy: 0.6548
Epoch 59/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3399 - Accuracy: 0.8473 - val_loss: 4.2453 - val_Accuracy: 0.6548
Epoch 60/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3398 - Accuracy: 0.8443 - val_loss: 4.5099 - val_Accuracy: 0.6548
Epoch 61/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3451 - Accuracy: 0.8473 - val_loss: 4.2826 - val_Accuracy: 0.6667
Epoch 62/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3448 - Accuracy: 0.8443 - val_loss: 4.4700 - val_Accuracy: 0.6548
Epoch 63/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3380 - Accuracy: 0.8503 - val_loss: 4.5465 - val_Accuracy: 0.6667
Epoch 64/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3395 - Accuracy: 0.8443 - val_loss: 4.6871 - val_Accuracy: 0.6548
Epoch 66/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3226 - Accuracy: 0.8653 - val_loss: 4.6991 - val_Accuracy: 0.6667
Epoch 67/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3319 - Accuracy: 0.8563 - val_loss: 4.9848 - val_Accuracy: 0.6548
Epoch 68/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3355 - Accuracy: 0.8593 - val_loss: 4.3689 - val_Accuracy: 0.6667
Epoch 69/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3265 - Accuracy: 0.8653 - val_loss: 4.4005 - val_Accuracy: 0.6667
Epoch 70/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3330 - Accuracy: 0.8563 - val_loss: 4.6671 - val_Accuracy: 0.6667
Epoch 71/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3216 - Accuracy: 0.8653 - val_loss: 4.7820 - val_Accuracy: 0.6667
Epoch 72/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3366 - Accuracy: 0.8623 - val_loss: 5.1011 - val_Accuracy: 0.6429
Epoch 73/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3202 - Accuracy: 0.8593 - val_loss: 4.7743 - val_Accuracy: 0.6667
Epoch 74/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3259 - Accuracy: 0.8593 - val_loss: 4.7021 - val_Accuracy: 0.6667
Epoch 75/100
28/28 [==============================] - 0s 3ms/step - loss: 0.3167 - Accuracy: 0.8713 - val_loss: 4.8719 - val_Accuracy: 0.6667
Epoch 76/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3230 - Accuracy: 0.8623 - val_loss: 5.0639 - val_Accuracy: 0.6548
Epoch 77/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3224 - Accuracy: 0.8653 - val_loss: 5.1049 - val_Accuracy: 0.6548
Epoch 78/100
28/28 [==============================] - 0s 4ms/step - loss: 0.3230 - Accuracy: 0.8623 - val_loss: 4.9810 - val_Accuracy: 0.6548
```

```
print('\nConfusion matrix for training set:\n')
model_cm_train = confusion_matrix(np.argmax(y_train,axis=1)+1, y_train_pred)
sns.heatmap(model_cm_train, annot=True,  fmt='.2f', xticklabels = ["1", "2", "3", "4","5"] , yticklabels = ["1", "2", "3", "4","5"] )
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

```
print('\nConfusion matrix for testing set:\n')
model_cm_test = confusion_matrix(np.argmax(y_test,axis=1)+1, y_test_pred)
sns.heatmap(model_cm_test, annot=True,  fmt='.2f', xticklabels = ["1", "2", "3", "4","5"] , yticklabels = ["1", "2", "3", "4","5"]  )
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
print('**********************************************************************')
print('\nClassification report for Training set:\n')
print(classification_report(np.argmax(y_train,axis=1)+1,y_train_pred))

print('\nClassification report for Testing set:\n')
print(classification_report(np.argmax(y_test,axis=1)+1,y_test_pred))
print('**********************************************************************')
```

```
evaluation_matrix(model,x_train,y_train,x_test,y_test)
```

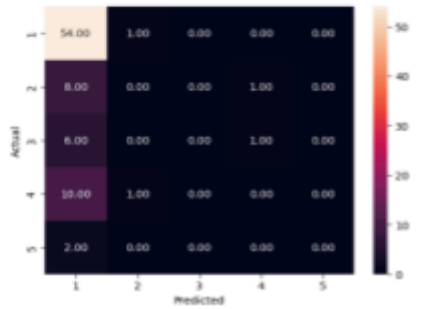
```
11/11 [==============================] - 0s 3ms/step
3/3 [==============================] - 0s 5ms/step
Train Recall: 90.72
Test Recall: 64.29
Train Precision: 85.23
Test Precision: 66.2
Train F1: 87.32
Test F1: 52.38

Confusion matrix for training set:
```



Classification report for Training set:

	precision	recall	f1-score	support
1	1.00	1.00	1.00	254
2	0.57	1.00	0.73	31
3	0.00	0.00	0.00	24
4	0.73	1.00	0.84	19
5	0.00	0.00	0.00	6
accuracy			0.91	334
macro avg	0.46	0.60	0.51	334
weighted avg	0.85	0.91	0.87	334

```
Confusion matrix for testing set:
```



Classification report for Testing set:

	precision	recall	f1-score	support
1	0.68	0.98	0.80	55
2	0.00	0.00	0.00	9
3	0.00	0.00	0.00	7
4	0.00	0.00	0.00	11
5	0.00	0.00	0.00	2
accuracy			0.64	84
macro avg	0.14	0.20	0.16	84
weighted avg	0.44	0.64	0.52	84

```
**********
```

From the Classification report we can understand that

- The model is overfitting
- Test f1-score of the model goes down

Hence, we would build another Neural network

Building Neural Networks 2

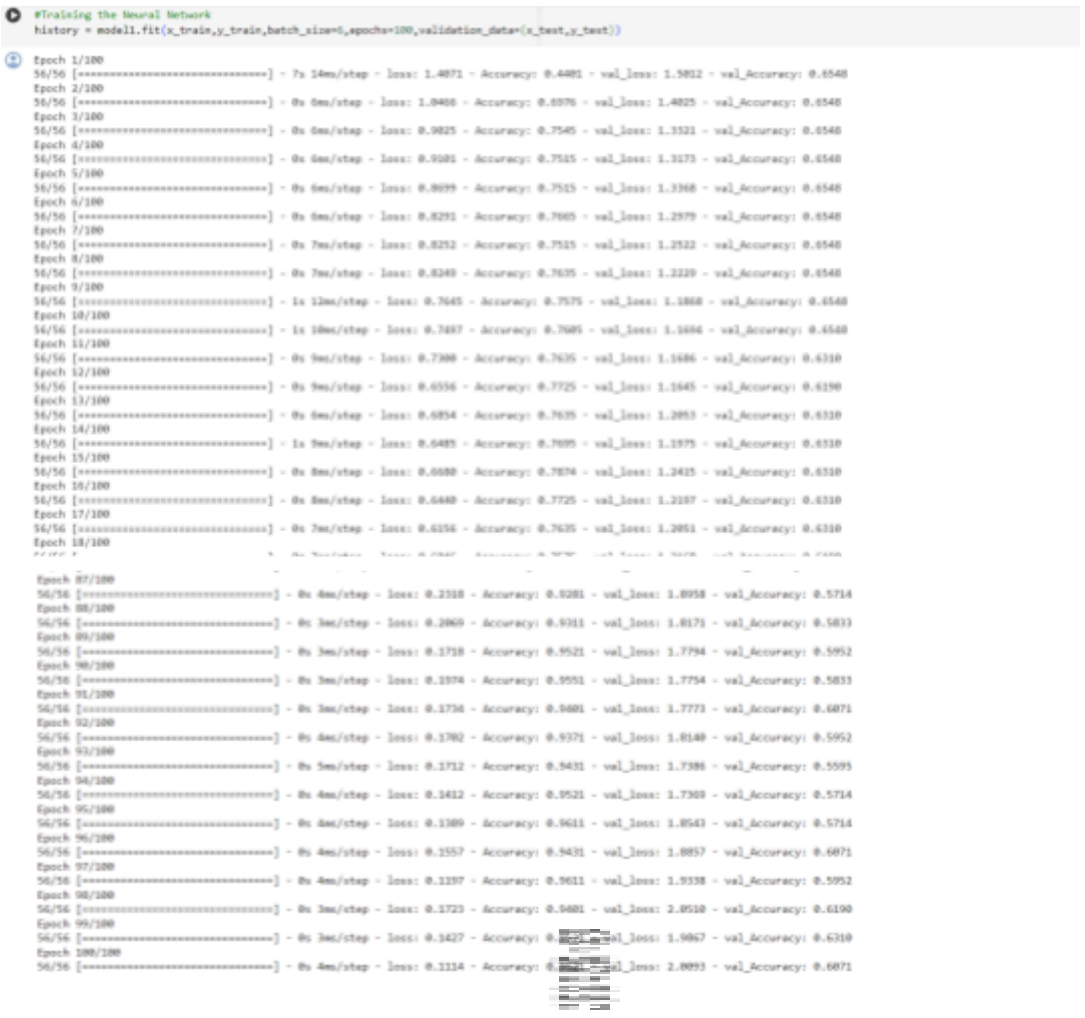
- Building Neural Network2

```
[ ]  #building Neural nEtworks 2
     model1 = tf.keras.models.Sequential()
     model1.add(tf.keras.layers.Dense(units=25,activation='relu',kernel_initializer='he_uniform'))
     model1.add(tf.keras.layers.BatchNormalization())
     model1.add(tf.keras.layers.Dropout(0.2))
     model1.add(tf.keras.layers.Dense(units=30,activation='relu'))
     model1.add(tf.keras.layers.BatchNormalization())
     model1.add(tf.keras.layers.Dropout(0.2))
     model1.add(tf.keras.layers.Dense(units=30,activation='relu'))
     model1.add(tf.keras.layers.BatchNormalization())
     model1.add(tf.keras.layers.Dropout(0.2))
     model1.add(tf.keras.layers.Dense(units=15,activation='relu'))
     model1.add(tf.keras.layers.Dense(units=5,activation='softmax'))
```

- Compiling the model

```
[ ]  #Compiling the model
     model1.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['Accuracy'])
```

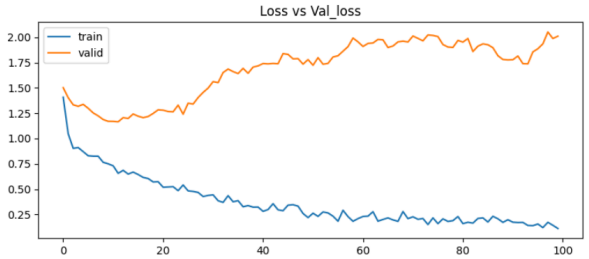
- Training the Neural Network

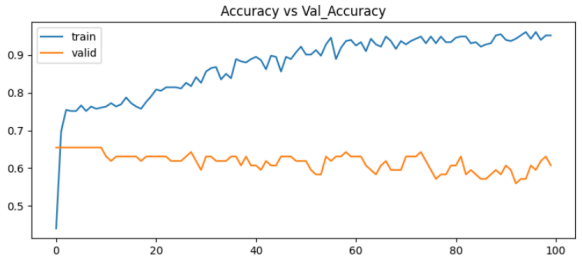


- Plotting Loss and accuracy curve vs epochs

```
#Plotting loss and accuracy curve vs epochs
plt.figure(figsize=(20,8))
plt.subplot(2,2,1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(("train" , "valid") , loc =0)
plt.title("Loss vs Val_loss")
plt.subplot(2,2,2)
plt.plot(history.history['Accuracy'])
plt.plot(history.history['val_Accuracy'])
plt.legend(("train" , "valid") , loc =0)
plt.title("Accuracy vs Val_Accuracy")
```

```
Text(0.5, 1.0, 'Accuracy vs Val_Accuracy')
```

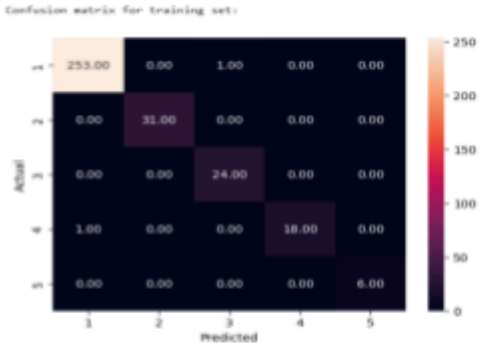


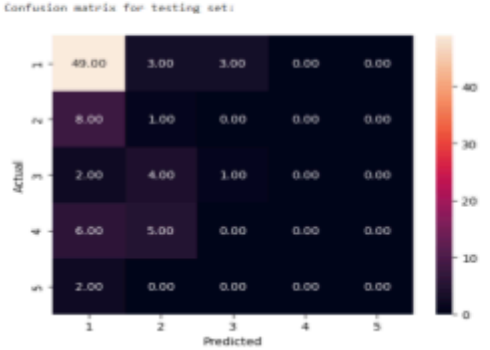


- Creating function to display Evaluation matrix

```
evaluation_matrix(model1,x_train,y_train,x_test,y_test)
```

```
11/11 [==============================] - 0s 2ms/step
3/3 [==============================] - 0s 4ms/step
Train Recall: 99.4
Test Recall: 60.71
Train Precision: 99.41
Test Precision: 50.79
Train F1: 99.4
Test F1: 55.08
```






```
**********************************************************************

Classification report for Training set:

              precision    recall  f1-score   support

           1       1.00      1.00      1.00       254
           2       1.00      1.00      1.00        31
           3       0.96      1.00      0.98        24
           4       1.00      0.95      0.97        19
           5       1.00      1.00      1.00         6

    accuracy                           0.99       334
   macro avg       0.99      0.99      0.99       334
weighted avg       0.99      0.99      0.99       334


Classification report for Testing set:

              precision    recall  f1-score   support

           1       0.73      0.89      0.80        55
           2       0.08      0.11      0.09         9
           3       0.25      0.14      0.18         7
           4       0.00      0.00      0.00        11
           5       0.00      0.00      0.00         2

    accuracy                           0.61        84
   macro avg       0.21      0.23      0.22        84
weighted avg       0.51      0.61      0.55        84

**********************************************************************
```

Seems still we can improve the model scores so we Build Neural Network3

Building Neural Networks 3

- Building Neural Network3

```
[ ]  #building Neural nEtworks 3
     param=1e-4
     model2 = tf.keras.models.Sequential()
     model2.add(tf.keras.layers.Dense(units=15,activation='relu',kernel_initializer='he_uniform',kernel_regularizer=l2(param),
                    kernel_constraint=unit_norm()))
     model2.add(tf.keras.layers.BatchNormalization())
     model2.add(tf.keras.layers.Dropout(0.5))
     model2.add(tf.keras.layers.Dense(units=15,activation='relu',kernel_initializer='he_uniform',kernel_regularizer=l2(param),
                    kernel_constraint=unit_norm()))
     model2.add(tf.keras.layers.BatchNormalization())
     model2.add(tf.keras.layers.Dropout(0.5))
     model2.add(tf.keras.layers.Dense(units=10,activation='relu',kernel_initializer='he_uniform',kernel_regularizer=l2(param),
                    kernel_constraint=unit_norm()))
     model2.add(tf.keras.layers.BatchNormalization())
     model2.add(tf.keras.layers.Dropout(0.5))
     model2.add(tf.keras.layers.Dense(units=10,activation='relu',kernel_initializer='he_uniform',kernel_regularizer=l2(param),
                    kernel_constraint=unit_norm()))
     model2.add(tf.keras.layers.BatchNormalization())
     model2.add(tf.keras.layers.Dropout(0.5))
     model2.add(tf.keras.layers.Dense(units=5,activation='softmax',kernel_regularizer=l2(param),
                    kernel_constraint=unit_norm()))
```

- Compiling the model

```
[ ]  #Compiling the model
     model2.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['Accuracy'])
```

- Training the Neural Network

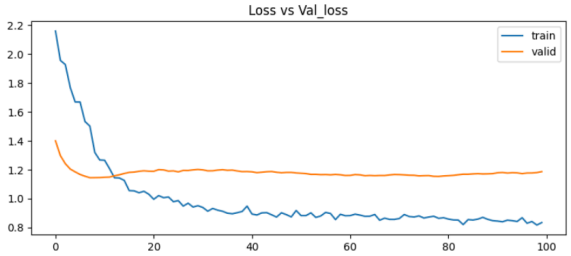
```
#Training the Neural Network
history = model2.fit(x_train,y_train,batch_size=8,epochs=100,validation_data=(x_test,y_test))
```

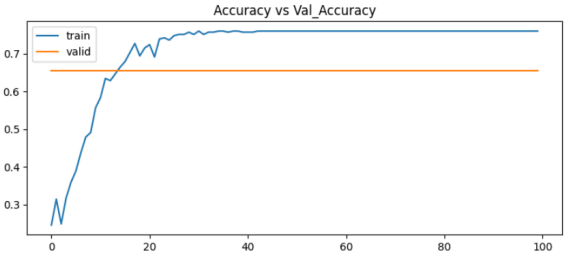
```
Epoch 1/100
42/42 [==============================] - 5s 16ms/step - loss: 2.1599 - Accuracy: 0.2455 - val_loss: 1.3998 - val_Accuracy: 0.6548
Epoch 2/100
42/42 [==============================] - 0s 7ms/step - loss: 1.9565 - Accuracy: 0.3144 - val_loss: 1.2971 - val_Accuracy: 0.6548
Epoch 3/100
42/42 [==============================] - 0s 8ms/step - loss: 1.9280 - Accuracy: 0.2485 - val_loss: 1.2418 - val_Accuracy: 0.6548
Epoch 4/100
42/42 [==============================] - 0s 4ms/step - loss: 1.7699 - Accuracy: 0.3174 - val_loss: 1.2049 - val_Accuracy: 0.6548
Epoch 5/100
42/42 [==============================] - 0s 4ms/step - loss: 1.6690 - Accuracy: 0.3593 - val_loss: 1.1851 - val_Accuracy: 0.6548
Epoch 6/100
42/42 [==============================] - 0s 4ms/step - loss: 1.6688 - Accuracy: 0.3892 - val_loss: 1.1672 - val_Accuracy: 0.6548
Epoch 7/100
42/42 [==============================] - 0s 4ms/step - loss: 1.5339 - Accuracy: 0.4371 - val_loss: 1.1551 - val_Accuracy: 0.6548
Epoch 8/100
42/42 [==============================] - 0s 4ms/step - loss: 1.5018 - Accuracy: 0.4790 - val_loss: 1.1449 - val_Accuracy: 0.6548
Epoch 9/100
42/42 [==============================] - 0s 4ms/step - loss: 1.3195 - Accuracy: 0.4910 - val_loss: 1.1453 - val_Accuracy: 0.6548
Epoch 10/100
42/42 [==============================] - 0s 4ms/step - loss: 1.2676 - Accuracy: 0.5569 - val_loss: 1.1459 - val_Accuracy: 0.6548
Epoch 11/100
42/42 [==============================] - 0s 4ms/step - loss: 1.2657 - Accuracy: 0.5838 - val_loss: 1.1477 - val_Accuracy: 0.6548
Epoch 12/100
42/42 [==============================] - 0s 4ms/step - loss: 1.2076 - Accuracy: 0.6347 - val_loss: 1.1490 - val_Accuracy: 0.6548
Epoch 13/100
42/42 [==============================] - 0s 4ms/step - loss: 1.1446 - Accuracy: 0.6287 - val_loss: 1.1582 - val_Accuracy: 0.6548
Epoch 14/100
42/42 [==============================] - 0s 4ms/step - loss: 1.1427 - Accuracy: 0.6467 - val_loss: 1.1648 - val_Accuracy: 0.6548
Epoch 15/100
42/42 [==============================] - 0s 4ms/step - loss: 1.1260 - Accuracy: 0.6647 - val_loss: 1.1741 - val_Accuracy: 0.6548
Epoch 16/100
42/42 [==============================] - 0s 4ms/step - loss: 1.0556 - Accuracy: 0.6796 - val_loss: 1.1813 - val_Accuracy: 0.6548
Epoch 17/100
42/42 [==============================] - 0s 4ms/step - loss: 1.0538 - Accuracy: 0.7036 - val_loss: 1.1838 - val_Accuracy: 0.6548
Epoch 85/100
42/42 [==============================] - 0s 4ms/step - loss: 0.8547 - Accuracy: 0.7605 - val_loss: 1.1689 - val_Accuracy: 0.6548
Epoch 86/100
42/42 [==============================] - 0s 4ms/step - loss: 0.8517 - Accuracy: 0.7605 - val_loss: 1.1712 - val_Accuracy: 0.6548
Epoch 87/100
42/42 [==============================] - 0s 4ms/step - loss: 0.8583 - Accuracy: 0.7605 - val_loss: 1.1728 - val_Accuracy: 0.6548
Epoch 88/100
42/42 [==============================] - 0s 4ms/step - loss: 0.8702 - Accuracy: 0.7605 - val_loss: 1.1710 - val_Accuracy: 0.6548
Epoch 89/100
42/42 [==============================] - 0s 5ms/step - loss: 0.8565 - Accuracy: 0.7605 - val_loss: 1.1716 - val_Accuracy: 0.6548
Epoch 90/100
42/42 [==============================] - 0s 4ms/step - loss: 0.8475 - Accuracy: 0.7605 - val_loss: 1.1730 - val_Accuracy: 0.6548
Epoch 91/100
42/42 [==============================] - 0s 4ms/step - loss: 0.8446 - Accuracy: 0.7605 - val_loss: 1.1709 - val_Accuracy: 0.6548
Epoch 92/100
42/42 [==============================] - 0s 4ms/step - loss: 0.8398 - Accuracy: 0.7605 - val_loss: 1.1805 - val_Accuracy: 0.6548
Epoch 93/100
42/42 [==============================] - 0s 4ms/step - loss: 0.8509 - Accuracy: 0.7605 - val_loss: 1.1771 - val_Accuracy: 0.6548
Epoch 94/100
42/42 [==============================] - 0s 5ms/step - loss: 0.8473 - Accuracy: 0.7605 - val_loss: 1.1793 - val_Accuracy: 0.6548
Epoch 95/100
```

- Plotting Loss and accuracy curve vs epochs

```
#Plotting loss and accuracy curve vs epochs
plt.figure(figsize=(20,8))
plt.subplot(2,2,1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(("train" , "valid") , loc =0)
plt.title("Loss vs Val_loss")
plt.subplot(2,2,2)
plt.plot(history.history['Accuracy'])
plt.plot(history.history['val_Accuracy'])
plt.legend(("train" , "valid") , loc =0)
plt.title("Accuracy vs Val_Accuracy")
```

```
Text(0.5, 1.0, 'Accuracy vs Val_Accuracy')
```



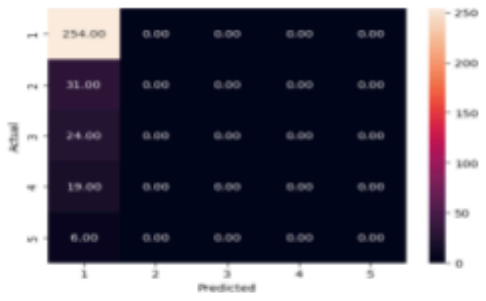


- Creating function to display Evaluation matrix

```
evaluation_matrix(model2,x_train,y_train,x_test,y_test)
```

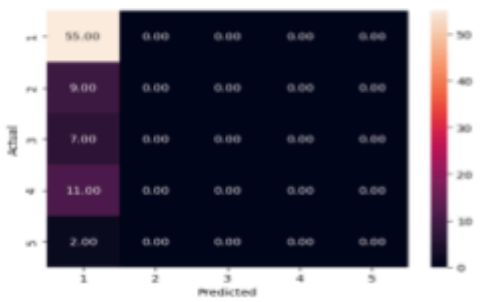
```
11/11 [==============================] - 0s 2ms/step
3/3 [==============================] - 0s 4ms/step
Train Recall: 76.05
Test Recall: 65.48
Train Precision: 57.83
Test Precision: 42.87
Train F1: 65.7
Test F1: 51.82
```

```
Confusion matrix for training set:
```




```
**********************************************************************

Classification report for Training set:

              precision    recall  f1-score   support

           1       0.76      1.00      0.86       254
           2       0.00      0.00      0.00        31
           3       0.00      0.00      0.00        24
           4       0.00      0.00      0.00        19
           5       0.00      0.00      0.00         6

    accuracy                           0.76       334
   macro avg       0.15      0.20      0.17       334
weighted avg       0.58      0.76      0.66       334
```

```
Confusion matrix for testing set:
```




```
Classification report for Testing set:

              precision    recall  f1-score   support

           1       0.65      1.00      0.79        55
           2       0.00      0.00      0.00         9
           3       0.00      0.00      0.00         7
           4       0.00      0.00      0.00        11
           5       0.00      0.00      0.00         2

    accuracy                           0.65        84
   macro avg       0.13      0.20      0.16        84
weighted avg       0.43      0.65      0.52        84

**********************************************************************
```

We have got much improved score from the neural network model hence saving it

- Saving the model

```
[ ]  #saving the model
     model2.save('model2.h5')
```

Training the model with SMOTE data

- Building Neural Network3

```
[ ]  #building Neural nEtworks 3
     param=1e-4
     model3 = tf.keras.models.Sequential()
     model3.add(tf.keras.layers.Dense(units=10,activation='relu',kernel_initializer='he_uniform',kernel_regularizer=l2(param),
                    kernel_constraint=unit_norm()))
     model3.add(tf.keras.layers.BatchNormalization())
     model3.add(tf.keras.layers.Dropout(0.5))
     model3.add(tf.keras.layers.Dense(units=20,activation='relu',kernel_initializer='he_uniform',kernel_regularizer=l2(param),
                    kernel_constraint=unit_norm()))
     model3.add(tf.keras.layers.BatchNormalization())
     model3.add(tf.keras.layers.Dropout(0.5))
     model3.add(tf.keras.layers.Dense(units=30,activation='relu',kernel_initializer='he_uniform',kernel_regularizer=l2(param),
                    kernel_constraint=unit_norm()))
     model3.add(tf.keras.layers.BatchNormalization())
     model3.add(tf.keras.layers.Dropout(0.5))
     model3.add(tf.keras.layers.Dense(units=20,activation='relu',kernel_initializer='he_uniform',kernel_regularizer=l2(param),
                    kernel_constraint=unit_norm()))
     model3.add(tf.keras.layers.BatchNormalization())
     model3.add(tf.keras.layers.Dropout(0.5))
     model3.add(tf.keras.layers.Dense(units=10,activation='relu',kernel_initializer='he_uniform',kernel_regularizer=l2(param),
                    kernel_constraint=unit_norm()))
     model3.add(tf.keras.layers.BatchNormalization())
     model3.add(tf.keras.layers.Dropout(0.5))
     model3.add(tf.keras.layers.Dense(units=5,activation='softmax',kernel_regularizer=l2(param),
                    kernel_constraint=unit_norm()))
```

- Compiling the model

```
[ ]  #Compiling the model
     model3.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['Accuracy'])
```

- Training the Neural Network

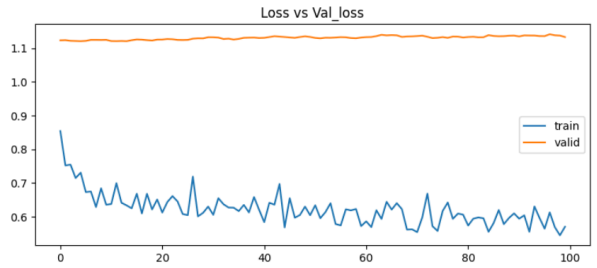
```
#Training the Neural Network
history = model3.fit(x_train_sample,y_train_sample,batch_size=12,epochs=100,validation_data=(x_test,y_test))
```

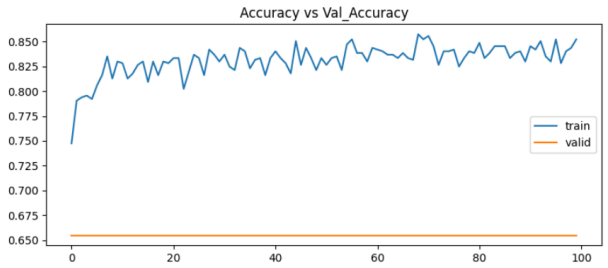
```
Epoch 1/100
49/49 [==============================] - 0s 10ms/step - loss: 0.8541 - Accuracy: 0.7474 - val_loss: 1.1232 - val_Accuracy: 0.6548
Epoch 2/100
49/49 [==============================] - 0s 4ms/step - loss: 0.7518 - Accuracy: 0.7904 - val_loss: 1.1237 - val_Accuracy: 0.6548
Epoch 3/100
49/49 [==============================] - 0s 4ms/step - loss: 0.7544 - Accuracy: 0.7938 - val_loss: 1.1217 - val_Accuracy: 0.6548
Epoch 4/100
49/49 [==============================] - 0s 4ms/step - loss: 0.7146 - Accuracy: 0.7955 - val_loss: 1.1212 - val_Accuracy: 0.6548
Epoch 5/100
49/49 [==============================] - 0s 4ms/step - loss: 0.7305 - Accuracy: 0.7921 - val_loss: 1.1207 - val_Accuracy: 0.6548
Epoch 6/100
49/49 [==============================] - 0s 4ms/step - loss: 0.6729 - Accuracy: 0.8058 - val_loss: 1.1216 - val_Accuracy: 0.6548
Epoch 7/100
49/49 [==============================] - 0s 4ms/step - loss: 0.6746 - Accuracy: 0.8162 - val_loss: 1.1245 - val_Accuracy: 0.6548
Epoch 8/100
49/49 [==============================] - 0s 4ms/step - loss: 0.6285 - Accuracy: 0.8351 - val_loss: 1.1245 - val_Accuracy: 0.6548
Epoch 9/100
49/49 [==============================] - 0s 4ms/step - loss: 0.6841 - Accuracy: 0.8127 - val_loss: 1.1243 - val_Accuracy: 0.6548
Epoch 10/100
49/49 [==============================] - 0s 4ms/step - loss: 0.6355 - Accuracy: 0.8299 - val_loss: 1.1246 - val_Accuracy: 0.6548
Epoch 11/100
49/49 [==============================] - 0s 4ms/step - loss: 0.6375 - Accuracy: 0.8282 - val_loss: 1.1210 - val_Accuracy: 0.6548
Epoch 12/100
49/49 [==============================] - 0s 4ms/step - loss: 0.6994 - Accuracy: 0.8127 - val_loss: 1.1200 - val_Accuracy: 0.6548
Epoch 13/100
49/49 [==============================] - 0s 4ms/step - loss: 0.6431 - Accuracy: 0.8179 - val_loss: 1.1212 - val_Accuracy: 0.6548
Epoch 14/100
49/49 [==============================] - 0s 4ms/step - loss: 0.6334 - Accuracy: 0.8265 - val_loss: 1.1207 - val_Accuracy: 0.6548
Epoch 15/100
49/49 [==============================] - 0s 4ms/step - loss: 0.6247 - Accuracy: 0.8299 - val_loss: 1.1236 - val_Accuracy: 0.6548
Epoch 16/100
49/49 [==============================] - 0s 4ms/step - loss: 0.6680 - Accuracy: 0.8093 - val_loss: 1.1258 - val_Accuracy: 0.6548
Epoch 17/100
49/49 [==============================] - 0s 4ms/step - loss: 0.6095 - Accuracy: 0.8299 - val_loss: 1.1252 - val_Accuracy: 0.6548
Epoch 83/100
49/49 [==============================] - 0s 4ms/step - loss: 0.5976 - Accuracy: 0.8385 - val_loss: 1.1321 - val_Accuracy: 0.6548
Epoch 84/100
49/49 [==============================] - 0s 5ms/step - loss: 0.5951 - Accuracy: 0.8454 - val_loss: 1.1324 - val_Accuracy: 0.6548
Epoch 85/100
49/49 [==============================] - 0s 6ms/step - loss: 0.5551 - Accuracy: 0.8454 - val_loss: 1.1386 - val_Accuracy: 0.6548
Epoch 86/100
49/49 [==============================] - 0s 6ms/step - loss: 0.5812 - Ac[redacted] 0.8454 - val_loss: 1.1361 - val_Accuracy: 0.6548
Epoch 87/100
49/49 [==============================] - 0s 6ms/step - loss: 0.6196 - Ac[redacted] 0.8333 - val_loss: 1.1352 - val_Accuracy: 0.6548
Epoch 88/100
49/49 [==============================] - 0s 6ms/step - loss: 0.5779 - Ac[redacted] 0.8385 - val_loss: 1.1356 - val_Accuracy: 0.6548
Epoch 89/100
49/49 [==============================] - 0s 6ms/step - loss: 0.5959 - Accuracy: 0.8402 - val_loss: 1.1368 - val_Accuracy: 0.6548
Epoch 90/100
```

- Plotting Loss and accuracy curve vs epochs

```
#Plotting loss and accuracy curve vs epochs
plt.figure(figsize=(20,8))
plt.subplot(2,2,1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(("train" , "valid") , loc =0)
plt.title("Loss vs Val_loss")
plt.subplot(2,2,2)
plt.plot(history.history['Accuracy'])
plt.plot(history.history['val_Accuracy'])
plt.legend(("train" , "valid") , loc =0)
plt.title("Accuracy vs Val_Accuracy")
```

```
Text(0.5, 1.0, 'Accuracy vs Val_Accuracy')
```



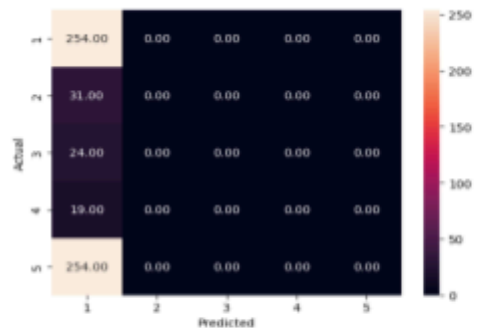


- Creating function to display Evaluation matrix

```
evaluation_matrix(model2,x_train_sample,y_train_sample,x_test,y_test)
```

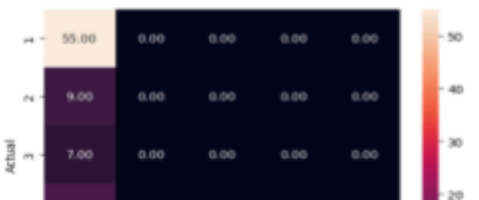
```
19/19 [==============================] - 0s 2ms/step
3/3 [==============================] - 0s 3ms/step
Train Recall: 43.64
Test Recall: 65.48
Train Precision: 19.05
Test Precision: 42.87
Train F1: 26.52
Test F1: 51.82
```

```
Confusion matrix for training set:
```




```
**********************************************************************
Classification report for Training set:
              precision    recall  f1-score   support

           1       0.44      1.00      0.61       254
           2       0.00      0.00      0.00        31
           3       0.00      0.00      0.00        24
           4       0.00      0.00      0.00        19
           5       0.00      0.00      0.00       254

    accuracy                           0.44       582
   macro avg       0.09      0.20      0.12       582
weighted avg       0.19      0.44      0.27       582
```

```
Confusion matrix for testing set:
```




```
Classification report for Testing set:
              precision    recall  f1-score   support

           1       0.65      1.00      0.79        55
           2       0.00      0.00      0.00         9
           3       0.00      0.00      0.00         7
           4       0.00      0.00      0.00        11
           5       0.00      0.00      0.00         2

    accuracy                           0.65        84
   macro avg       0.13      0.20      0.16        84
```

-

- The model performance is not improving even after using SMOTE data. Let's try LSTM model

Step2: Design, train and test LSTM classifiers

Data Preprocessing for LSTM model

- Word Embeddings using glove vector
- Vectorization using tfidf

```
[ ]  #Vectorization using tfidf
     vec = TfidfVectorizer()
```

- Splitting the data into train and test

```
[ ]  #Splitting data into train and test
     x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=0.8,random_state=2)
```

- Tokenizing the text

```
[ ]  #tokenizing the text
     tokenizer = Tokenizer()
     tokenizer.fit_on_texts(x_train)
     x_train_1 = tokenizer.texts_to_sequences(x_train)
     x_test_1 = tokenizer.texts_to_sequences(x_test)
```

- Pickling the tokenizer

```
     #pickling the tokenizer
     filename = "/content/drive/My Drive/tokenizer1.pickle"
     pickle.dump(tokenizer, open(filename, 'wb'))
```

- Finding the length of vocabulary and maximum Length of text

```
#Finding the length of vocabulary and maximum length of text

vocab = len(tokenizer.word_index) + 1
print("vocabulary size::", vocab)

maxlen = max(len(text) for text in x)
print("Maximum length:", maxlen)
```

```
vocabulary size:: 2356
Maximum length: 562
```

- Padding the data using maxlen

```
#padding the data using maxlen
x_train_1 = pad_sequences(x_train_1,padding='post',maxlen=maxlen)
x_test_1 = pad_sequences(x_test_1,padding='post',maxlen=maxlen)
```

```
x_train_2 = vec.fit_transform(x_train).toarray()
x_test_2 = vec.transform(x_test).toarray()
```

- Pickling the vectorizer

```
[ ] #Pickling the vectorizer
    filename = "/content/drive/My Drive/vec1.pickle"
    pickle.dump(vec, open(filename, 'wb'))
```

```
[ ] y_train = pd.get_dummies(y_train).values
    y_test = pd.get_dummies(y_test).values
```

- Glove Embeddings

```
# We need to load the built-in GloVe word embeddings
embedding_size = 200
embeddings_dictionary = dict()

glove_file = open('/content/drive/My Drive/glove.6B.200d.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, 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
```

```
[ ] x_test_1
```

```
array([[   9,  439,  156, ...,    0,    0,    0],
       [1602,   14,  103, ...,    0,    0,    0],
       [  17, 1845,   63, ...,    0,    0,    0],
       ...,
       [ 519,   13,   80, ...,    0,    0,    0],
       [  43, 1202,   58, ...,    0,    0,    0],
       [1030,  254,    2, ...,    0,    0,    0]], dtype=int32)
```

Building LSTM model1

```
[ ] model_lstm = tf.keras.models.Sequential()
    model_lstm.add(tf.keras.Input(shape=(maxlen,)))
    model_lstm.add(tf.keras.layers.Embedding(vocab, embedding_size, weights=[embedding_matrix], trainable=False))
    model_lstm.add(tf.keras.layers.LSTM(256,return_sequences = True))
    model_lstm.add(tf.keras.layers.Dropout(0.2))
    model_lstm.add(tf.keras.layers.LSTM(128))
    model_lstm.add(tf.keras.layers.Dense(128,activation='relu'))
    model_lstm.add(tf.keras.layers.Dropout(0.5))
    model_lstm.add(tf.keras.layers.Dense(64,activation='relu'))
    model_lstm.add(tf.keras.layers.Dropout(0.5))
    model_lstm.add(tf.keras.layers.Dense(5,activation='softmax'))
```

```
[ ] model_lstm.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

- LSTM Model 1 Summary

```
[ ]  model_lstm.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_6 (Embedding)     (None, 562, 200)          471200

 lstm_7 (LSTM)               (None, 562, 256)          467968

 dropout_36 (Dropout)        (None, 562, 256)          0

 lstm_8 (LSTM)               (None, 128)               197120

 dense_40 (Dense)            (None, 128)               16512

 dropout_37 (Dropout)        (None, 128)               0

 dense_41 (Dense)            (None, 64)                8256

 dropout_38 (Dropout)        (None, 64)                0

 dense_42 (Dense)            (None, 5)                 325

=================================================================
Total params: 1161381 (4.43 MB)
Trainable params: 690181 (2.63 MB)
Non-trainable params: 471200 (1.80 MB)
_________________________________________________________________
```

- Fit the keras model on the dataset

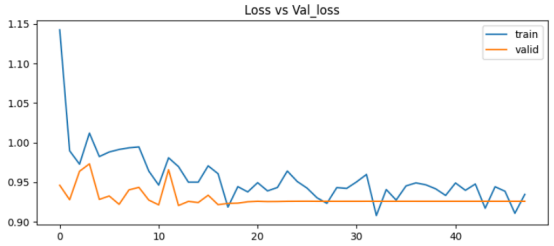
```
[ ] callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=15, min_delta=1e-3)
    rlrp = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5, min_delta=1e-6)

    # fit the keras model on the dataset
    history = model_lstm.fit(x_train_1,y_train,batch_size=8,epochs=100,verbose=1,validation_data=(x_test_1,y_test),callbacks=[rlrp,callback])

Epoch 1/100
42/42 [==============================] - 81s 2s/step - loss: 1.1424 - accuracy: 0.7066 - val_loss: 0.9461 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 2/100
42/42 [==============================] - 75s 2s/step - loss: 0.9896 - accuracy: 0.7365 - val_loss: 0.9278 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 3/100
42/42 [==============================] - 78s 2s/step - loss: 0.9727 - accuracy: 0.7455 - val_loss: 0.9638 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 4/100
42/42 [==============================] - 75s 2s/step - loss: 1.0120 - accuracy: 0.7425 - val_loss: 0.9731 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 5/100
42/42 [==============================] - 77s 2s/step - loss: 0.9824 - accuracy: 0.7425 - val_loss: 0.9282 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 6/100
42/42 [==============================] - 76s 2s/step - loss: 0.9881 - accuracy: 0.7425 - val_loss: 0.9326 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 7/100
42/42 [==============================] - 81s 2s/step - loss: 0.9913 - accuracy: 0.7425 - val_loss: 0.9220 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 8/100
42/42 [==============================] - 81s 2s/step - loss: 0.9934 - accuracy: 0.7425 - val_loss: 0.9403 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 9/100
42/42 [==============================] - 77s 2s/step - loss: 0.9965 - accuracy: 0.7425 - val_loss: 0.9434 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 10/100
42/42 [==============================] - 75s 2s/step - loss: 0.9660 - accuracy: 0.7425 - val_loss: 0.9274 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 11/100
42/42 [==============================] - 78s 2s/step - loss: 0.9463 - accuracy: 0.7425 - val_loss: 0.9213 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 12/100
42/42 [==============================] - 76s 2s/step - loss: 0.9808 - accuracy: 0.7425 - val_loss: 0.9657 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 13/100
42/42 [==============================] - 77s 2s/step - loss: 0.9697 - accuracy: 0.7425 - val_loss: 0.9285 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 14/100
42/42 [==============================] - 77s 2s/step - loss: 0.9501 - accuracy: 0.7425 - val_loss: 0.9257 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 15/100
42/42 [==============================] - 81s 2s/step - loss: 0.9501 - accuracy: 0.7425 - val_loss: 0.9243 - val_accuracy: 0.7262 - lr: 0.0010

Epoch 34/100
42/42 [==============================] - 83s 2s/step - loss: 0.9407 - accuracy: 0.7425 - val_loss: 0.9259 - val_accuracy: 0.7262 - lr: 1.0000e-07
Epoch 35/100
42/42 [==============================] - 76s 2s/step - loss: 0.9274 - accuracy: 0.7425 - val_loss: 0.9259 - val_accuracy: 0.7262 - lr: 1.0000e-07
Epoch 36/100
42/42 [==============================] - 75s 2s/step - loss: 0.9453 - accuracy: 0.7425 - val_loss: 0.9259 - val_accuracy: 0.7262 - lr: 1.0000e-07
Epoch 37/100
42/42 [==============================] - 78s 2s/step - loss: 0.9491 - accuracy: 0.7425 - val_loss: 0.9259 - val_accuracy: 0.7262 - lr: 1.0000e-07
Epoch 38/100
42/42 [==============================] - 78s 2s/step - loss: 0.9467 - accuracy: 0.7425 - val_loss: 0.9259 - val_accuracy: 0.7262 - lr: 1.0000e-07
Epoch 39/100
42/42 [==============================] - 77s 2s/step - loss: 0.9416 - accuracy: 0.7425 - val_loss: 0.9259 - val_accuracy: 0.7262 - lr: 1.0000e-08
Epoch 40/100
42/42 [==============================] - 76s 2s/step - loss: 0.9332 - accuracy: 0.7425 - val_loss: 0.9259 - val_accuracy: 0.7262 - lr: 1.0000e-08
Epoch 41/100
42/42 [==============================] - 76s 2s/step - loss: 0.9490 - accuracy: 0.7425 - val_loss: 0.9259 - val_accuracy: 0.7262 - lr: 1.0000e-08
Epoch 42/100
42/42 [==============================] - 83s 2s/step - loss: 0.9397 - accuracy: 0.7425 - val_loss: 0.9259 - val_accuracy: 0.7262 - lr: 1.0000e-08
Epoch 43/100
42/42 [==============================] - 76s 2s/step - loss: 0.9477 - accuracy: 0.7425 - val_loss: 0.9259 - val_accuracy: 0.7262 - lr: 1.0000e-08
Epoch 44/100
42/42 [==============================] - 76s 2s/step - loss: 0.9171 - accuracy: 0.7425 - val_loss: 0.9259 - val_accuracy: 0.7262 - lr: 1.0000e-09
Epoch 45/100
42/42 [==============================] - 77s 2s/step - loss: 0.9442 - accuracy: 0.7425 - val_loss: 0.9259 - val_accuracy: 0.7262 - lr: 1.0000e-09
Epoch 46/100
42/42 [==============================] - 78s 2s/step - loss: 0.9386 - accuracy: 0.7425 - val_loss: 0.9259 - val_accuracy: 0.7262 - lr: 1.0000e-09
Epoch 47/100
42/42 [==============================] - 75s 2s/step - loss: 0.9187 - accuracy: 0.7425 - val_loss: 0.9259 - val_accuracy: 0.7262 - lr: 1.0000e-09
Epoch 48/100
42/42 [==============================] - 75s 2s/step - loss: 0.9346 - accuracy: 0.7425 - val_loss: 0.9259 - val_accuracy: 0.7262 - lr: 1.0000e-09
```

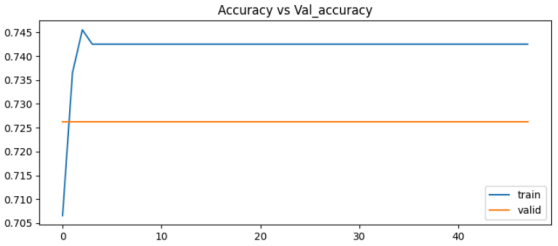
```
[ ]  #Saving the model
     model_lstm.save('model_lstm.h5')
```

- Plotting Loss and accuracy curve vs epochs

```
[ ]  #Plotting loss and accuracy curve vs epochs
     plt.figure(figsize=(20,8))
     plt.subplot(2,2,1)
     plt.plot(history.history['loss'])
     plt.plot(history.history['val_loss'])
     plt.legend(("train" , "valid") , loc =0)
     plt.title("Loss vs Val_loss")
     plt.subplot(2,2,2)
     plt.plot(history.history['accuracy'])
     plt.plot(history.history['val_accuracy'])
     plt.legend(("train" , "valid") , loc =0)
     plt.title("Accuracy vs Val_accuracy")
```

```
Text(0.5, 1.0, 'Accuracy vs Val_accuracy')
```

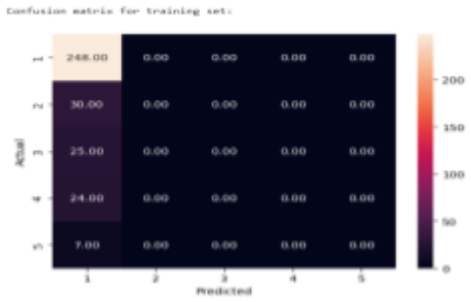




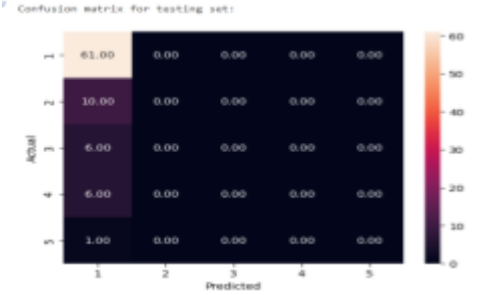
- Creating function to display Evaluation matrix

```
evaluation_matrix(model_lstm,x_train_1,y_train,x_test_1,y_test)
```

```
11/11 [==============================] - 11s 954ms/step
3/3 [==============================] - 2s 729ms/step
Train Recall: 74.25
Test Recall: 72.62
Train Precision: 55.13
Test Precision: 52.74
Train F1: 63.28
Test F1: 61.1
```




```
**********************************************************************

Classification report for Training set:

              precision    recall  f1-score   support

           1       0.74      1.00      0.85       248
           2       0.00      0.00      0.00        30
           3       0.00      0.00      0.00        25
           4       0.00      0.00      0.00        24
           5       0.00      0.00      0.00         7

    accuracy                           0.74       334
   macro avg       0.15      0.20      0.17       334
weighted avg       0.55      0.74      0.63       334
```




```
Classification report for Testing set:

              precision    recall  f1-score   support

           1       0.73      1.00      0.84        61
           2       0.00      0.00      0.00        10
           3       0.00      0.00      0.00         6
           4       0.00      0.00      0.00         6
           5       0.00      0.00      0.00         1

    accuracy                           0.73        84
   macro avg       0.15      0.20      0.17        84
weighted avg       0.53      0.73      0.61        84

**********************************************************************
```

Test F1 score of the model improved to 61.1%

Building LSTM model 2 with Bidirectional layer

- Build a LSTM Neural Network

```
# Build a LSTM Neural Network
deep_inputs = tf.keras.Input(shape=(maxlen,))
embedding_layer = tf.keras.layers.Embedding(vocab, embedding_size, weights=[embedding_matrix], trainable=False)(deep_inputs)

LSTM_Layer_1 = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128, return_sequences = True))(embedding_layer)
max_pool_layer_1 = tf.keras.layers.GlobalAveragePooling1D()(LSTM_Layer_1)
drop_out_layer_1 = tf.keras.layers.Dropout(0.5, input_shape = (256,))(max_pool_layer_1)
dense_layer_1 = tf.keras.layers.Dense(128, activation = 'relu')(drop_out_layer_1)
drop_out_layer_2 = tf.keras.layers.Dropout(0.5, input_shape = (128,))(dense_layer_1)
dense_layer_2 = tf.keras.layers.Dense(64, activation = 'relu')(drop_out_layer_2)
drop_out_layer_3 = tf.keras.layers.Dropout(0.5, input_shape = (64,))(dense_layer_2)

dense_layer_3 = tf.keras.layers.Dense(32, activation = 'relu')(drop_out_layer_3)
drop_out_layer_4 = tf.keras.layers.Dropout(0.5, input_shape = (32,))(dense_layer_3)

dense_layer_4 = tf.keras.layers.Dense(10, activation = 'relu')(drop_out_layer_4)
drop_out_layer_5 = tf.keras.layers.Dropout(0.5, input_shape = (10,))(dense_layer_4)

dense_layer_5 = tf.keras.layers.Dense(5, activation='softmax')(drop_out_layer_5)

model_lstm1 = tf.keras.Model(inputs=deep_inputs, outputs=dense_layer_5)

model_lstm1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

- Fit the keras model on the dataset

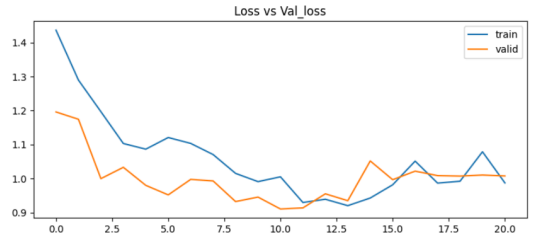
```
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=7, min_delta=1e-6)
rlrp = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.125, patience=5, min_delta=1e-6)

# fit the keras model on the dataset
history = model_lstm1.fit(x_train_1,y_train,batch_size=8,epochs=100,verbose=1,validation_data=(x_test_1,y_test),callbacks=[rlrp,callback])
```

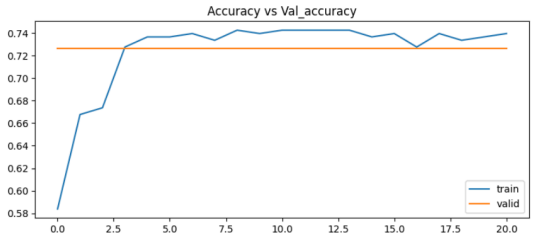
```
Epoch 1/100
42/42 [==============================] - 48s 957ms/step - loss: 1.4366 - accuracy: 0.5838 - val_loss: 1.1959 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 2/100
42/42 [==============================] - 39s 924ms/step - loss: 1.2897 - accuracy: 0.6677 - val_loss: 1.1744 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 3/100
42/42 [==============================] - 41s 972ms/step - loss: 1.1968 - accuracy: 0.6737 - val_loss: 0.9996 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 4/100
42/42 [==============================] - 39s 936ms/step - loss: 1.1030 - accuracy: 0.7275 - val_loss: 1.0330 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 5/100
42/42 [==============================] - 41s 972ms/step - loss: 1.0864 - accuracy: 0.7365 - val_loss: 0.9799 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 6/100
42/42 [==============================] - 40s 962ms/step - loss: 1.1207 - accuracy: 0.7365 - val_loss: 0.9518 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 7/100
42/42 [==============================] - 42s 1s/step - loss: 1.1034 - accuracy: 0.7395 - val_loss: 0.9974 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 8/100
42/42 [==============================] - 40s 963ms/step - loss: 1.0705 - accuracy: 0.7335 - val_loss: 0.9929 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 9/100
42/42 [==============================] - 40s 959ms/step - loss: 1.0151 - accuracy: 0.7425 - val_loss: 0.9521 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 10/100
42/42 [==============================] - 38s 900ms/step - loss: 0.9908 - accuracy: 0.7395 - val_loss: 0.9453 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 11/100
42/42 [==============================] - 38s 902ms/step - loss: 1.0050 - accuracy: 0.7425 - val_loss: 0.9103 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 12/100
42/42 [==============================] - 39s 928ms/step - loss: 0.9294 - accuracy: 0.7425 - val_loss: 0.9135 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 13/100
42/42 [==============================] - 39s 943ms/step - loss: 0.9389 - accuracy: 0.7425 - val_loss: 0.9550 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 14/100
42/42 [==============================] - 39s 944ms/step - loss: 0.9201 - accuracy: 0.7425 - val_loss: 0.9347 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 15/100
42/42 [==============================] - 39s 957ms/step - loss: 0.9426 - accuracy: 0.7365 - val_loss: 1.0516 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 16/100
42/42 [==============================] - 40s 962ms/step - loss: 0.9815 - accuracy: 0.7395 - val_loss: 0.9965 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 17/100
42/42 [==============================] - 38s 910ms/step - loss: 1.0512 - accuracy: 0.7275 - val_loss: 1.0217 - val_accuracy: 0.7262 - lr: 1.2500e-04
Epoch 18/100
42/42 [==============================] - 39s 909ms/step - loss: 0.9865 - accuracy: 0.7395 - val_loss: 1.0085 - val_accuracy: 0.7262 - lr: 1.2500e-04
Epoch 19/100
42/42 [==============================] - 39s 933ms/step - loss: 0.9920 - accuracy: 0.7335 - val_loss: 1.0072 - val_accuracy: 0.7262 - lr: 1.2500e-04
Epoch 20/100
42/42 [==============================] - 39s 932ms/step - loss: 1.0787 - accuracy: 0.7365 - val_loss: 1.0103 - val_accuracy: 0.7262 - lr: 1.2500e-04
Epoch 21/100
42/42 [==============================] - 39s 943ms/step - loss: 0.9870 - accuracy: 0.7395 - val_loss: 1.0075 - val_accuracy: 0.7262 - lr: 1.2500e-04
```

- Plotting Loss and accuracy curve vs epochs

```
#Plotting loss and accuracy curve vs epochs
plt.figure(figsize=(20,8))
plt.subplot(2,2,1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(("train" , "valid") , loc =0)
plt.title("Loss vs Val_loss")
plt.subplot(2,2,2)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(("train" , "valid") , loc =0)
plt.title("Accuracy vs Val_accuracy")
```

```
Text(0.5, 1.0, 'Accuracy vs Val_accuracy')
```



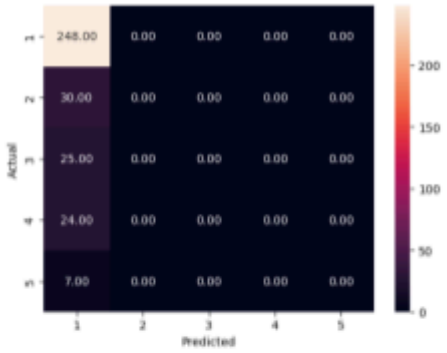


- Creating function to display Evaluation matrix

```
[ ] evaluation_matrix(model_lstm1,x_train_1,y_train,x_test_1,y_test)
```

```
11/11 [==============================] - 14s 827ms/step
3/3 [==============================] - 4s 1s/step
Train Recall: 74.25
Test Recall: 72.62
Train Precision: 55.13
Test Precision: 52.74
Train F1: 63.28
Test F1: 61.1
```

Confusion matrix for training set:

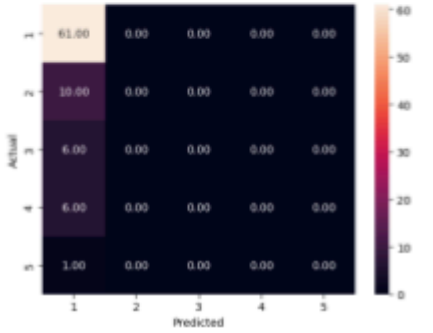


****

Classification report for Training set:

	precision	recall	f1-score	support
1	0.74	1.00	0.85	248
2	0.00	0.00	0.00	30
3	0.00	0.00	0.00	25
4	0.00	0.00	0.00	24
5	0.00	0.00	0.00	7
accuracy			0.74	334
macro avg	0.15	0.20	0.17	334
weighted avg	0.55	0.74	0.63	334

Confusion matrix for testing set:



Classification report for Testing set:

	precision	recall	f1-score	support
1	0.73	1.00	0.84	61
2	0.00	0.00	0.00	10
3	0.00	0.00	0.00	6
4	0.00	0.00	0.00	6
5	0.00	0.00	0.00	1
accuracy			0.73	84
macro avg	0.15	0.20	0.17	84
weighted avg	0.53	0.73	0.61	84

****

- Saving the model

```
[ ] #Saving the model
    model_lstm1.save('model_lstm1.h5)')
```

Building LSTM model 3

- Build a LSTM Neural Network

```
# Build a LSTM Neural Network
input_1 = tf.keras.Input(shape=(maxlen,))
embedding_layer = tf.keras.layers.Embedding(vocab, embedding_size, weights=[embedding_matrix], trainable=False)(input_1)

LSTM_1 = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128, return_sequences = True))(embedding_layer)
maxpool_1 = tf.keras.layers.GlobalMaxPooling1D()(LSTM_1)
dropout_1 = tf.keras.layers.Dropout(0.5, input_shape = (256,))(maxpool_1)
dense_1 = tf.keras.layers.Dense(128, activation = 'relu')(dropout_1)
dropout_2 = tf.keras.layers.Dropout(0.5, input_shape = (128,))(dense_1)
dense_2 = tf.keras.layers.Dense(64, activation = 'relu')(dropout_2)
dropout_3 = tf.keras.layers.Dropout(0.5, input_shape = (64,))(dense_2)

dense_3 = tf.keras.layers.Dense(32, activation = 'relu')(dropout_3)
dropout_4 = tf.keras.layers.Dropout(0.5, input_shape = (32,))(dense_3)

dense_4 = tf.keras.layers.Dense(10, activation = 'relu')(dropout_4)
dropout_5 = tf.keras.layers.Dropout(0.5, input_shape = (32,))(dense_4)

input_2 = tf.keras.Input(shape=(x_train_2.shape[1],))
dense_5 = tf.keras.layers.Dense(10, input_dim=x_train_2.shape[1], activation='relu', kernel_initializer='he_uniform',
                    kernel_constraint=unit_norm())(input_2)
dropout_6 = tf.keras.layers.Dropout(0.2)(dense_5)
batchnorm_1  = tf.keras.layers.BatchNormalization()(dropout_6)
dense_6 = tf.keras.layers.Dense(10, activation='relu', kernel_initializer='he_uniform',
                        kernel_constraint=unit_norm())(batchnorm_1)
dropout_7 = tf.keras.layers.Dropout(0.5)(dense_6)
batchnorm_2 = tf.keras.layers.BatchNormalization()(dropout_7)

concat_layer = tf.keras.layers.Concatenate()([dropout_5, batchnorm_2])
dense_7 = tf.keras.layers.Dense(10, activation='relu')(concat_layer)
output = tf.keras.layers.Dense(5, activation='softmax')(dense_7)
model_lstm2 = tf.keras.models.Model(inputs=[input_1, input_2], outputs=output)

model_lstm2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model_lstm2.summary()
```

- Summary of the model

```
Model: "model_1"
```

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[(None, 562)]	0	[]
embedding_2 (Embedding)	(None, 562, 200)	471200	['input_3[0][0]']
bidirectional_1 (Bidirectional)	(None, 562, 256)	336896	['embedding_2[0][0]']
global_max_pooling1d (GlobalMaxPooling1D)	(None, 256)	0	['bidirectional_1[0][0]']
dropout_8 (Dropout)	(None, 256)	0	['global_max_pooling1d[0][0]']
dense_8 (Dense)	(None, 128)	32896	['dropout_8[0][0]']
dropout_9 (Dropout)	(None, 128)	0	['dense_8[0][0]']
input_4 (InputLayer)	[(None, 2340)]	0	[]
dense_9 (Dense)	(None, 64)	8256	['dropout_9[0][0]']
dense_12 (Dense)	(None, 10)	23410	['input_4[0][0]']
dropout_10 (Dropout)	(None, 64)	0	['dense_9[0][0]']
dropout_13 (Dropout)	(None, 10)	0	['dense_12[0][0]']
dense_10 (Dense)	(None, 32)	2080	['dropout_10[0][0]']
batch_normalization (BatchNormalization)	(None, 10)	40	['dropout_13[0][0]']
dropout_11 (Dropout)	(None, 32)	0	['dense_10[0][0]']
dense_13 (Dense)	(None, 10)	110	['batch_normalization[0][0]']
dense_11 (Dense)	(None, 10)	330	['dropout_11[0][0]']
dropout_14 (Dropout)	(None, 10)	0	['dense_13[0][0]']
dropout_12 (Dropout)	(None, 10)	0	['dense_11[0][0]']
batch_normalization_1 (BatchNormalization)	(None, 10)	40	['dropout_14[0][0]']
concatenate (Concatenate)	(None, 20)	0	['dropout_12[0][0]', 'batch_normalization_1[0][0]']
dense_14 (Dense)	(None, 10)	210	['concatenate[0][0]']
dense_15 (Dense)	(None, 5)	55	['dense_14[0][0]']

```
Total params: 875523 (3.34 MB)
Trainable params: 404283 (1.54 MB)
Non-trainable params: 471240 (1.80 MB)
```

- Fit the keras model on the dataset

```
[ ] callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=15, min_delta=1e-6)
    rlrp = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.125, patience=5, min_delta=1e-6)

    # fit the keras model on the dataset
    history = model_lstm2.fit([x_train_1,x_train_2],y_train,batch_size=8,epochs=100,verbose=1,validation_data=([x_test_1,x_test_2],y_test),callbacks=[rlrp,callback])
```

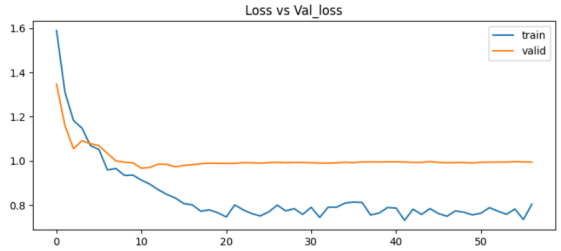
```
Epoch 1/100
42/42 [==============================] - 44s 959ms/step - loss: 1.5890 - accuracy: 0.2994 - val_loss: 1.3465 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 2/100
42/42 [==============================] - 40s 957ms/step - loss: 1.3099 - accuracy: 0.5449 - val_loss: 1.1588 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 3/100
42/42 [==============================] - 41s 991ms/step - loss: 1.1825 - accuracy: 0.6078 - val_loss: 1.0554 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 4/100
42/42 [==============================] - 39s 922ms/step - loss: 1.1478 - accuracy: 0.6766 - val_loss: 1.0906 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 5/100
42/42 [==============================] - 42s 995ms/step - loss: 1.0690 - accuracy: 0.7126 - val_loss: 1.0780 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 6/100
42/42 [==============================] - 41s 971ms/step - loss: 1.0511 - accuracy: 0.7126 - val_loss: 1.0682 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 7/100
42/42 [==============================] - 40s 960ms/step - loss: 0.9585 - accuracy: 0.7246 - val_loss: 1.0339 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 8/100
42/42 [==============================] - 39s 925ms/step - loss: 0.9652 - accuracy: 0.7455 - val_loss: 0.9994 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 9/100
42/42 [==============================] - 39s 928ms/step - loss: 0.9340 - accuracy: 0.7335 - val_loss: 0.9936 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 10/100
42/42 [==============================] - 39s 917ms/step - loss: 0.9353 - accuracy: 0.7305 - val_loss: 0.9908 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 11/100
42/42 [==============================] - 39s 926ms/step - loss: 0.9127 - accuracy: 0.7275 - val_loss: 0.9663 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 12/100
42/42 [==============================] - 39s 937ms/step - loss: 0.8942 - accuracy: 0.7335 - val_loss: 0.9702 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 13/100
42/42 [==============================] - 41s 984ms/step - loss: 0.8690 - accuracy: 0.7335 - val_loss: 0.9850 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 14/100
42/42 [==============================] - 41s 978ms/step - loss: 0.8480 - accuracy: 0.7365 - val_loss: 0.9839 - val_accuracy: 0.7262 - lr: 0.0010
```

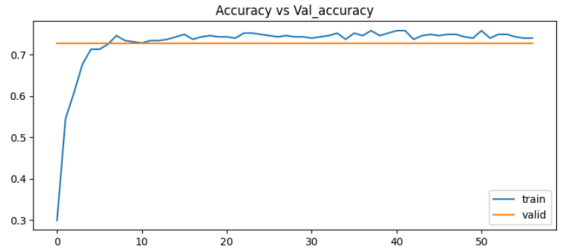
```
Epoch 44/100
42/42 [==============================] - 37s 883ms/step - loss: 0.7574 - accuracy: 0.7455 - val_loss: 0.9926 - val_accuracy: 0.7262 - lr: 3.8147e-09
Epoch 45/100
42/42 [==============================] - 39s 926ms/step - loss: 0.7838 - accuracy: 0.7485 - val_loss: 0.9967 - val_accuracy: 0.7262 - lr: 3.8147e-09
Epoch 46/100
42/42 [==============================] - 39s 922ms/step - loss: 0.7611 - accuracy: 0.7455 - val_loss: 0.9926 - val_accuracy: 0.7262 - lr: 3.8147e-09
Epoch 47/100
42/42 [==============================] - 39s 934ms/step - loss: 0.7491 - accuracy: 0.7485 - val_loss: 0.9910 - val_accuracy: 0.7262 - lr: 4.7684e-10
Epoch 48/100
42/42 [==============================] - 39s 945ms/step - loss: 0.7738 - accuracy: 0.7485 - val_loss: 0.9918 - val_accuracy: 0.7262 - lr: 4.7684e-10
Epoch 49/100
42/42 [==============================] - 40s 953ms/step - loss: 0.7672 - accuracy: 0.7425 - val_loss: 0.9922 - val_accuracy: 0.7262 - lr: 4.7684e-10
Epoch 50/100
42/42 [==============================] - 37s 874ms/step - loss: 0.7552 - accuracy: 0.7395 - val_loss: 0.9902 - val_accuracy: 0.7262 - lr: 4.7684e-10
Epoch 51/100
42/42 [==============================] - 38s 896ms/step - loss: 0.7631 - accuracy: 0.7575 - val_loss: 0.9932 - val_accuracy: 0.7262 - lr: 4.7684e-10
Epoch 52/100
42/42 [==============================] - 39s 932ms/step - loss: 0.7881 - accuracy: 0.7395 - val_loss: 0.9938 - val_accuracy: 0.7262 - lr: 5.9605e-11
Epoch 53/100
42/42 [==============================] - 39s 927ms/step - loss: 0.7723 - accuracy: 0.7485 - val_loss: 0.9940 - val_accuracy: 0.7262 - lr: 5.9605e-11
Epoch 54/100
42/42 [==============================] - 40s 956ms/step - loss: 0.7579 - accuracy: 0.7485 - val_loss: 0.9945 - val_accuracy: 0.7262 - lr: 5.9605e-11
Epoch 55/100
42/42 [==============================] - 40s 951ms/step - loss: 0.7817 - accuracy: 0.7425 - val_loss: 0.9961 - val_accuracy: 0.7262 - lr: 5.9605e-11
Epoch 56/100
42/42 [==============================] - 40s 948ms/step - loss: 0.7345 - accuracy: 0.7395 - val_loss: 0.9948 - val_accuracy: 0.7262 - lr: 5.9605e-11
Epoch 57/100
42/42 [==============================] - 42s 999ms/step - loss: 0.8039 - accuracy: 0.7395 - val_loss: 0.9944 - val_accuracy: 0.7262 - lr: 7.4506e-12
```

- Plotting Loss and accuracy curve vs epochs

```
[ ] #Plotting loss and accuracy curve vs epochs
    plt.figure(figsize=(20,8))
    plt.subplot(2,2,1)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.legend(("train" , "valid") , loc =0)
    plt.title("Loss vs Val_loss")
    plt.subplot(2,2,2)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.legend(("train" , "valid") , loc =0)
    plt.title("Accuracy vs Val_accuracy")
```

```
Text(0.5, 1.0, 'Accuracy vs Val_accuracy')
```



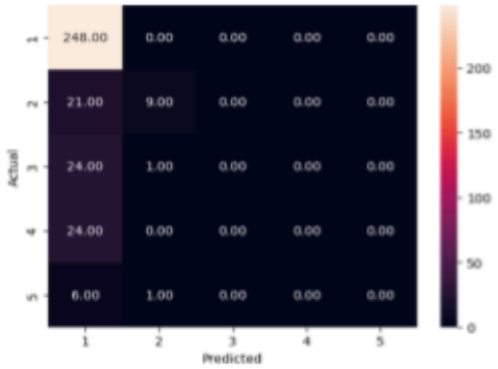


- Creating function to display Evaluation matrix

```
[ ] evaluation_matrix(model_lstm2,[x_train_1,x_train_2],y_train,[x_test_1,x_test_2],y_test)
```

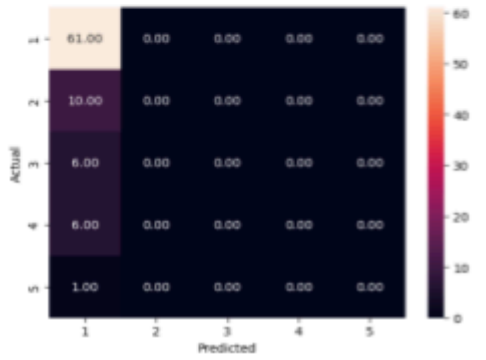
```
11/11 [==============================] - 8s 673ms/step
3/3 [==============================] - 1s 436ms/step
Train Recall: 76.95
Test Recall: 72.62
Train Precision: 64.36
Test Precision: 52.74
Train F1: 68.44
Test F1: 61.1
```

Confusion matrix for training set:




```
**********************************************************************

Classification report for Training set:

              precision    recall  f1-score   support

           1       0.77      1.00      0.87       248
           2       0.82      0.30      0.44        30
           3       0.00      0.00      0.00        25
           4       0.00      0.00      0.00        24
           5       0.00      0.00      0.00         7

    accuracy                           0.77       334
   macro avg       0.32      0.26      0.26       334
weighted avg       0.64      0.77      0.68       334
```

Confusion matrix for testing set:




```
Classification report for Testing set:

              precision    recall  f1-score   support

           1       0.73      1.00      0.84        61
           2       0.00      0.00      0.00        10
           3       0.00      0.00      0.00         6
           4       0.00      0.00      0.00         6
           5       0.00      0.00      0.00         1

    accuracy                           0.73        84
   macro avg       0.15      0.20      0.17        84
weighted avg       0.53      0.73      0.61        84

**********************************************************************
```

- Saving the model

```
[ ] model_lstm2.save('model_lstm2.h5')
```

Building LSTM model 4

- Build a LSTM Neural Network

```
# Build a LSTM Neural Network
input_1 = tf.keras.Input(shape=(maxlen,))
embedding_layer = tf.keras.layers.Embedding(vocab, embedding_size, weights=[embedding_matrix], trainable=False)(input_1)

LSTM_1 = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128, return_sequences = True))(embedding_layer)
maxpool_1 = tf.keras.layers.GlobalAveragePooling1D()(LSTM_1)
dropout_1 = tf.keras.layers.Dropout(0.5, input_shape = (256,))(maxpool_1)
dense_1 = tf.keras.layers.Dense(128, activation = 'relu')(dropout_1)
dropout_2 = tf.keras.layers.Dropout(0.5, input_shape = (128,))(dense_1)
dense_2 = tf.keras.layers.Dense(64, activation = 'relu')(dropout_2)
dropout_3 = tf.keras.layers.Dropout(0.5, input_shape = (64,))(dense_2)

dense_3 = tf.keras.layers.Dense(32, activation = 'relu')(dropout_3)
dropout_4 = tf.keras.layers.Dropout(0.5, input_shape = (32,))(dense_3)

dense_4 = tf.keras.layers.Dense(10, activation = 'relu')(dropout_4)
dropout_5 = tf.keras.layers.Dropout(0.5, input_shape = (32,))(dense_4)

input_2 = tf.keras.Input(shape=(x_train_2.shape[1],))
dense_5 = tf.keras.layers.Dense(20, input_dim=x_train_2.shape[1], activation='relu', kernel_initializer='he_uniform')(input_2)
dropout_6 = tf.keras.layers.Dropout(0.3)(dense_5)
batchnorm_1 = tf.keras.layers.BatchNormalization()(dropout_6)
dense_6 = tf.keras.layers.Dense(15, activation='relu', kernel_initializer='he_uniform')(batchnorm_1)
dropout_7 = tf.keras.layers.Dropout(0.3)(dense_6)
batchnorm_2 = tf.keras.layers.BatchNormalization()(dropout_7)

concat_layer = tf.keras.layers.Concatenate()([dropout_5, batchnorm_2])
dense_7 = tf.keras.layers.Dense(10, activation='relu')(concat_layer)
output = tf.keras.layers.Dense(5, activation='softmax')(dense_7)
model_lstm3 = tf.keras.models.Model(inputs=[input_1, input_2], outputs=output)
```

- Summary of the model

```
model_lstm3.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model_lstm3.summary()
```

```
Model: "model_4"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_9 (InputLayer)        [(None, 562)]             0         []

 embedding_5 (Embedding)     (None, 562, 200)          471200    ['input_9[0][0]']

 bidirectional_4 (Bidirecti  (None, 562, 256)          336896    ['embedding_5[0][0]']
 onal)

 global_average_pooling1d_3  (None, 256)               0         ['bidirectional_4[0][0]']
  (GlobalAveragePooling1D)

 dropout_29 (Dropout)        (None, 256)               0         ['global_average_pooling1d_3[0
                                                                 ][0]']

 dense_32 (Dense)            (None, 128)               32896     ['dropout_29[0][0]']

 dropout_30 (Dropout)        (None, 128)               0         ['dense_32[0][0]']

 input_10 (InputLayer)       [(None, 2548)]            0         []

 dense_33 (Dense)            (None, 64)                8256      ['dropout_30[0][0]']

 dense_36 (Dense)            (None, 20)                46820     ['input_10[0][0]']

 dropout_31 (Dropout)        (None, 64)                0         ['dense_33[0][0]']

 dropout_34 (Dropout)        (None, 20)                0         ['dense_36[0][0]']

 dense_34 (Dense)            (None, 32)                2080      ['dropout_31[0][0]']

 batch_normalization_6 (Bat  (None, 20)                80        ['dropout_34[0][0]']
 chNormalization)

 dropout_32 (Dropout)        (None, 32)                0         ['dense_34[0][0]']

 dense_37 (Dense)            (None, 15)                315       ['batch_normalization_6[0][0]'
                                                                 ]

 dense_35 (Dense)            (None, 10)                330       ['dropout_32[0][0]']

 dropout_35 (Dropout)        (None, 15)                0         ['dense_37[0][0]']

 dropout_33 (Dropout)        (None, 10)                0         ['dense_35[0][0]']

 batch_normalization_7 (Bat  (None, 15)                60        ['dropout_35[0][0]']
 chNormalization)

 concatenate_3 (Concatenate  (None, 25)                0         ['dropout_33[0][0]',
 )                                                               'batch_normalization_7[0][0]'
                                                                 ]

 dense_38 (Dense)            (None, 10)                260       ['concatenate_3[0][0]']

 dense_39 (Dense)            (None, 5)                 55        ['dense_38[0][0]']

==================================================================================================
Total params: 899268 (3.43 MB)
Trainable params: 427978 (1.63 MB)
Non-trainable params: 471270 (1.80 MB)
__________________________________________________________________________________________________
```

- Fit the keras model on the dataset

```
[ ] callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=15, min_delta=1e-6)
    rlrp = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5, min_delta=1e-6)

    # fit the keras model on the dataset
    history = model_lstm3.fit([x_train_1,x_train_2],y_train,batch_size=6,epochs=100,verbose=1,validation_data=([x_test_1,x_test_2],y_test),callbacks=[rlrp,callback])
```

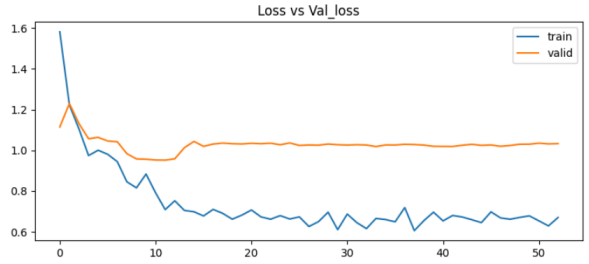
```
Epoch 1/100
56/56 [==============================] - 54s 856ms/step - loss: 1.5821 - accuracy: 0.3824 - val_loss: 1.1147 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 2/100
56/56 [==============================] - 46s 828ms/step - loss: 1.2229 - accuracy: 0.6287 - val_loss: 1.2298 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 3/100
56/56 [==============================] - 47s 824ms/step - loss: 1.1049 - accuracy: 0.6407 - val_loss: 1.1314 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 4/100
56/56 [==============================] - 45s 810ms/step - loss: 0.9741 - accuracy: 0.7066 - val_loss: 1.0569 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 5/100
56/56 [==============================] - 47s 836ms/step - loss: 1.0004 - accuracy: 0.7186 - val_loss: 1.0635 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 6/100
56/56 [==============================] - 46s 822ms/step - loss: 0.9803 - accuracy: 0.7216 - val_loss: 1.0459 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 7/100
56/56 [==============================] - 47s 849ms/step - loss: 0.9443 - accuracy: 0.7305 - val_loss: 1.0420 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 8/100
56/56 [==============================] - 45s 808ms/step - loss: 0.8452 - accuracy: 0.7455 - val_loss: 0.9856 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 9/100
56/56 [==============================] - 48s 864ms/step - loss: 0.8155 - accuracy: 0.7455 - val_loss: 0.9576 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 10/100
56/56 [==============================] - 45s 812ms/step - loss: 0.8833 - accuracy: 0.7395 - val_loss: 0.9562 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 11/100
56/56 [==============================] - 48s 862ms/step - loss: 0.7908 - accuracy: 0.7485 - val_loss: 0.9524 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 12/100
56/56 [==============================] - 45s 808ms/step - loss: 0.7085 - accuracy: 0.7545 - val_loss: 0.9518 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 13/100
56/56 [==============================] - 49s 878ms/step - loss: 0.7521 - accuracy: 0.7425 - val_loss: 0.9583 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 14/100
56/56 [==============================] - 45s 804ms/step - loss: 0.7046 - accuracy: 0.7455 - val_loss: 1.0127 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 15/100
56/56 [==============================] - 49s 873ms/step - loss: 0.6981 - accuracy: 0.7575 - val_loss: 1.0437 - val_accuracy: 0.7262 - lr: 0.0010
Epoch 16/100
56/56 [==============================] - 45s 806ms/step - loss: 0.6775 - accuracy: 0.7605 - val_loss: 1.0195 - val_accuracy: 0.7262 - lr: 0.0010
```

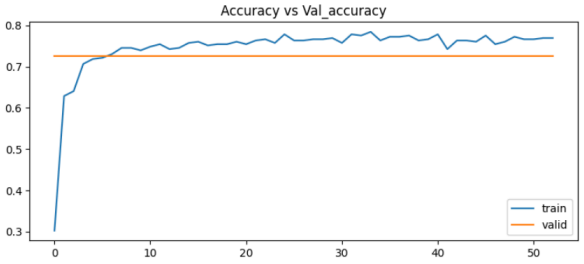
```
Epoch 38/100
56/56 [==============================] - 45s 808ms/step - loss: 0.6854 - accuracy: 0.7754 - val_loss: 1.0282 - val_accuracy: 0.7262 - lr: 1.0000e-08
Epoch 39/100
56/56 [==============================] - 51s 923ms/step - loss: 0.6545 - accuracy: 0.7635 - val_loss: 1.0257 - val_accuracy: 0.7262 - lr: 1.0000e-08
Epoch 40/100
56/56 [==============================] - 45s 806ms/step - loss: 0.6960 - accuracy: 0.7665 - val_loss: 1.0197 - val_accuracy: 0.7262 - lr: 1.0000e-08
Epoch 41/100
56/56 [==============================] - 46s 831ms/step - loss: 0.6532 - accuracy: 0.7784 - val_loss: 1.0191 - val_accuracy: 0.7262 - lr: 1.0000e-08
Epoch 42/100
56/56 [==============================] - 45s 805ms/step - loss: 0.6797 - accuracy: 0.7425 - val_loss: 1.0188 - val_accuracy: 0.7262 - lr: 1.0000e-08
Epoch 43/100
56/56 [==============================] - 48s 859ms/step - loss: 0.6722 - accuracy: 0.7635 - val_loss: 1.0250 - val_accuracy: 0.7262 - lr: 1.0000e-09
Epoch 44/100
56/56 [==============================] - 45s 806ms/step - loss: 0.6590 - accuracy: 0.7635 - val_loss: 1.0292 - val_accuracy: 0.7262 - lr: 1.0000e-09
Epoch 45/100
56/56 [==============================] - 48s 854ms/step - loss: 0.6446 - accuracy: 0.7605 - val_loss: 1.0244 - val_accuracy: 0.7262 - lr: 1.0000e-09
Epoch 46/100
56/56 [==============================] - 45s 811ms/step - loss: 0.6974 - accuracy: 0.7754 - val_loss: 1.0262 - val_accuracy: 0.7262 - lr: 1.0000e-09
Epoch 47/100
56/56 [==============================] - 48s 860ms/step - loss: 0.6677 - accuracy: 0.7545 - val_loss: 1.0198 - val_accuracy: 0.7262 - lr: 1.0000e-09
Epoch 48/100
56/56 [==============================] - 45s 814ms/step - loss: 0.6614 - accuracy: 0.7605 - val_loss: 1.0237 - val_accuracy: 0.7262 - lr: 1.0000e-10
Epoch 49/100
56/56 [==============================] - 48s 854ms/step - loss: 0.6703 - accuracy: 0.7725 - val_loss: 1.0297 - val_accuracy: 0.7262 - lr: 1.0000e-10
Epoch 50/100
56/56 [==============================] - 45s 807ms/step - loss: 0.6779 - accuracy: 0.7665 - val_loss: 1.0301 - val_accuracy: 0.7262 - lr: 1.0000e-10
Epoch 51/100
56/56 [==============================] - 48s 859ms/step - loss: 0.6534 - accuracy: 0.7665 - val_loss: 1.0351 - val_accuracy: 0.7262 - lr: 1.0000e-10
Epoch 52/100
56/56 [==============================] - 48s 853ms/step - loss: 0.6286 - accuracy: 0.7695 - val_loss: 1.0316 - val_accuracy: 0.7262 - lr: 1.0000e-10
Epoch 53/100
56/56 [==============================] - 49s 879ms/step - loss: 0.6699 - accuracy: 0.7695 - val_loss: 1.0329 - val_accuracy: 0.7262 - lr: 1.0000e-11
```

- Plotting Loss and accuracy curve vs epochs

```
#Plotting loss and accuracy curve vs epochs
plt.figure(figsize=(20,8))
plt.subplot(2,2,1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(("train" , "valid") , loc =0)
plt.title("Loss vs Val_loss")
plt.subplot(2,2,2)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(("train" , "valid") , loc =0)
plt.title("Accuracy vs Val_accuracy")
```

```
Text(0.5, 1.0, 'Accuracy vs Val_accuracy')
```



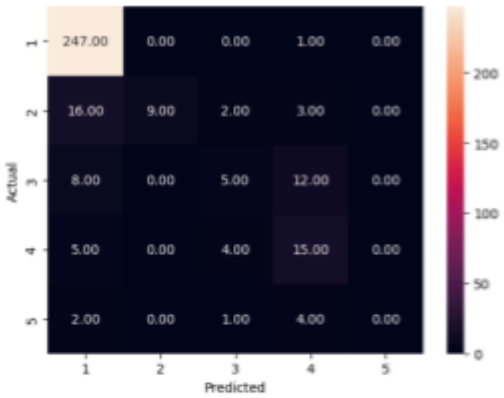


- Creating function to display Evaluation matrix

```
evaluation_matrix(model_lstm3,[x_train_1,x_train_2],y_train,[x_test_1,x_test_2],y_test)
```

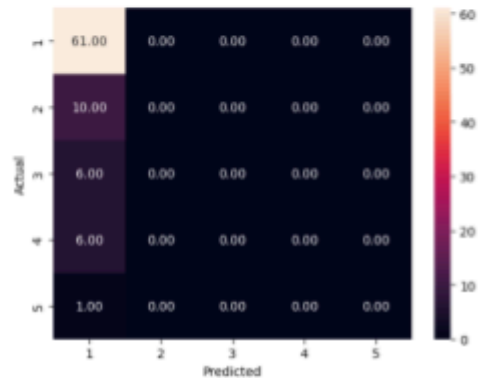
```
11/11 [==============================] - 8s 676ms/step
3/3 [==============================] - 2s 454ms/step
Train Recall: 82.63
Test Recall: 72.62
Train Precision: 81.15
Test Precision: 52.74
Train F1: 79.56
Test F1: 61.1
```

```
Confusion matrix for training set:
```




```
**********************************************************************

Classification report for Training set:

              precision    recall  f1-score   support

           1       0.89      1.00      0.94       248
           2       1.00      0.30      0.46        30
           3       0.42      0.20      0.27        25
           4       0.43      0.62      0.51        24
           5       0.00      0.00      0.00         7

    accuracy                           0.83       334
   macro avg       0.55      0.42      0.44       334
weighted avg       0.81      0.83      0.80       334
```

```
Confusion matrix for testing set:
```




```
assification report for Testing set:

              precision    recall  f1-score   support

           1       0.73      1.00      0.84        61
           2       0.00      0.00      0.00        10
           3       0.00      0.00      0.00         6
           4       0.00      0.00      0.00         6
           5       0.00      0.00      0.00         1

    accuracy                           0.73        84
   macro avg       0.15      0.20      0.17        84
ighted avg         0.53      0.73      0.61        84

**********************************************************************
```

- Saving the model

```
[ ]  model_lstm3.save('model_lstm3.h5')
```

Milestone3

Step1: Design a clickable UI based chatbot interface

Streamlit chatbot

```
[ ]  !pip install pyngrok
     !pip install numpy==1.26.3
     !pip install sickit-learn==1.3.2
     !pip install streamlit==1.29.0
     !pip install streamlit-option-menu==0.3.6
```

```
[ ]  import os
     from threading import Thread
     from pyngrok import ngrok
     import pickle
     import streamlit as st
     from streamlit_option_menu import option_menu
```

```
[ ]  ngrok.set_auth_token('2cCznoLWD9rNBIQxjaoQYVEuaI9_7EVn3a83LcJZtueLeQjvD')
```

In [ ]:

```
%%writefile app.py

import streamlit as st
import pickle
from sklearn.feature_extraction.text import TfidfVectorizer
import tensorflow
import numpy as np

#!pip install unidecode
import unidecode

#!pip install wordcloud
from wordcloud import WordCloud

import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords

nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

#NLP preprocessing functions
def rmv_uni(sen):
  wrds = sen.split()
  new_sen = [unidecode.unidecode(w) for w in wrds]
  new_sen = ' '.join(new_sen)

  return new_sen

def rmv_schar(x):
  wrds = x.split()
  new_text = [w for w in wrds if w.isalnum()]
  new_text = ' '.join(new_text)

  return new_text

def lower_case(x):
  x = x.lower()
  return x

def rmv_stpwrds(sen):
  wrds = sen.split()
  new_text = [w for w in wrds if w not in stopwords.words('english')]
  new_text = ' '.join(new_text)

  return new_text

lemmatizer = WordNetLemmatizer()

def stem(sen):
  wrds = sen.split()
  new_text = [lemmatizer.lemmatize(w) for w in wrds]
  new_text = ' '.join(new_text)

  return new_text

def strp(sen):
  sen = sen.strip()
  return sen


#title
st.set_page_config(page_title="Industrial Safety NLP Based Chatbot",layout="centered",page_icon="👷")

st.title("Industrial Safety NLP Based chatbot")



# loading the trained model
token = "/content/drive/My Drive/tokeniser1.pickle"
vec = "/content/drive/My Drive/vec1.pickle"
loaded_model = pickle.load(open("/content/drive/My Drive/model_lstm3.pickle",'rb'))
loaded_token = pickle.load(open(token,'rb'))
loaded_vec = pickle.load(open(vec,'rb'))


txt = st.text_area(
  "Report the Incident",
)

pred = ''
if st.button('Check Potential Accident Level'):

  txt = rmv_uni(txt)
  txt = rmv_schar(txt)
  txt = lower_case(txt)
  txt = rmv_stpwrds(txt)
  txt = stem(txt)
  txt = strp(txt)

  txt = np.array([txt],dtype='object')
  x1 = loaded_token.texts_to_sequences(txt)
  x1 = pad_sequences(x1,padding='post',maxlen=562)

  x2 = loaded_vec.transform(txt).toarray()

  prediction = loaded_model.predict([x1,x2])
  prediction = np.argmax(pred,axis=0)+1
  pred = prediction

st.success(pred)
  #st.write(txt)
```

```
Overwriting app.py
```

```
Overwriting app.py
```

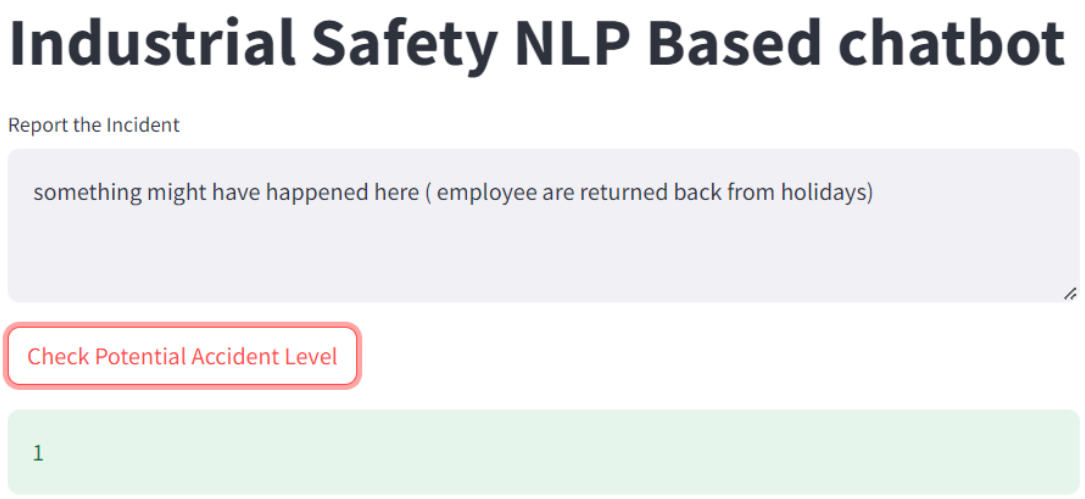
```
[ ] def run_streamlit():
      os.system("streamlit run /content/app.py --server.port 8501")
```

```
[ ] thread = Thread(target=run_streamlit)
    thread.start()
```

```
[ ] public_url = ngrok.connect(addr='8501', proto='http', bind_tls=True)
    print("Your streamlit app is live at :", public_url)
```

```
Your streamlit app is live at : NgrokTunnel: "https://d469-35-188-224-23.ngrok-free.app" -> "http://localhost:8501"
```

```
[ ] # ngrok.kill()
```



Conclusion of Milestone2

- Out of all the model bidirectional model with two input performs better.
- The model predict the accident level with a test f1-score of 61.10%.
- Also deep learning models perform better than machine learning model.
- Finally bidirectional LSTM model with two inputs can be considered to productionalize the model and predict the accident level.
- Finally created an web application chatbot using streamlit.

Limitations

- We have less number of observations to analyse the cause of accidents correctly and rather we should collect more number of observations to get better results.
- Lack of access to quality data.