INDUSTRIAL SAFETY-CHATBOT UTILITY

NLP based Industrial safety Chatbot

Submitted by

Chidanand Pujar

Mrinal Kalita

Thirumurugan S

Date:26/01/2024

Mentor: Jyant Mahara

Table of Contents

Problem Statement 3
Business value proposition: 3
Modelling Objective 3
Dataset 3
Source of dataset 3
Dataset Information 3
Columns Description: 3
What is a chatbot? 4
How to do chatbots work? 4
Why were chatbots created? 5
Evolution of chatbots 5
Milestone1 7
Step1: Import the data 7
Data Collection Summary: 8
Step2: Data Cleansing 9
Data Cleansing Summary: 10
Exploratory Data Analysis (EDA): 11
EDA Summary: 14
Step3: Data Preprocessing 15
Step 4: Data preparation - Cleansed data in .xlsx or .csv file 17
Step 5: Model Building using basic Machine learning classifiers 18
Naive Bayes without Sampling 19
KNearest Neighbour without Sampling 19
Logistic Regression without Sampling 20
SVM without Sampling 20
Decision Tree without Sampling 21
Random Forest without Sampling 21
Ada Boost without Sampling 22
Gradient Boost without Sampling 22
Naive Bayes without Sampling-Tuned 24
KNearest Neighbour without Sampling-Tuned 24
Logistic Regression without Sampling-Tuned 25
SVM without Sampling-Tuned 25

Decision Tree without Sampling-Tuned 26
Random Forest without Sampling-Tuned 26
Ada Boost without Sampling-Tuned 27
Gradient Boost without Sampling-Tuned 27
Summary of Tuning and comparing the Models 28
Model training after sampling 29
Naive Bayes with Sampling 29
KNearest Neighbour with Sampling 29
Logistic Regression with Sampling 30
SVM with Sampling 30
Decision Tree with Sampling 31
Random Forest with Sampling 31
Ada Boost with Sampling 32
Gradient Boost with Sampling 32
Naive Bayes with Sampling-Tuned 35
KNearest Neighbour with Sampling-Tuned 35
Logistic Regression with Sampling-Tuned 36
SVM with Sampling-Tuned 36
Decision Tree with Sampling-Tuned 37
Random Forest with Sampling-Tuned 37
Ada Boost with Sampling-Tuned 38
Gradient Boost with Sampling-Tuned 38
Conclusion of Milestone 1 39

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

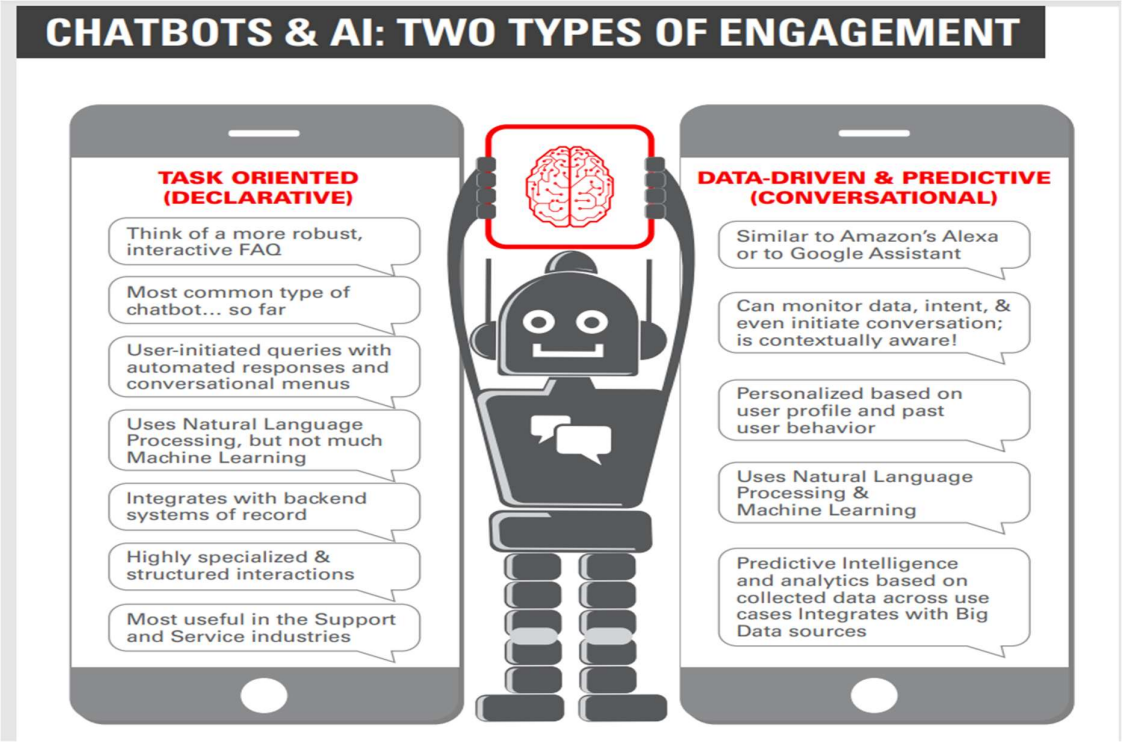
What is a chatbot?

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.

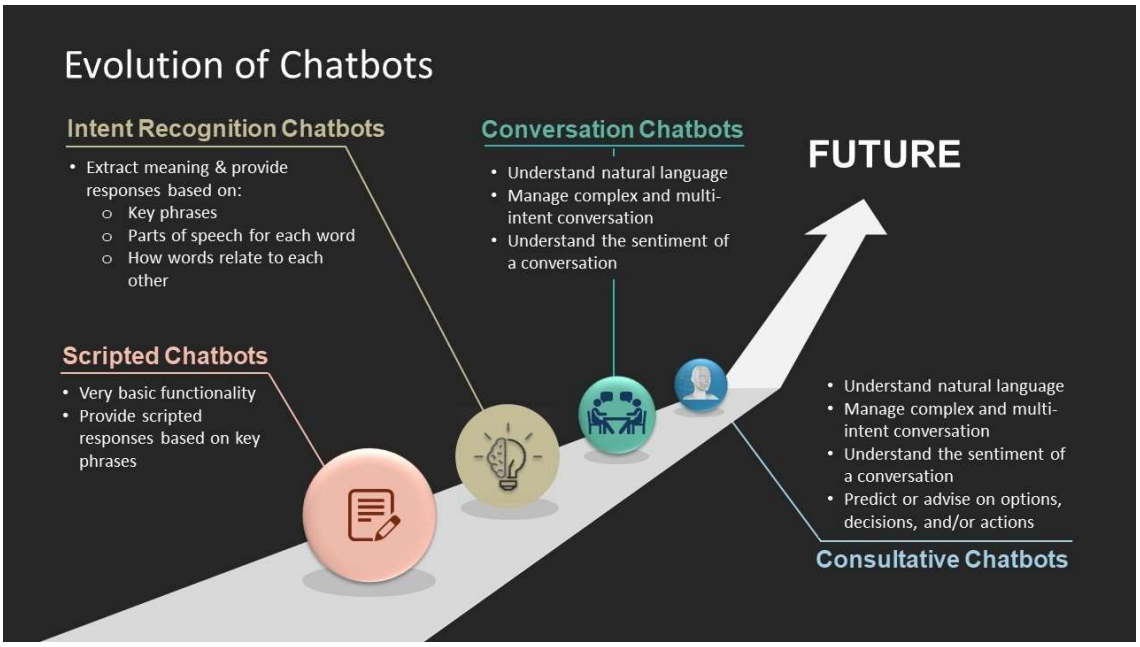
Chatbots allow businesses to connect with customers in a personal way without the expense of human representatives. For example, many of the questions or issues customers have been common and easily answered. That's why companies create FAQs and troubleshooting guides. Chatbots provide a personal alternative 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

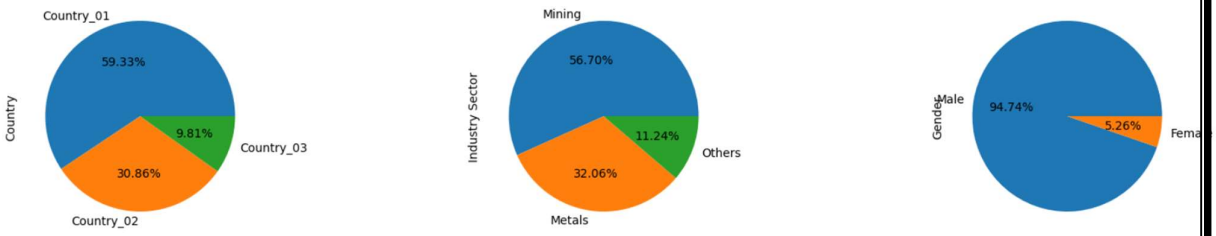
Exploratory Data Analysis (EDA):

- 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'>
```



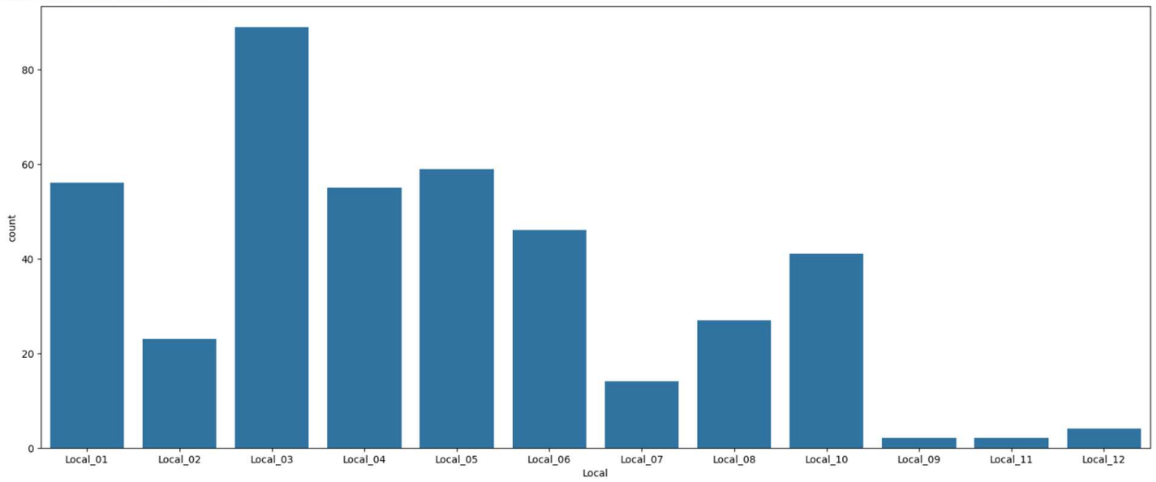
Inference from the univariate analysis:

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')
```

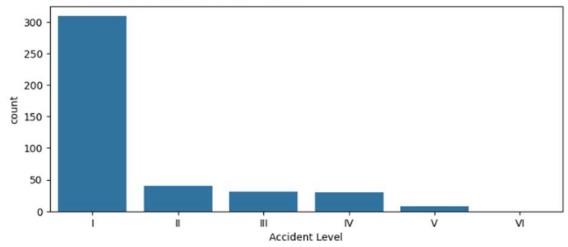
```
<Axes: xlabel='Local', ylabel='count'>
```

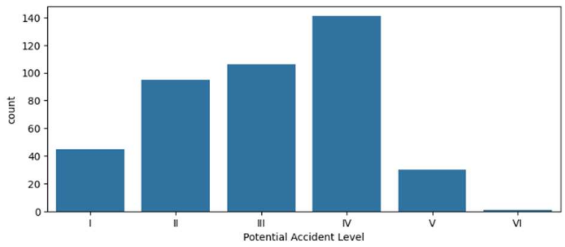


- Checking the distribution of target variable.

```
#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

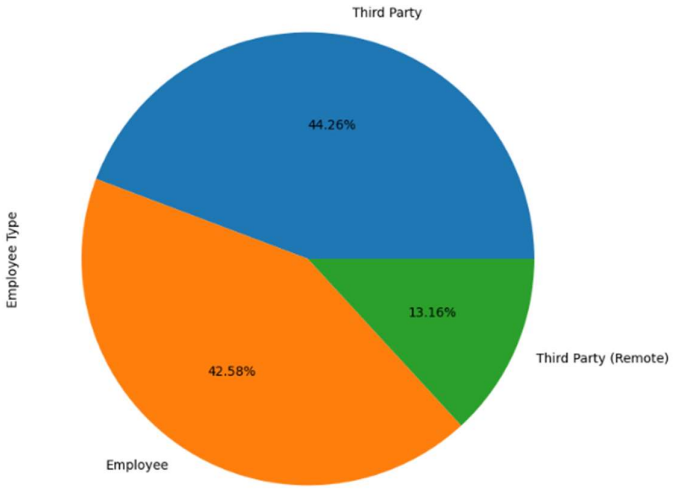
```
#Replacing Accident Level VI with V
df['Accident Level'] = df['Accident Level'].replace('VI','V')
df['Potential Accident Level'] = df['Potential Accident Level'].replace('VI','V')
```

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'>
```

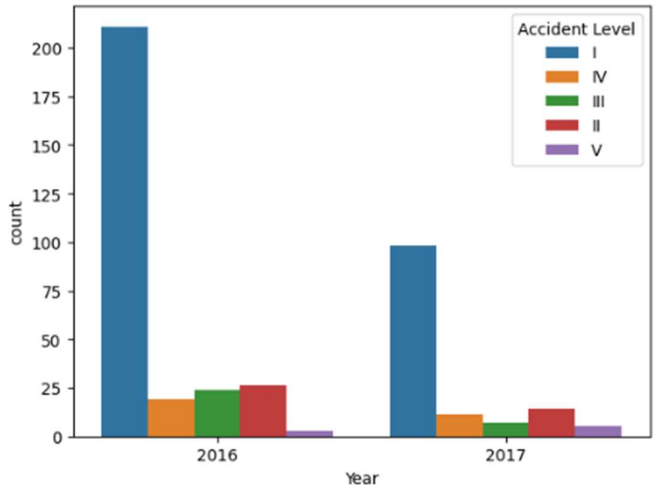


a. 44.26% accidents happen in Third party.
b. 42.585 accidents happen in Employee.
c. 13.16% accidents happen in Third Party (Remote).

- 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'>
```

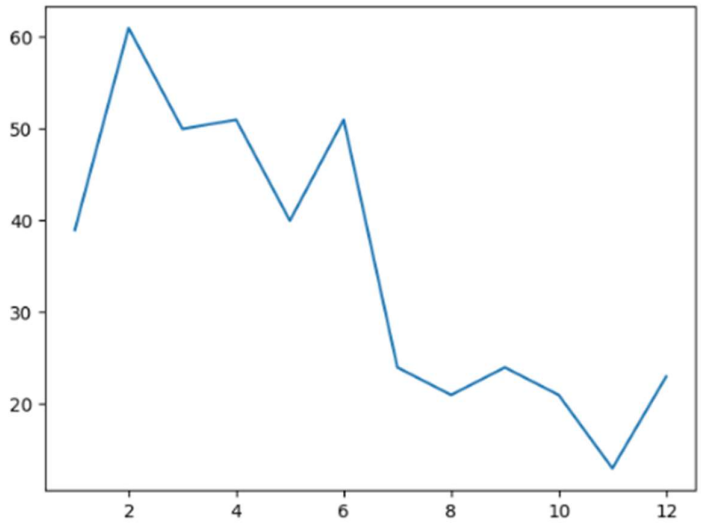


a. Year 2016 had more accident than 2017

- 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>]
```

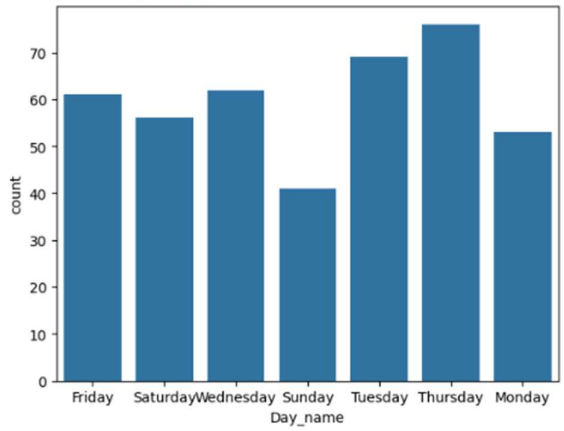


a. Highest accident happened in February.
b. November had lowest no of accident.

- 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()})
```

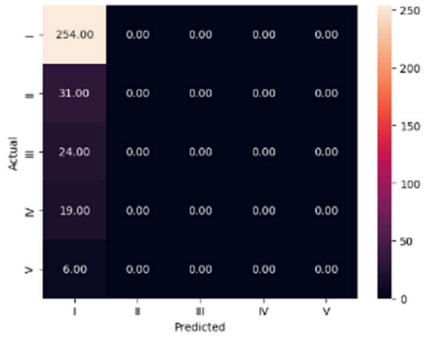
- Training all models

Naive Bayes without Sampling

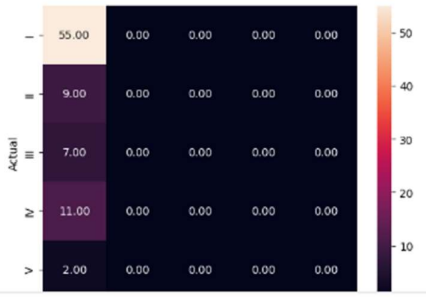
Model Name: Naive Bayes without Sampling

Training Accuracy: 0.7604790419161677
Testing Accuracy: 0.654761904761904

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:

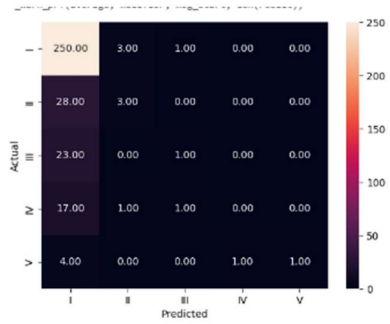
	precision	recall	f1-score	support
I	0.65	1.00	0.79	55
II	0.00	0.00	0.00	9
III	0.00	0.00	0.00	7
IV	0.00	0.00	0.00	11
V	0.00	0.00	0.00	2
accuracy			0.65	84
macro avg	0.13	0.20	0.16	84
weighted avg	0.43	0.65	0.52	84

****

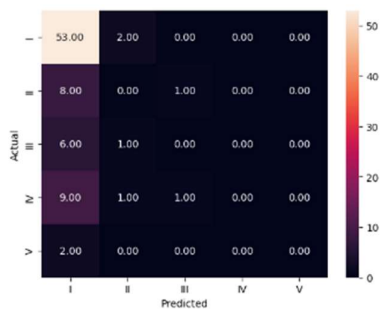
KNearest Neighbour without Sampling

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

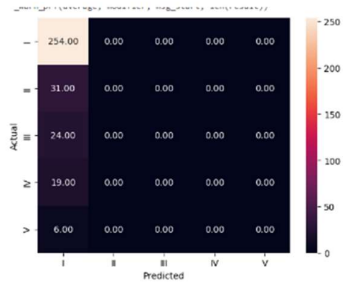
Classification report for Testing set:

	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

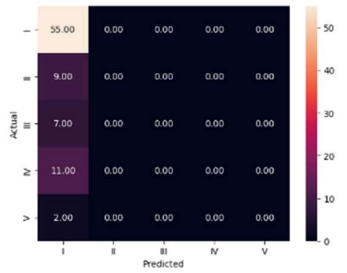
```
**********************************************************************

Model Name: Logistic Regression without Sampling

Training Accuracy:  0.7604790419161677
Testing Accuracy:  0.6547619047619048

Confusion matrix for training set:
```




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

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:
```

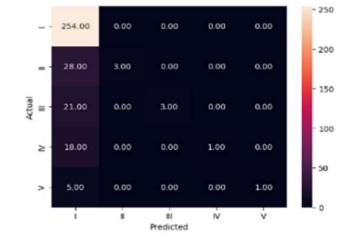
	precision	recall	f1-score	support
I	0.65	1.00	0.79	55
II	0.00	0.00	0.00	9
III	0.00	0.00	0.00	7
IV	0.00	0.00	0.00	11
V	0.00	0.00	0.00	2
accuracy			0.65	84
macro avg	0.13	0.20	0.16	84
weighted avg	0.43	0.65	0.52	84

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

SVM without Sampling

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

Model Name: SVM without Sampling

Training Accuracy:  0.7844311377245509
Testing Accuracy:  0.6547619047619048

Confusion matrix for training set:
```

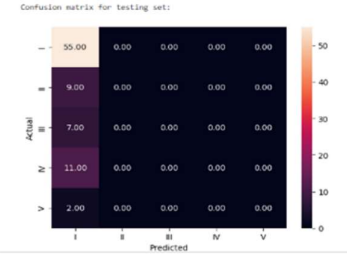
```
Confusion matrix for training set:
```




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

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

```
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

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

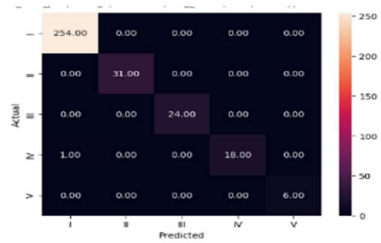
Decision Tree without Sampling

****

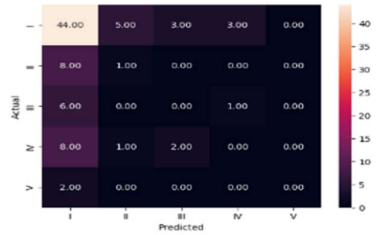
Model Name: Decision Tree without Sampling

Training Accuracy: 0.9970059880239521
Testing Accuracy: 0.5357142857142857

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	0.65	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

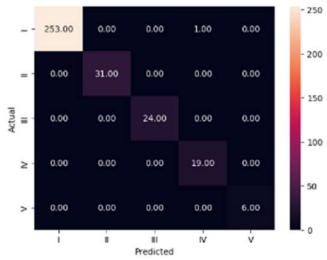
****

Random Forest without Sampling

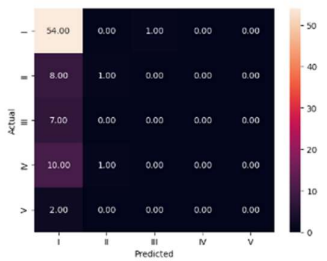
Model Name: Random Forest without Sampling

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

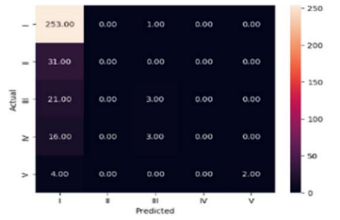
****

Ada Boost without Sampling

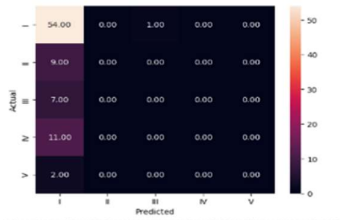
Model Name: Ada Boost without Sampling

Training Accuracy: 0.7724550898203593
Testing Accuracy: 0.6428571428571429

Confusion matrix for training set:



Confusion matrix for testing set:



Predicted

****

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

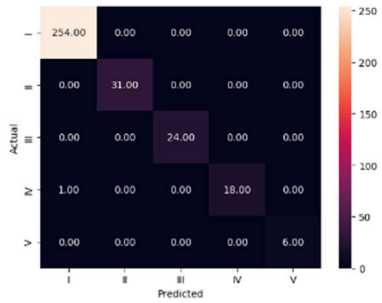
Gradient Boost without Sampling

****

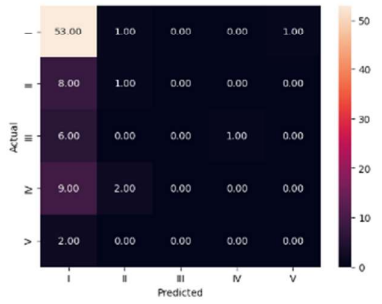
Model Name: Gradient Boost without Sampling

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	42.80	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

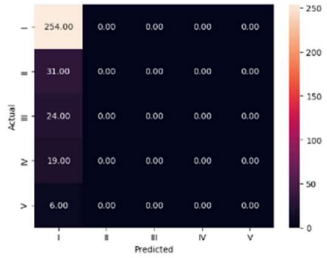
```
params_gb = {'n_estimators': [100, 150,200],'max_depth':np.arange(3,7,1),
             'max_features':np.arange(0.2,0.6,0.1)
             }
gb_param = hyperparameter_tunning(models['Gradient Boost without Sampling'],x_train,y_train,params_gb)
```

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

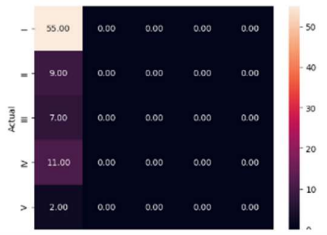
- Training all tuned models

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:
```



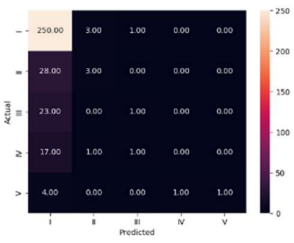

```
Confusion matrix for testing set:
```



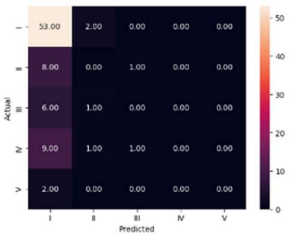

```
**********************************************************************

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:
```

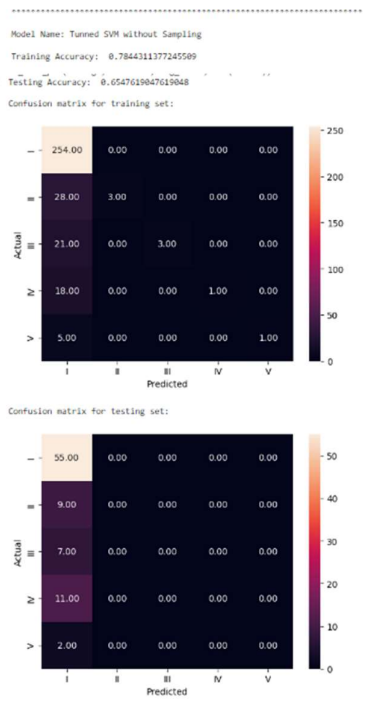



```
Classification report for Training set:

              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



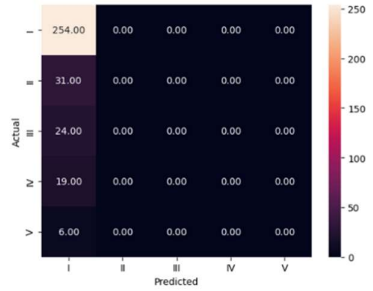

```
Classification report for Training set:
              precision    recall  f1-score   support

           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

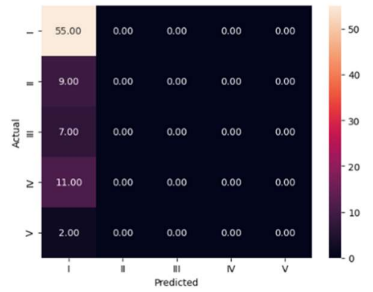
Confusion matrix for training set:



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:

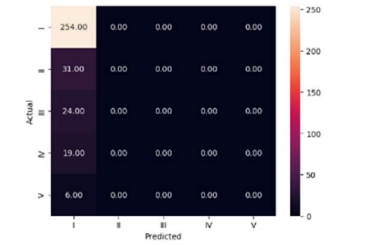
	precision	recall	f1-score	support
I	0.65	1.00	0.79	55
II	0.00	0.00	0.00	9
III	0.00	0.00	0.00	7
IV	0.00	0.00	0.00	11
V	0.00	0.00	0.00	2
accuracy			0.65	84
macro avg	0.13	0.20	0.16	84
weighted avg	0.43	0.65	0.52	84

Random Forest without Sampling-Tuned

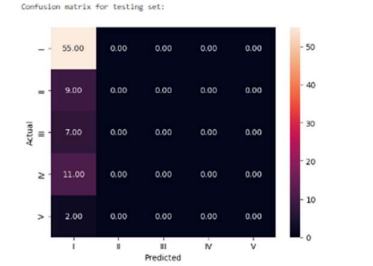
Model Name: Tunned Random Forest without Sampling

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

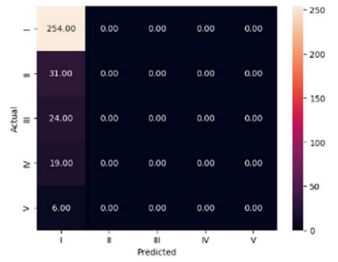
****

Ada Boost without Sampling-Tuned

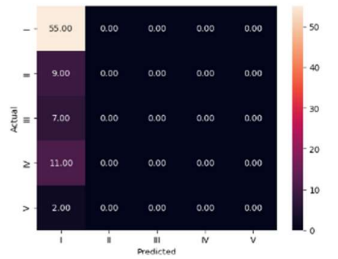
Model Name: Tunned Ada Boost without Sampling

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

****

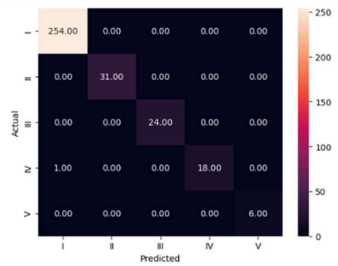
Gradient Boost without Sampling-Tuned

****

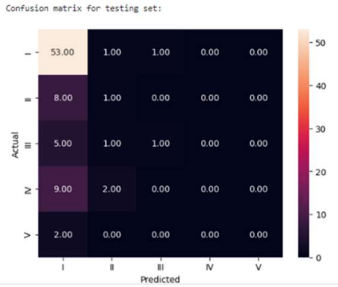
Model Name: Tunned Gradient Boost without Sampling

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

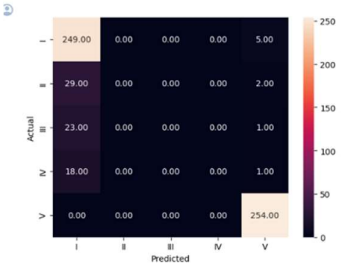
- Defining the models with sampling

```
#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()})
```

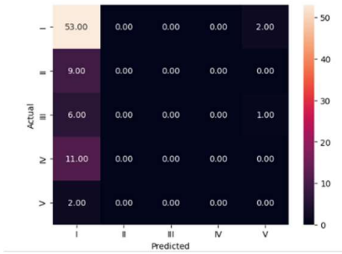
```
#Training all the moddels
train_test(models_sampling,x_train_sample,y_train_sample,x_test,y_test)
```

Naive Bayes with Sampling

```
Model Name: Naive Bayes with Sampling

Training Accuracy:  0.8642611683848798
Testing Accuracy:  0.6309523809523809

Confusion matrix for training set:
```



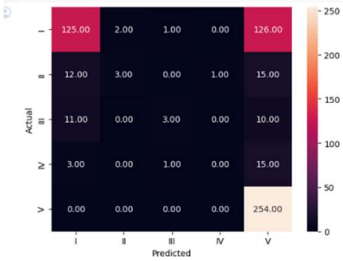

```
Confusion matrix for testing set:
```



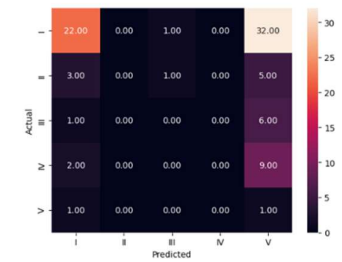

```
**********Predicted**********

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:
```



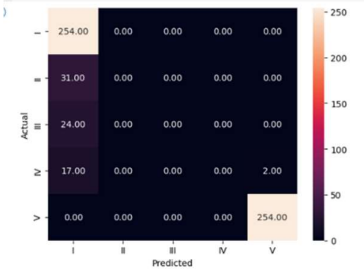

```
**********

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

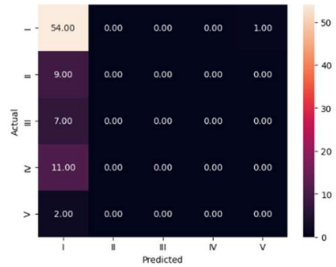
Model Name: Logistic Regression with Sampling

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

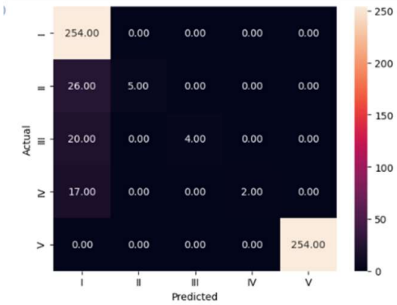
****

SVM with Sampling

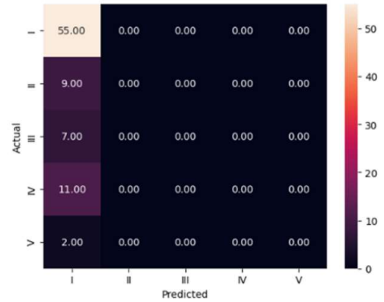
Model Name: SVM with Sampling

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

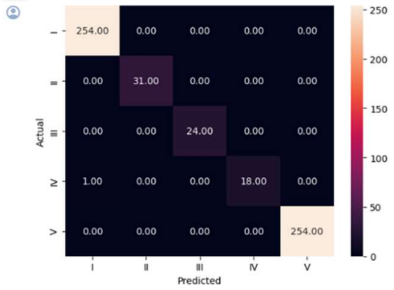
****

Decision Tree with Sampling

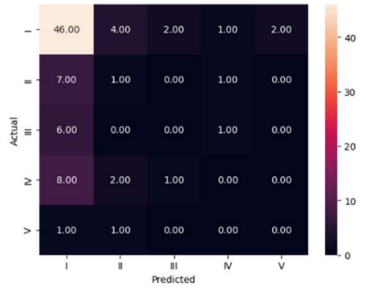
Model Name: Decision Tree with Sampling

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

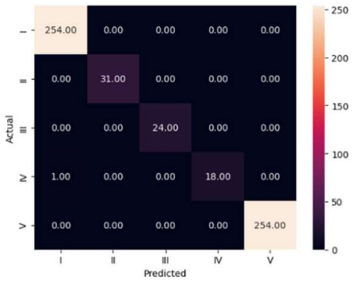
****

Random Forest with Sampling

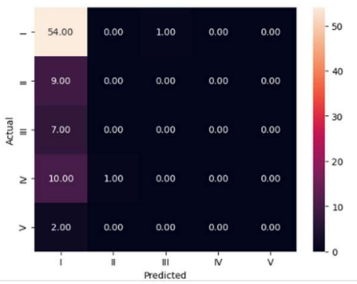
Model Name: Random Forest with Sampling

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

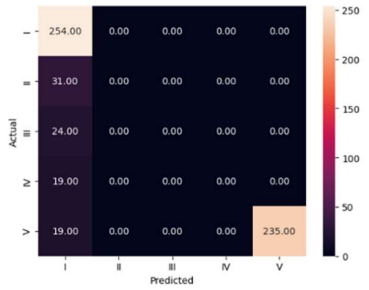
****

Ada Boost with Sampling

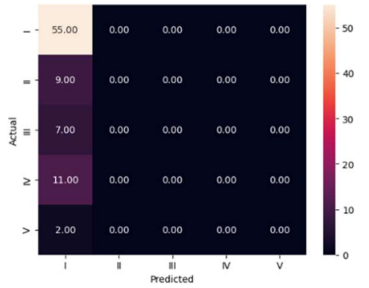
Model Name: Ada Boost with Sampling

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

****

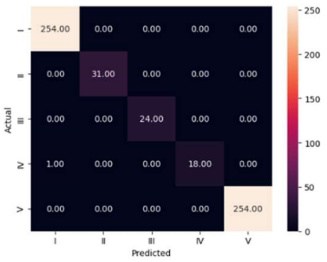
Gradient Boost with Sampling

****

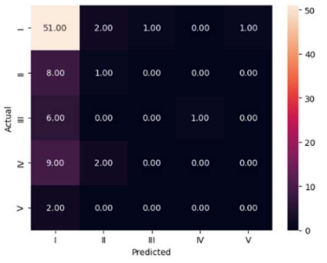
Model Name: Gradient Boost with Sampling

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 %)
```

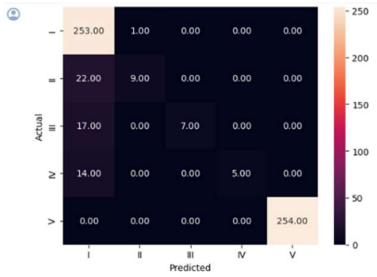
- Training models with sampling

```
[ ] #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)})
```

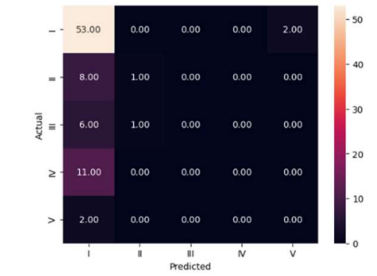
```
    #Training the model
    train_test(models_sampling_tunned,x_train_sample,y_train_sample,x_test,y_test)
```

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:
```



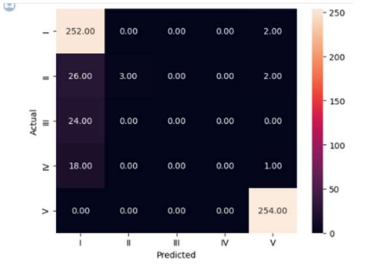

```
Confusion matrix for testing set:
```



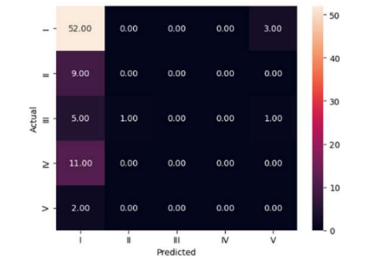

```
**********************************************************************

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:
```



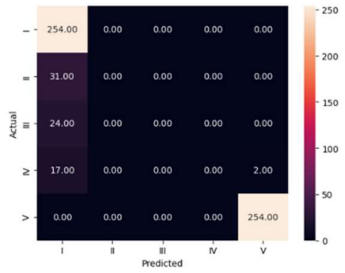

```
Confusion matrix for testing set:
```



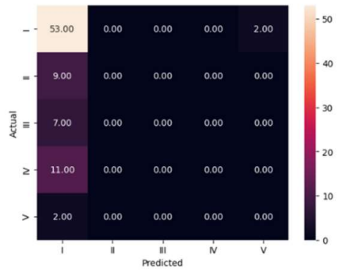

```
**********************************************************************

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:
```



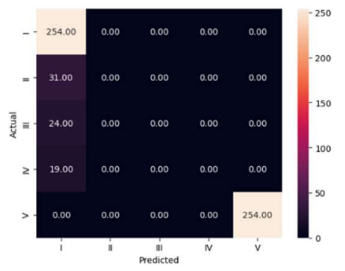

```
Confusion matrix for testing set:
```



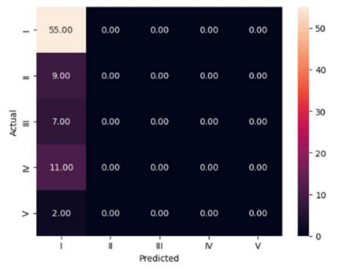

```
**********************************************************************

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:
```



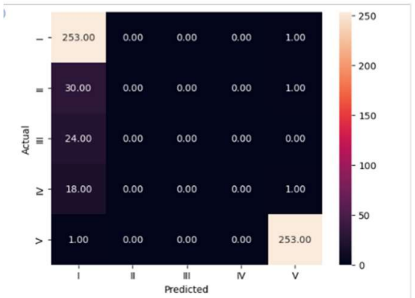

```
Confusion matrix for testing set:
```



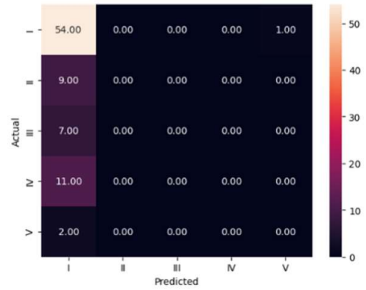

```
**********************************************************************

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:
```



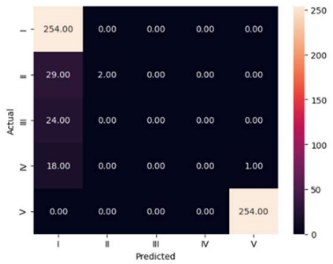

```
Confusion matrix for testing set:
```



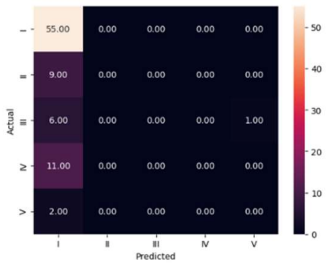

```
**********************************************************************

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

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

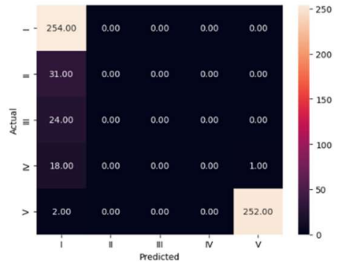
Ada Boost with Sampling-Tuned

****

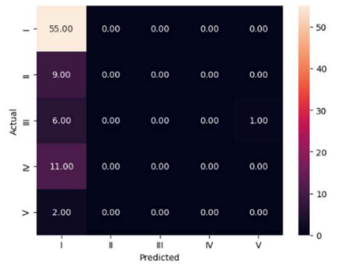
Model Name: Tunned Ada Boost with Sampling

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

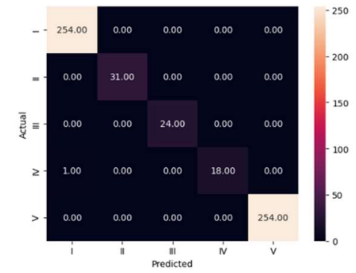
****

Gradient Boost with Sampling-Tuned

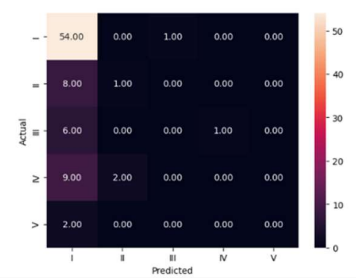
Model Name: Tunned Gradient Boost with Sampling

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.