Application of Natural Language Processing principles to prevent Industrial Accidents

Final Report

July 2022

Table of Contents

Team Details

#	Member	Email
1	Ajit Upadhyaya	ajit.u1987@gmail.com
2	Nandish Mahalingappa	nandishm1@outlook.com / nandi.89@gmail.com
3	Sainath Reddy	gummisainath@gmail.com
4	Prijesh M	prijesh1987@gmail.com
5	Thendral M	thendral.vasugi@gmail.com

Mentor Details

#	Name	Email
1	Naga Pavan Kumar	nagapavan.kalepu@gmail.com

Appreciation to our Mentor

Thanks for being a great mentor and providing your thorough guidance to help us complete this project . We sincerely appreciate and treasure everything you have taught us. You deserve a big thank you from all of us.

Abstract

According to International Labour Organization (ILO) statistics, about 2.3 million men and women worldwide die from work-related illnesses or accidents each year. Across the world, there are estimated 340 million occupational accidents. In a public open source platform, a Brazilian company named ‘IHM Stefanini’ has published a dataset on industrial accidents. To capture these complex problems surrounding industrial accidents, application of natural language processing methods demonstrate promising improvements to empower proactive industrial safety and assist early diagnosis. We provide an in-depth technical analysis on the dataset by applying traditional machine learning methods to deep learning methods. We also provided a technical chatbot which generates automated accident levels based on cause of the injury. This will enable the industrial engineers and healthcare professionals to design a safety protocol which might minimize the occupational accidents.

Introduction

Industrial accidents highlighted in the dataset focuses more on occupations related to mining and metal extraction industries. According to the International Standard Organization, the accidents can be caused by a number of different things, such as leaks of hazardous gasses, dust explosions, collapsing mine stopes, toxic gasses produced by mine fires, mining-induced seismicity, flooding, or simple mechanical mistakes from improperly used or malfunctioning mining equipment. In this dataset, the information about accidents in 12 manufacturing plants in 3 countries are given by a Brazilian company, IHM Stefanini. In reality, the description in the dataset highlights the workers injured or died from the above type of accidents.

A subset of artificial intelligence (AI) technologies called natural language processing (NLP) has become increasingly important in recent years for supporting the analysis and management of large amounts of textual data as well as for facilitating tasks like information extraction, sentiment analysis and emotion detection. We can use NLP approaches to automatically discover early indicators of accident cause to enable early detection, prevention, and treatment. Detecting accident level from text might be framed as a text classification or sentiment analysis problem.

This report focuses on specific industrial injuries and introduces computational methods for accident prediction. To the best of our knowledge, there hasn't been any new studies into the application of NLP algorithms to textual sources for accident detection. We demonstrate an expanded application of NLP for accident detection using Brazilian data. A technical overview of the NLP techniques utilized for text-based accident level detection is what this report seeks to achieve. The following questions are the focus of our report:

Questions

1. What are the major NLP techniques and strategies for detecting accident data ?
2. What valuable insights can be extracted using Exploratory data analysis from the data ?
3. Is the application of deep learning models in text classification better than traditional classification algorithms ?

Methodology and Results

Milestone 1

Exploratory Data Analysis

1. Data Collection

a. Load datasets

The dataset files provided on the kaggle link are:

i.IHMStefanini_industrial_safety_and_health_database.csv

ii.IHMStefanini_industrial_safety_and_health_database_with_accidents_description.csv

We can read the these files by using pandas as below:

```
dataset1 = pd.read_csv('IHMStefanini_industrial_safety_and_health_database.csv')
dataset2 = pd.read_csv('IHMStefanini_industrial_safety_and_health_database_with_accidents_description.csv')
```

b. Analyze the datasets

Let's look at the first 5 records from each datasets,

```
dataset1.head()
```

	Data	Countries	Local	Industry Sector	Accident Level	Potential Accident Level	Genre	Employee ou Terceiro	Risco Critico
0	2016-01-01 00:00:00	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed
1	2016-01-02 00:00:00	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems
2	2016-01-06 00:00:00	Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools
3	2016-01-08 00:00:00	Country_01	Local_04	Mining	I	I	Male	Third Party	Others
4	2016-01-10 00:00:00	Country_01	Local_04	Mining	IV	IV	Male	Third Party	Others

We can see the dataset1 is having the fields - Data, Countries, Local, Industry Sector, Accident Level, Potential Accident Level, Genre, Employee ou Terceiro, Risco Critico

```
dataset2.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 00:00:00	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...
1	1	2016-01-02 00:00:00	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...
2	2	2016-01-06 00:00:00	Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...
3	3	2016-01-08 00:00:00	Country_01	Local_04	Mining	I	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 C...
4	4	2016-01-10 00:00:00	Country_01	Local_04	Mining	IV	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances t...

We can see the dataset2 has the fields - Date, Country, Local, Industry Sector, Accident Level, Potential Accident Level, Gender, Employee type, Critical Risk, Description.

If we are comparing the values from each dataset, we can conclude the field matches as below:

#	dataset1	dataset2	Description
1		Unnamed: 0	Field with index value which is available only on dataset2. Can be removed
2	Data	Date	Field name difference
3	Countries	Country	Field name difference
4	Local	Local	Field name matching
5	Industry Sector	Industry Sector	Field name matching
6	Accident Level	Accident Level	Field name matching
7	Potential Accident Level	Potential Accident Level	Field name matching
8	Genre	Gender	Field name difference
9	Employee ou Terceiro	Employee type	Field name difference
10	Risco Critico	Critical Risk	Field name difference
11		Description	New field which is available only on dataset2

Here we can conclude that except the **Description** field, all other fields are the same on each dataset even though it has different names.

c. Verify shapes

Now let's analyze the shape of each datasets

```
dataset1.shape, dataset2.shape
```

```
((439, 9), (425, 11))
```

We can conclude that dataset1 is having 439 records and 9 columns and dataset2 is having 425 records and 11 columns.

d. Dataset finalization

Since it is an NLP problem and the Description field is a mandatory item, we can proceed further with **dataset2**. So we are finalizing **dataset2** for further processing by assigning it to a variable called **df**.

```
df = dataset2
```

e. Verify data types

Let's look at the data types of each observation.

```
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    object
 2   Countries                 425 non-null    object
 3   Local                     425 non-null    object
 4   Industry Sector           425 non-null    object
 5   Accident Level            425 non-null    object
 6   Potential Accident Level  425 non-null    object
 7   Genre                     425 non-null    object
 8   Employee or Third Party   425 non-null    object
 9   Critical Risk             425 non-null    object
 10  Description               425 non-null    object
dtypes: int64(1), object(10)
memory usage: 36.6+ KB
```

We have an index column that can be removed. Rest of the object columns can be changed to numeric or ML models.

2. Data Cleansing

a. Remove irrelevant columns

As we already mentioned, we can remove the **Unnamed: 0** column as it contains only the index values.

```
df.drop("Unnamed: 0", axis=1, inplace=True)
```

b. Rename the columns

Let's rename the columns with meaningful names.

```
df.rename(columns={'Data':'Date', 'Countries':'Country', 'Genre':'Gender', 'Employee or Third Party':'Employee type'}, inplace=True)
df.head(3)
```

	Date	Country	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Description
0	2016-01-01 00:00:00	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...
1	2016-01-02 00:00:00	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...
2	2016-01-06 00:00:00	Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...

c. Verify duplicate data

Let's verify the duplicate rows on the dataset

```
df.duplicated().sum()
```

```
7
```

```
duplicates = df.duplicated()

df[duplicates]
```

	Date	Country	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Description
77	2016-04-01 00:00:00	Country_01	Local_01	Mining	I	V	Male	Third Party (Remote)	Others	In circumstances that two workers of the Abrat...
262	2016-12-01 00:00:00	Country_01	Local_03	Mining	I	IV	Male	Employee	Others	During the activity of chuteo of ore in hopper...
303	2017-01-21 00:00:00	Country_02	Local_02	Mining	I	I	Male	Third Party (Remote)	Others	Employees engaged in the removal of material f...
345	2017-03-02 00:00:00	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 00:00:00	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 00:00:00	Country_03	Local_10	Others	I	I	Male	Third Party	Venomous Animals	Team of the VMS Project performed soil collect...
397	2017-05-23 00:00:00	Country_01	Local_04	Mining	I	IV	Male	Third Party	Projection of fragments	In moments when the 02 collaborators carried o...

Here we can see the dataset has 7 duplicate records.

d. Drop duplicate data

Let's drop the duplicate rows on the dataset.

```
df.drop_duplicates(inplace=True)
```

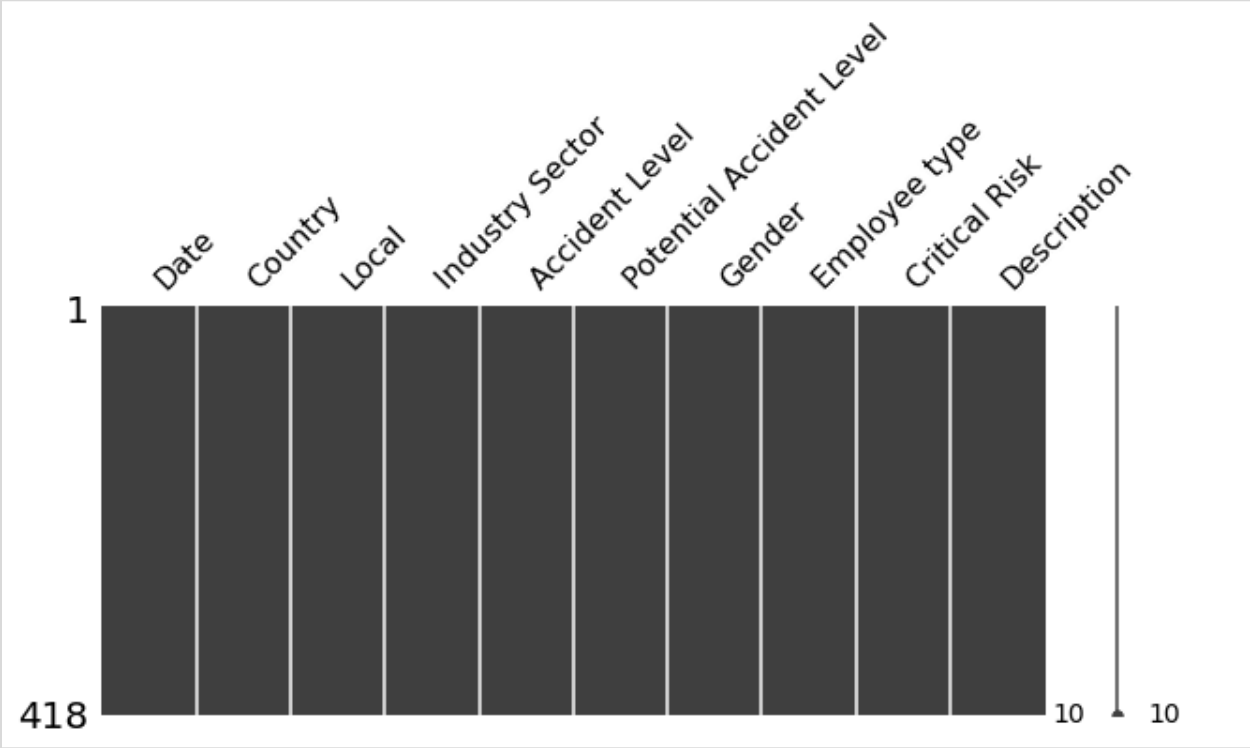
```
df.shape
```

```
(418, 10)
```

After dropping the duplicate values, now we have 418 records on the dataset.

e. Analyze missing values

Let's analyze the missing values available on the dataset.



Here we can see the current dataset doesn't have any missing values.

3. Data Preprocessing

For better understanding of the dataset, we can extract few values from the existing dataset fields as below:

1. **Date** - As all the Date values doesn't have proper time values, we can extract only the date part from the field
2. **Year** - We can extract it from the existing **Date** field
3. **Month** - We can extract it from the existing **Date** field
4. **Day** - We can extract it from the existing **Date** field
5. **Weekday** - We can extract it from the existing **Date** field
6. **WeekOfYear** - We can extract it from the existing **Date** field
7. **Season** - We can extract it from the existing **Date** field's month value
8. **IsHoliday** - We can extract this boolean value from the **Date** filed with the help of **holidays** package

Let's visualize the dataset after adding these additional fields:

	Date	Country	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Description	Year	Month	Day	Weekday	WeekofYear	Season	Is_Holiday
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	53	Summer	1
1	2016-01-02	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...	2016	1	2	Saturday	53	Summer	0
2	2016-01-06	Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...	2016	1	6	Wednesday	1	Summer	0

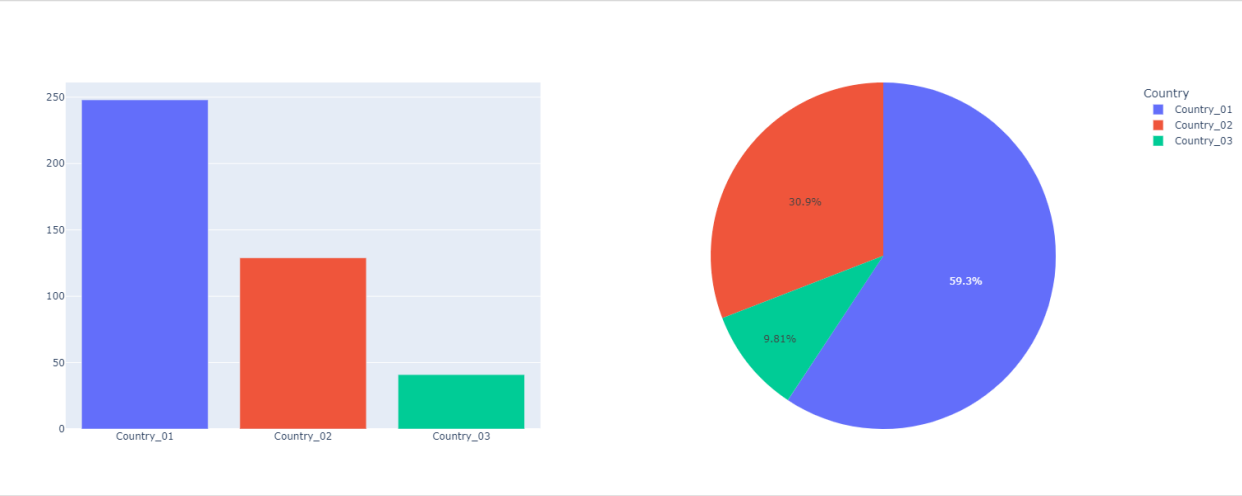
4. Univariate Analysis

In this section we will analyze some of the observations. This will give us better insight on the data. For this we have created a reusable function which will take the column name and dataset as the parameters and generate the visualizations dynamically.

```
def univariate_analysis(col, df):
  fig = make_subplots(rows=1, cols=2, specs=[[{"type": "xy"}, {"type": "domain"}]])

  labels = df[col].value_counts().index
  values = df[col].value_counts().values
  colors = px.colors.qualitative.Plotly + px.colors.qualitative.D3 + px.colors.qualitative.Vivid

  fig.add_trace(go.Bar(x=labels, y=values, name=col, marker=dict(color=colors), showlegend=False), row=1, col=1)
  fig.add_trace(go.Pie(labels=labels, values=values, name=col, marker=dict(colors=colors)), row=1, col=2)

  fig.update_layout(height=600, legend=dict(title=col))
  fig.show()
```

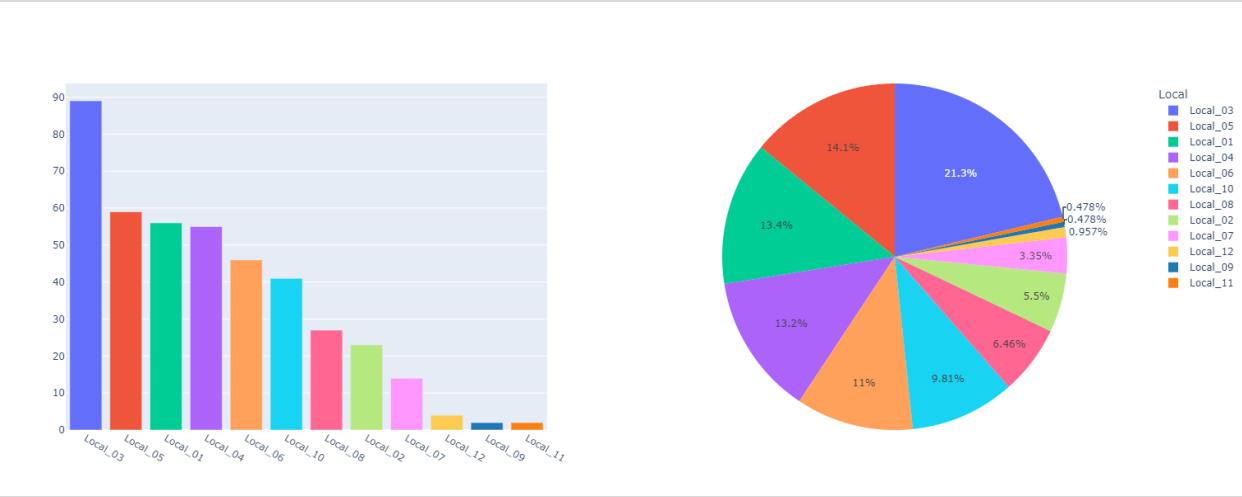
```
columns = df.drop(columns=['Date', 'Description', 'WeekofYear', 'Day']) .columns

for col in columns:
    univariate_analysis(col, df)
```

a. Country



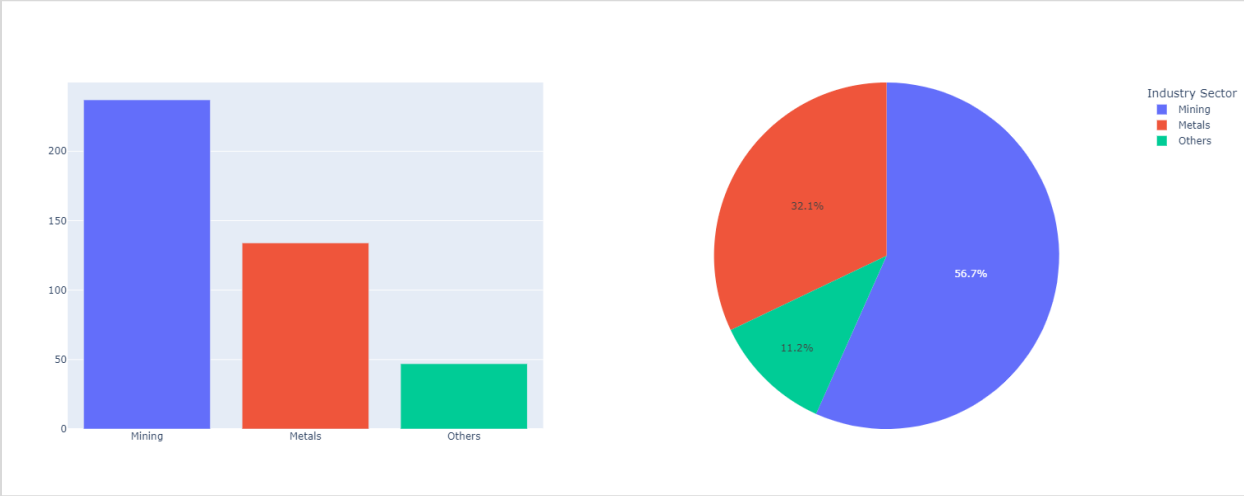
As indicated in the dataset we have data from 3 countries. Most of the observations are for Country_01.

b. Local



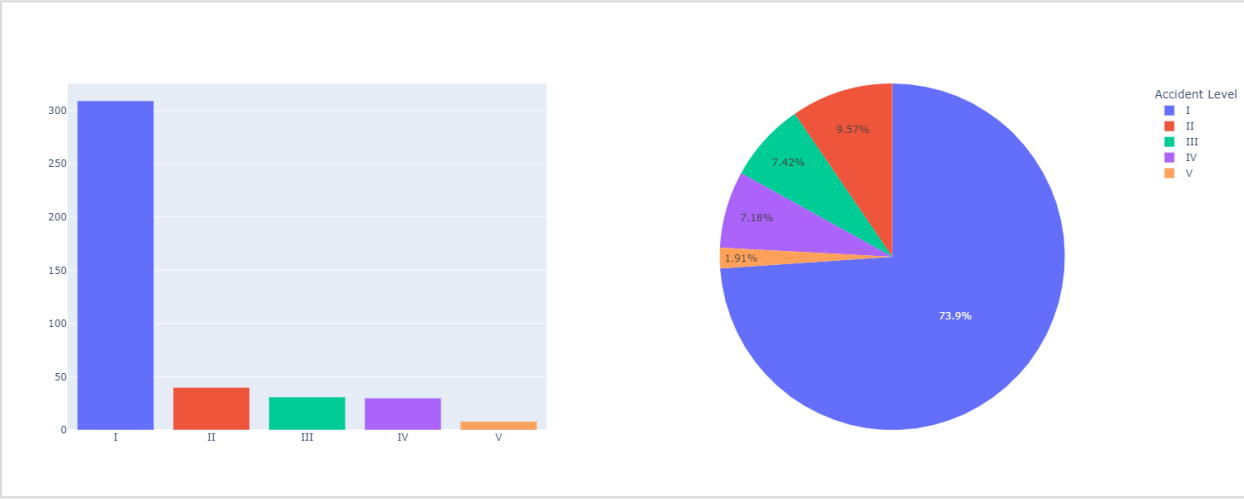
Data belongs to 12 different locations where the plant is located. Highest observations are noted at Local_03 and then Local_05, Local_01 and Local_04.

c. Industry Sector



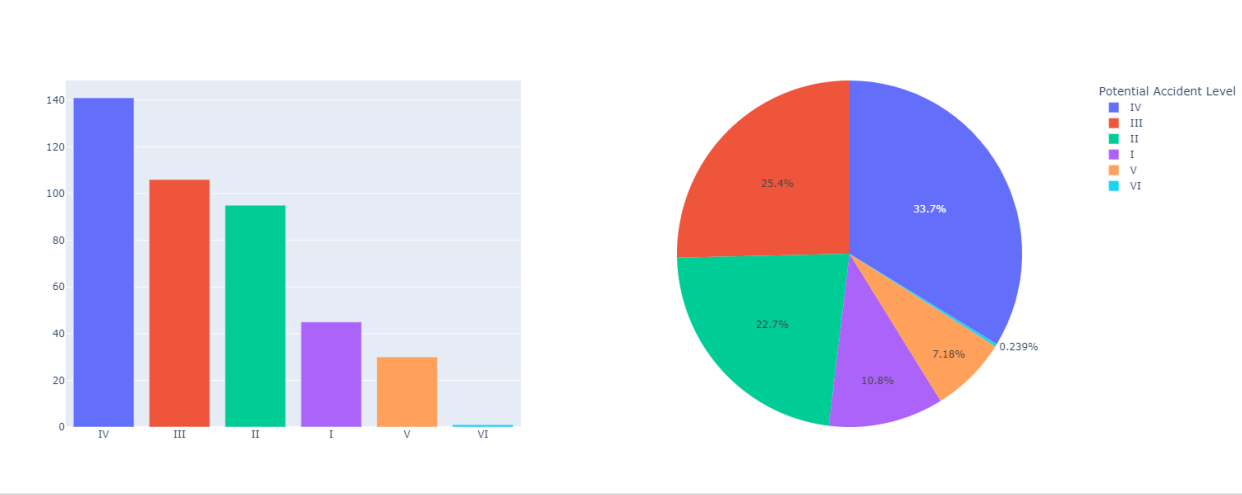
Data is gathered across three industry sectors. Most of the incidents seem to occur in the Mining industry followed by Metals.

d. Accident Level



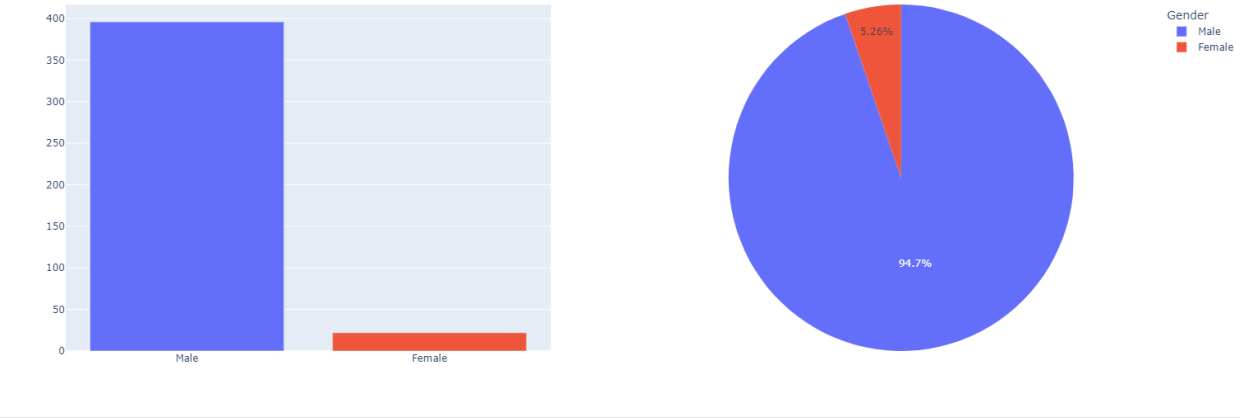
This indicates the severity of the incident. Most of the observation falls under Level I which is not severe/minor injuries. The severe injury incidents are very less which is V with a percentage of 1.91%. This will lead to data imabance issues while preparing the models by considering Accident Level as the target class.

e. Potential Accident Level



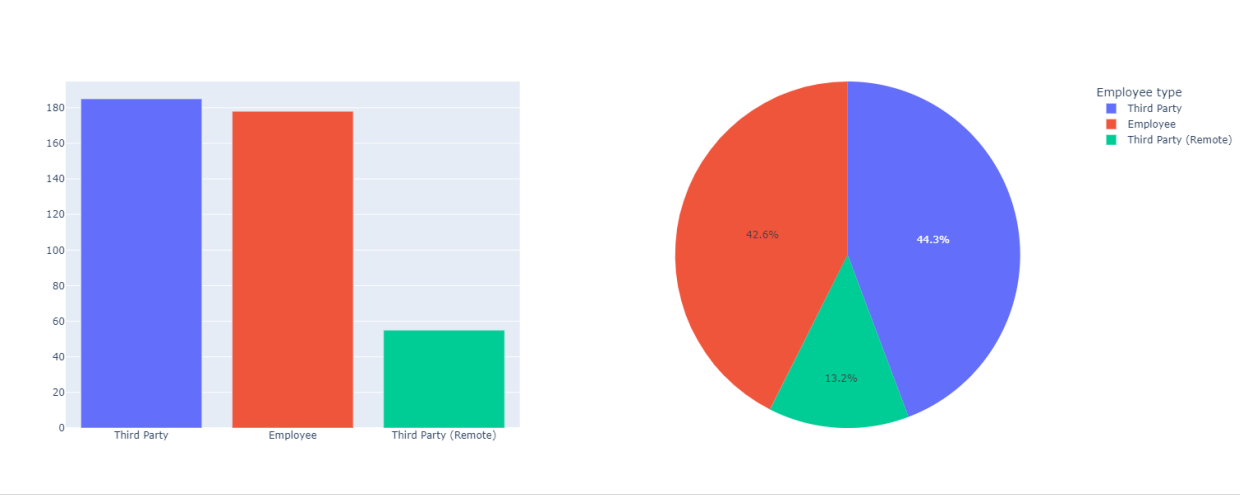
Here Potential Accident Level IV is having more incidents with a percentage of 33.7%. Potential Accident Level III and II are with nearly equal percentage of incidents which are 25.4% and 22.7%. Potential Accident Level VI is having very less incident percentage which is 0.239% only.

f. Gender



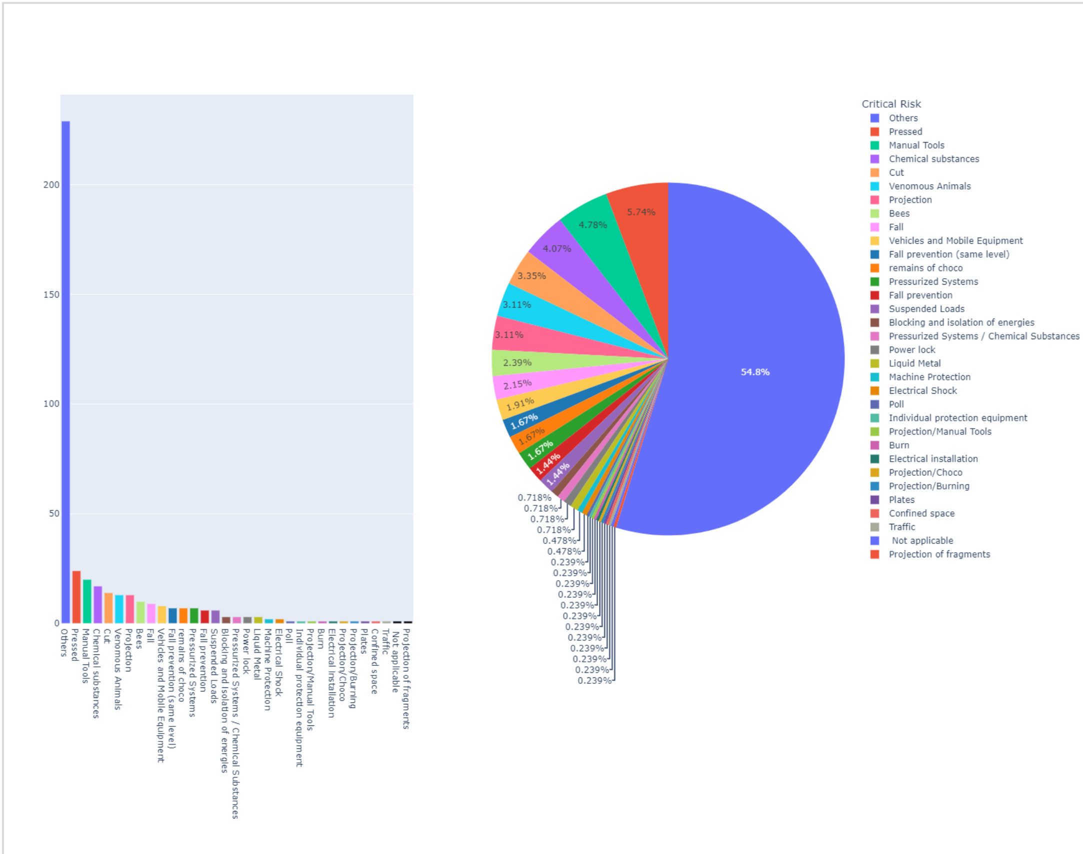
Gender ratio that had most of the accidents is Male which is at the rate of 94.7%. Usually in the Mining and Heavy Metal industry most of them are Male workers and hence most of the accidents are recorded under this category.

g. Employee type



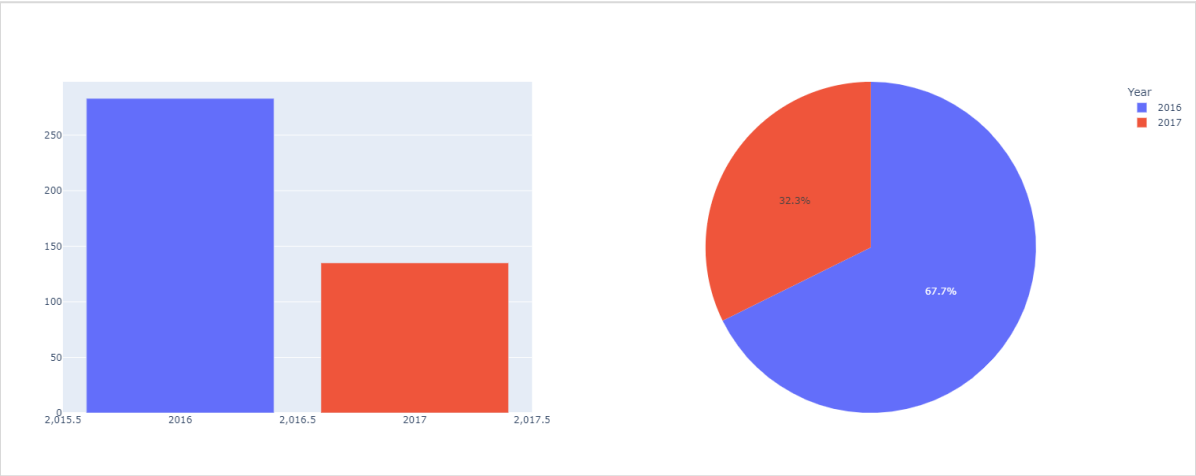
We notice that the accidents are majorly concentrated with “Third Party” and “Employee” and seemingly less with “Third party (Remote)” as they are usually operating outside the industrial environment..

h. Critical Risk



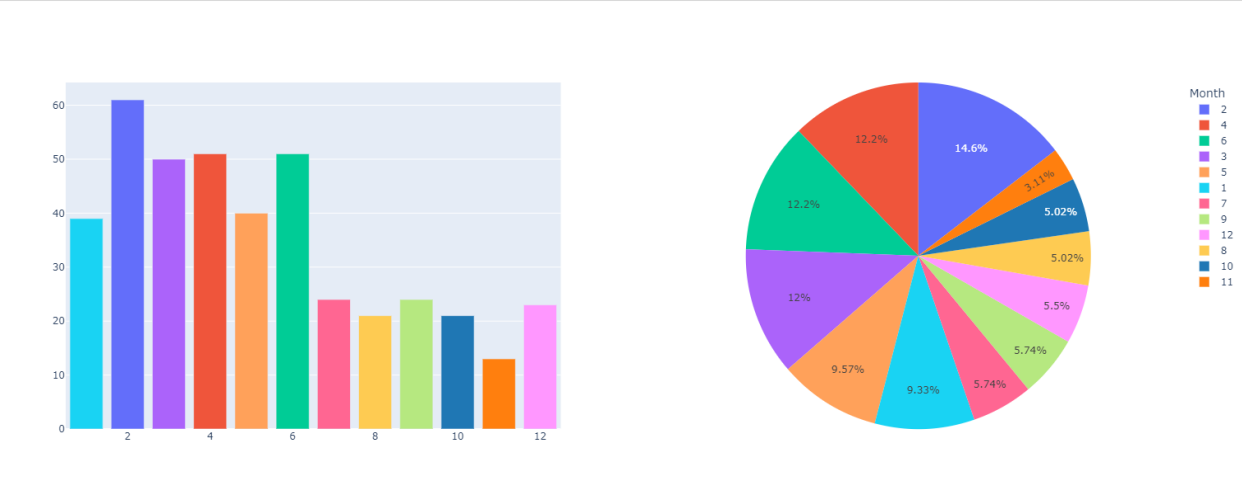
Some description of the risk involved. If we look closely, most of the observations are marked as “Others”. These are uncategorized.

i. Year



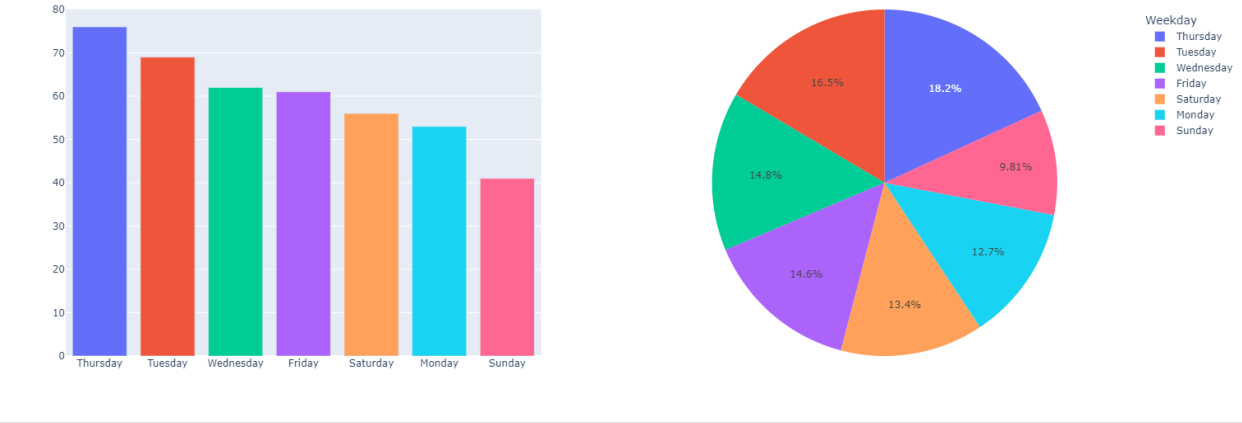
Here we can see the year 2016 has more incident data.

j. Month



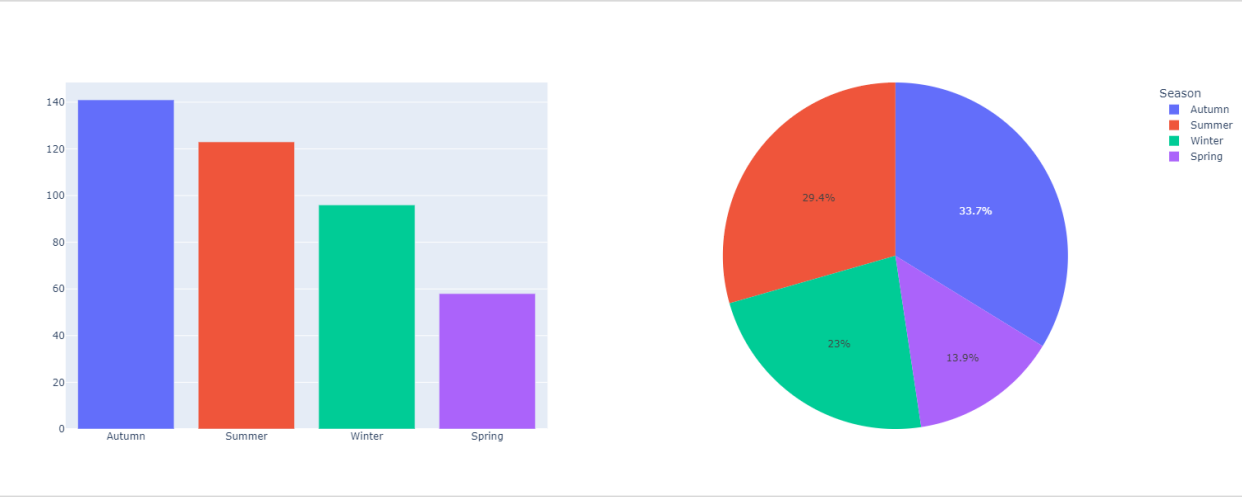
Here we can see the months 2, 4 and 6 have more incident data as compared to other months. The month 11 has very little incident data.

k. Weekday



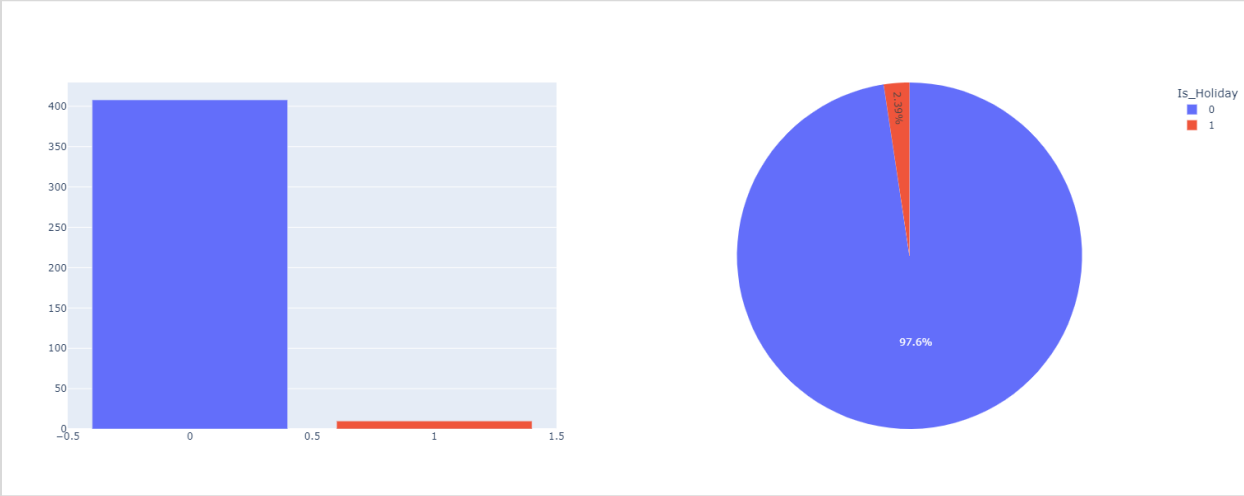
Here we can see the week days Thursday, Tuesday and Wednesday have more incident data as compared to other months. The weekday Sunday has very little incident data.

l. Season



Season during which most of the accidents seem to occur. Autumn seems to be the season with most observations followed by summer.

m. Is_Holiday



Here we can see on holidays, the incident data are very less.

5. Bivariate Analysis

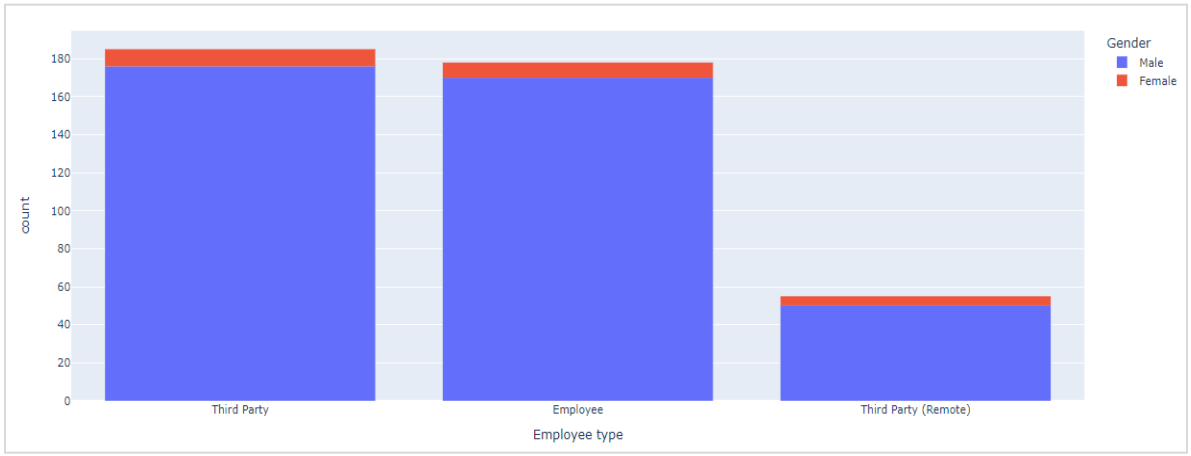
In this section we are analyzing two observations together to understand the correlation between them. For this we have created a reusable function which will take the column name, hue and dataset as the parameters and generate the visualizations dynamically.

```
def bivariate_analysis(df, col, hue):
  fig = px.histogram(df, x=df[col], color=hue, width=800, height=400, title=f'{hue} vs {col} analysis')
  fig.show()
```

Gender wide observations

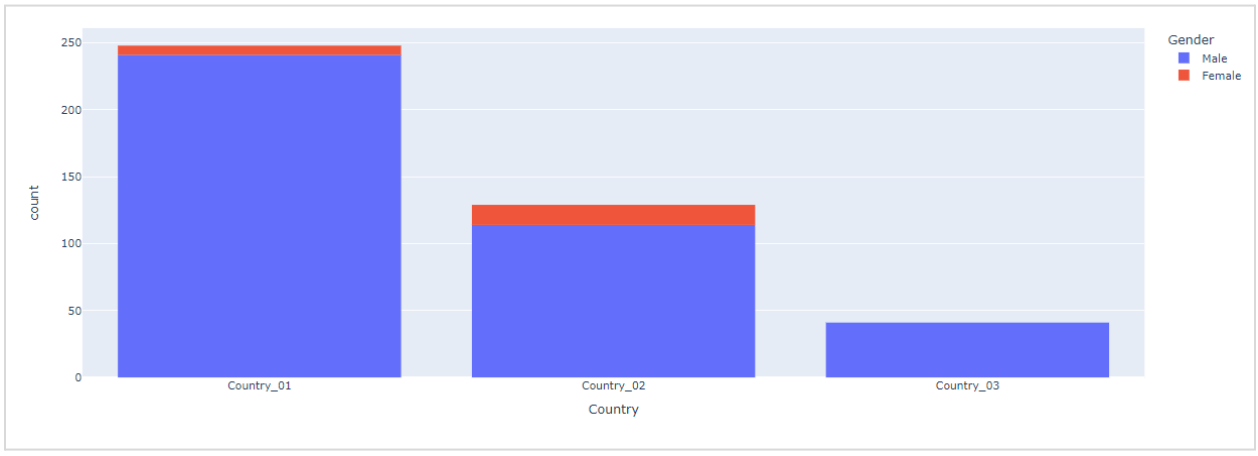
```
hue = 'Gender'
columns = ['Employee type', 'Country', 'Industry Sector', 'Is_Holiday', 'Accident Level', 'Weekday', 'Year', 'Season']
for col in columns:
    bivariate_analysis(df, col, hue)
```

a. Gender v/s Employee Type



Above chart defines the type of employees and their gender count. Most of the observations are for “Third Party” and “Employee” with Male count being the vast majority.

b. Gender v/s Country

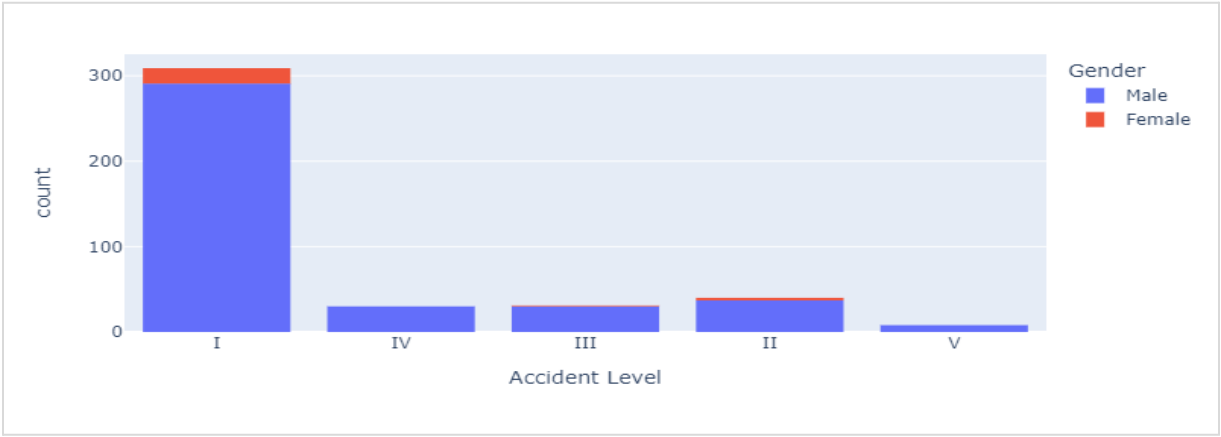


The above chart shows the correlations between country to gender count. Country_01 seems to have the highest number of accidents, followed by Country_02 and most of them being Male population.

c. Gender v/s Industry Sector Analysis

The above graph shows the Industry sector to gender count. Mining industry has caused the highest accidents followed by the Metals industry, with Male population being involved in the accidents the most.

d. Gender v/s Accident Level



From the above graph, we notice that the majority of the accidents are of Level-I which are not severe, and again this is topped by the Male employees. Female employees are not involved in Level IV or Level V accidents.

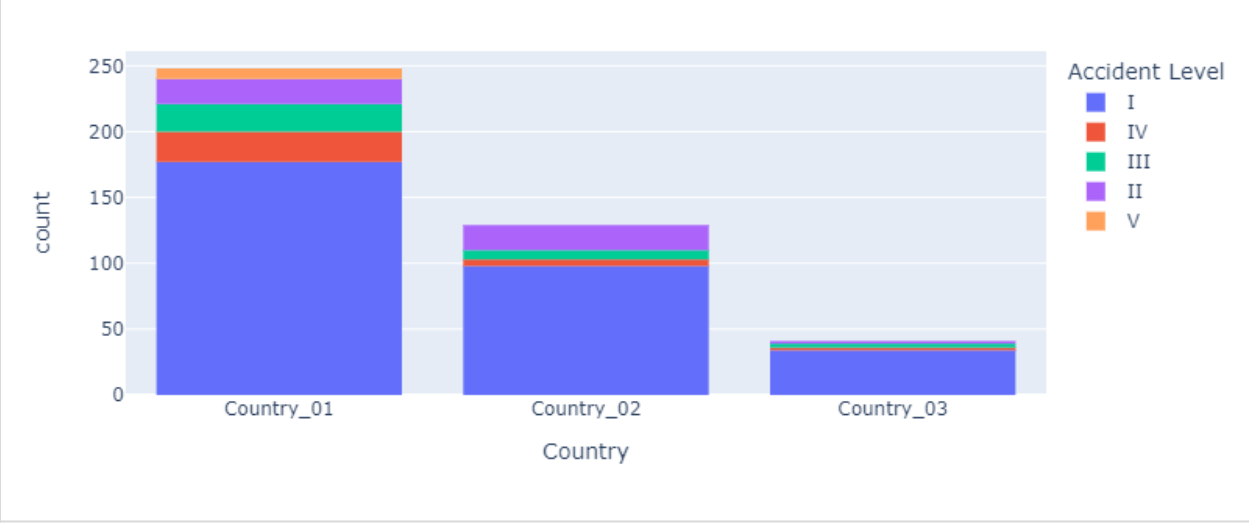
Accident Level wide observations

```
hue = 'Accident Level'
columns = ['Employee type', 'Country', 'Industry Sector', 'Is_Holiday', 'Gender', 'Weekday', 'Year', 'Season', 'Local']
for col in columns:
    bivariate_analysis(df, col, hue)
```

a. Accident Level v/s Employee Type

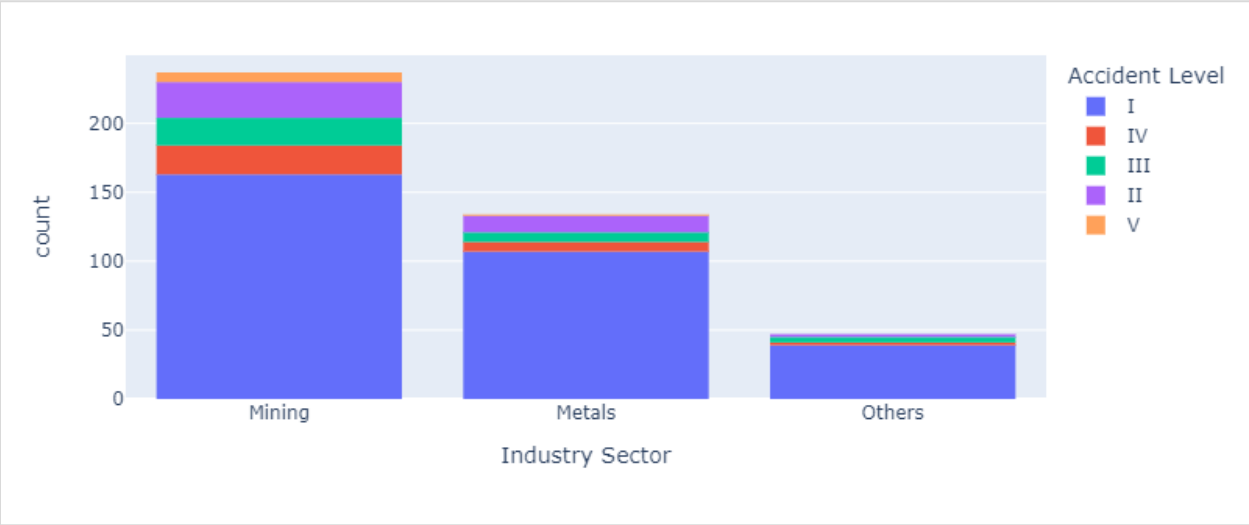
From the above graph, we observe that the majority of the accidents have occured with Third Party followed by Employee, with vast majority being Accident Level I followed by Accident Level II. Since the remote working employees are exposed to less risk, the accidents are relatively less for this class of employees.

b. Accident Level v/s Country



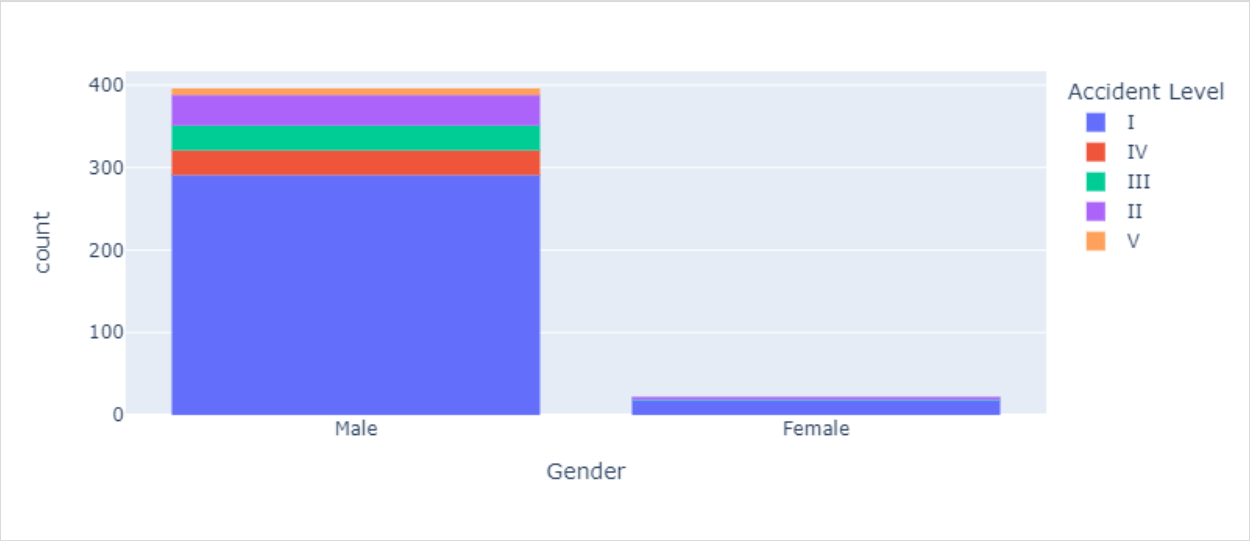
The above graph indicates that Country_01 has the highest occurrence of accidents followed by Country_02. Also, Accident Level V has not occurred in Country_02 and Country_03.

c. Accident Level v/s Industry Sector



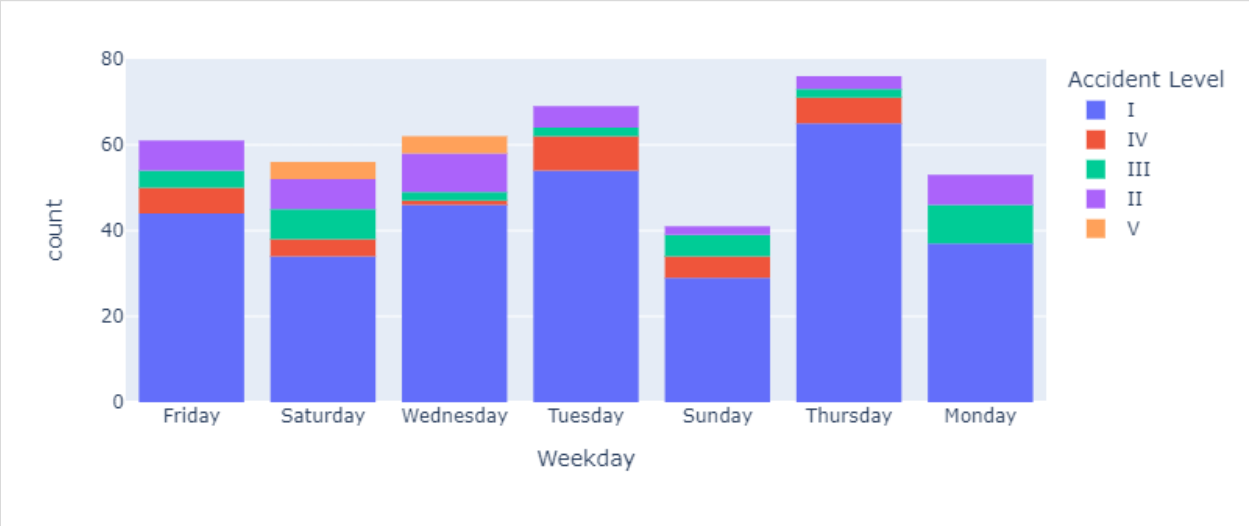
From the above graph, we find that the accidents have occured vastly in the Mining industry followed by Metals industry. Accident Level I and Level II are major contributors to the data. No Accident Level V incidents in Others industry sector.

d. Accident Level v/s Gender



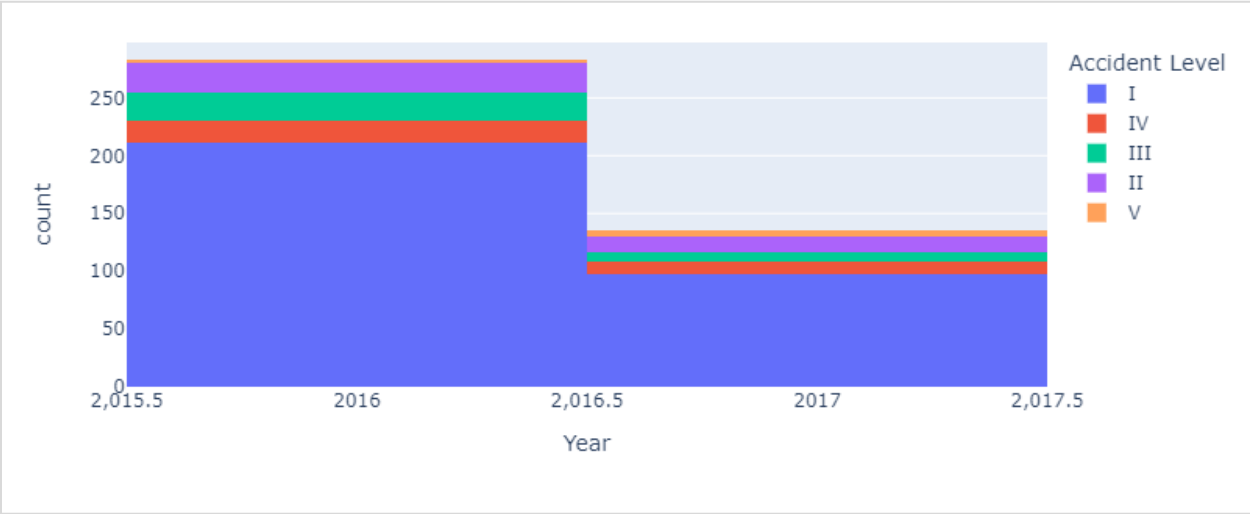
From the above graph, we observe that the Male population is mostly impacted by the accidents, with the majority of them being Accident Level I and Level II. No women population were involved in Accident Level iV or Level V accidents.

e. Accident Level v/s Weekday



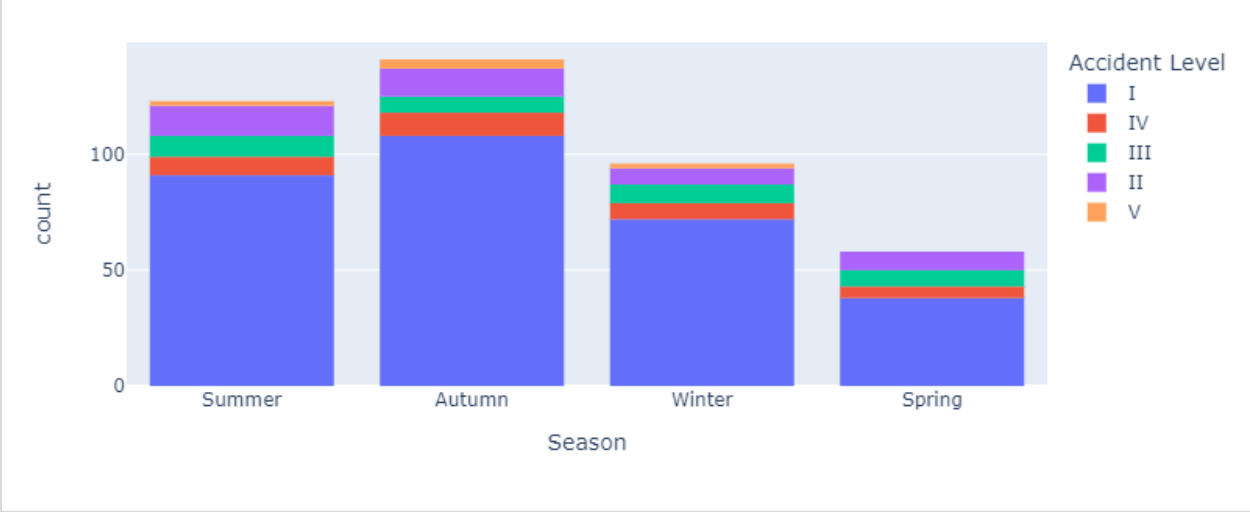
We observe that the accidents have occurred the most on Thursdays and Tuesdays, followed by the second most occurrences on Wednesdays and Fridays.

f. Accident Level v/s Year



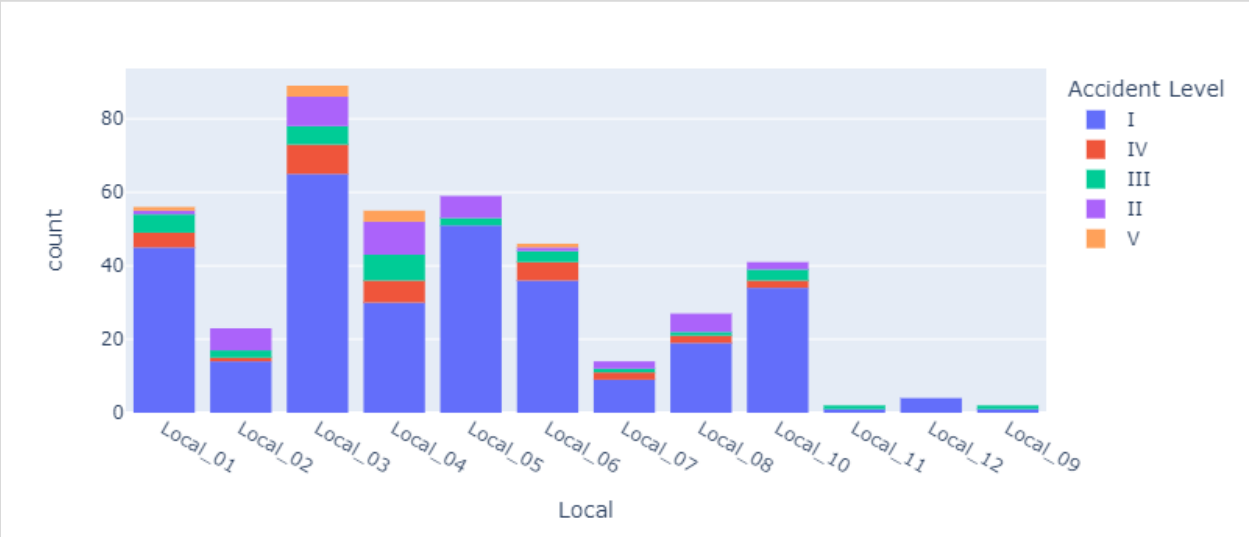
The data for accidents are recorded from Jan-2016 until July-2017. The majority of the accidents have occurred during 2016. Maybe if the data for 2017 was complete until year end, we would have noticed a different pattern.
Also, the majority of the accidents are of Accident Level I.

g. Accident Level v/s Season



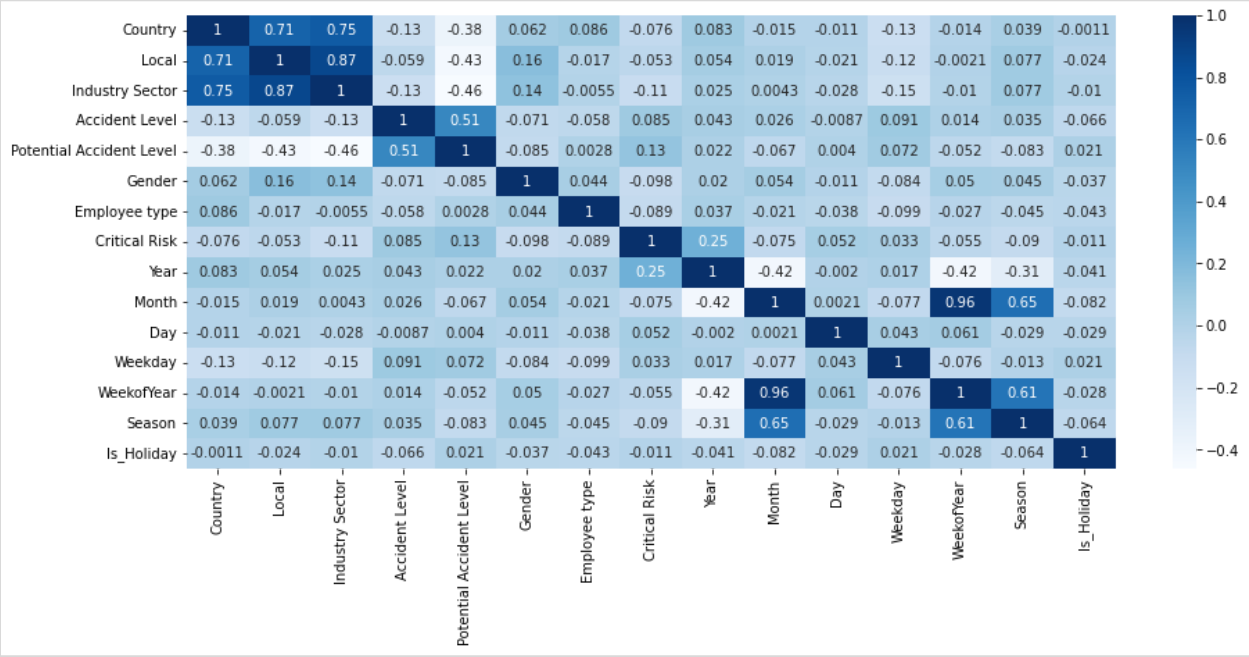
The accidents are more likely to have occurred during Autumn season followed by second major occurrence during Summer season.

h. Accident Level v/s Location



From the above graph, we notice that the location which is extremely prone to accidents is Local_03, followed by Local_05. The next most affected locations are Local_04 and Local_01.

6. Multivariate Analysis



Here we can see the correlation between the fields are very less except the extracted fields from the **Date** column.

7. Word Cloud

Here we can see the most used words on the Description field. Few of them are listed below:

1. employee
2. operator
3. moment
4. activity
5. collaborator

8. Conclusion

Based on data analysis, we can conclude the following:.

- Post data cleansing we have 418 observations.
- There is not much of a positive correlation between parameters which is good for ML models training.
- As per available data, Male gender has the most accidents observed.
- Highest occurrence of accidents seems to happen to the third party and Male category.
- Country_01 is the country that seems to be affected the most.
- Mining industry seems to attract most of the incidents.

Milestone 2

NLP Preprocessing

Since we have to do a text classification, the preprocessing of text data is very necessary. We have a created a function to preprocess the text data as follows:

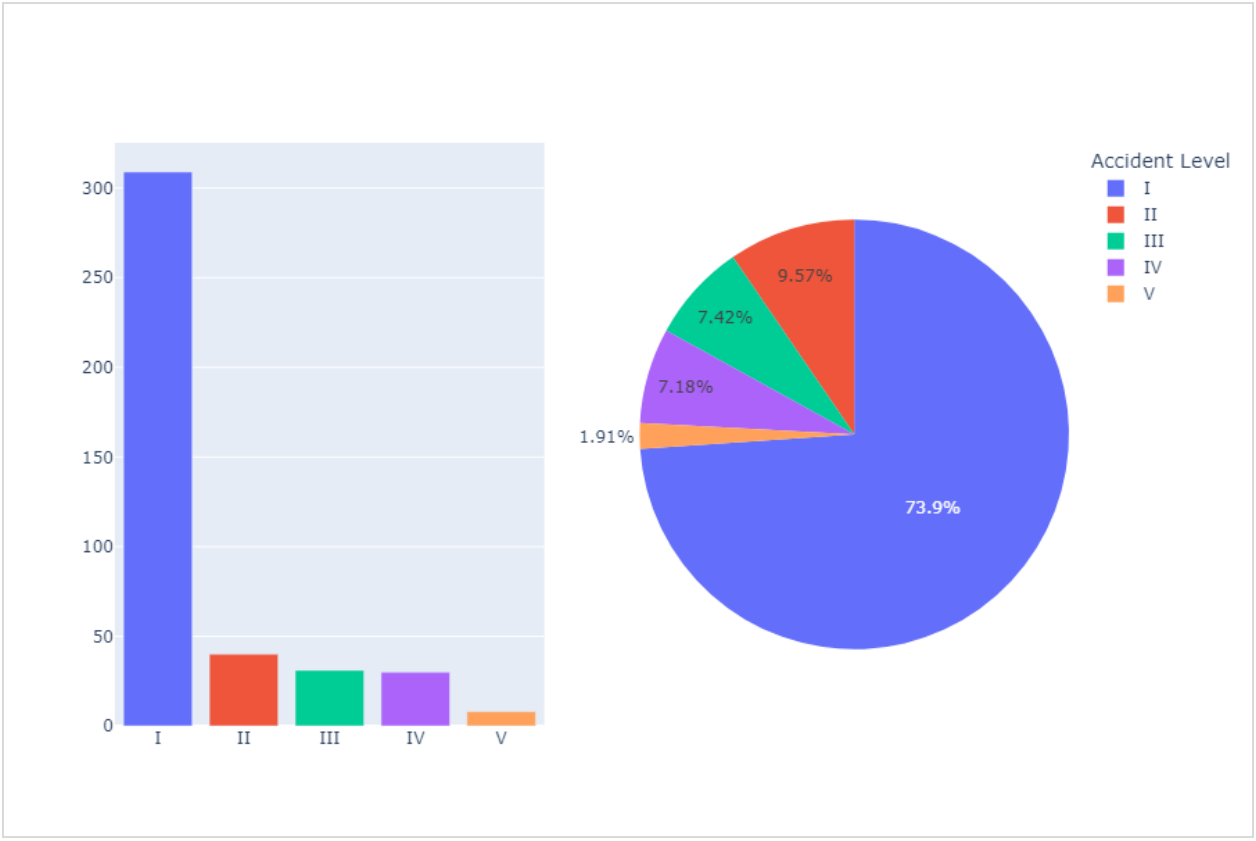
```
def preprocess_text(text):
    # Expand the contractions
    text = contractions.fix(text)

    # Remove URLs
    text = re.sub(r"https?://\S+|www\.\S+", "", text)

    # Remove HTML tags if any
    html = re.compile(r"<.*?>|&([a-z0-9]+|#[0-9]{1,6}|#x[0-9a-f]{1,6});")
    text = re.sub(html, "", text)

    # Remove Non ANSCI
    text = re.sub(r'[^\x00-\x7f]', "", text)

    # Remove imojis
    emoji_pattern = re.compile(
        '['
        u'\U0001F600-\U0001F64F'
        u'\U0001F300-\U0001F5FF'
        u'\U0001F680-\U0001F6FF'
        u'\U0001F1E0-\U0001F1FF'
        u'\U00002702-\U000027B0'
        u'\U000024C2-\U0001F251'
        ']+',
        flags=re.UNICODE)
    text = emoji_pattern.sub(r'', text)

    # Remove all special characters
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)

    # Convert to lowercase
    text = text.lower()

    # Remove unnecessary spaces
    text = text.strip()

    tokens = word_tokenize(text)
    word_list = [w for w in tokens if w not in stopwords.words('english')]
    return word_list
```

Above function, removes URLs, emoji, special characters, HTML tags, unnecessary spaces, converts text to lower case and expands on abbreviations.

1. Target Imbalance

Since we will be predicting Accident Level, it's important to look at existing data to identify the existing samples on which model will be trained.

Class imbalance can affect our model extensively and hence needs to be looked at. We found following:



From above we can see that the majority (73.9%) of the observations belong to Class I. Now this is expected since most of the accidents are of this category. This data however is highly imbalanced. If we train a model using this data, most of the observations would be incorrectly classified.

73.9% of data lies under **Accident Level I**

9.57% of data lies under **Accident Level II**

7.42% of data lies under **Accident Level III**

7.18% of data lies under **Accident Level IV**

1.91% of data lies under **Accident Level V**

We have prepared two datasets, one with balanced data and another one with the above data.

Before over-sampling:

Analyze the target class counts before over-sampling

```
[ ] print("Before UpSampling, counts of label 'I': {}".format(sum(y_train_o==1)))
    print("Before UpSampling, counts of label 'II': {}".format(sum(y_train_o==2)))
    print("Before UpSampling, counts of label 'III': {}".format(sum(y_train_o==3)))
    print("Before UpSampling, counts of label 'IV': {}".format(sum(y_train_o==4)))
    print("Before UpSampling, counts of label 'V': {}".format(sum(y_train_o==5)))
```

```
Before UpSampling, counts of label 'I': 243
Before UpSampling, counts of label 'II': 32
Before UpSampling, counts of label 'III': 23
Before UpSampling, counts of label 'IV': 29
Before UpSampling, counts of label 'V': 7
```

After over-sampling:

Analyze the target class counts after over-sampling

```
[ ] print("Before UpSampling, counts of label 'I': {}".format(sum(y_train_o==1)))
    print("Before UpSampling, counts of label 'II': {}".format(sum(y_train_o==2)))
    print("Before UpSampling, counts of label 'III': {}".format(sum(y_train_o==3)))
    print("Before UpSampling, counts of label 'IV': {}".format(sum(y_train_o==4)))
    print("Before UpSampling, counts of label 'V': {}".format(sum(y_train_o==5)))
```

```
Before UpSampling, counts of label 'I': 243
Before UpSampling, counts of label 'II': 243
Before UpSampling, counts of label 'III': 243
Before UpSampling, counts of label 'IV': 243
Before UpSampling, counts of label 'V': 243
```

2. Vocabulary

We have removed stop words and digits from input.

Reusable Functions

```
[ ] def get_vocabularies(X):
        vocabulary = set()
        for description in X:
            words = word_tokenize(description)
            vocabulary.update(words)

        vocabulary = list(vocabulary)
        return vocabulary
```

3. TF IDF vectorizer

We have used TF IDF vector generation over imbalance and balanced data:

Imbalance data:

Prepare Train and Test sets

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X.values, y.values, test_size=0.2, random_state=7)
    print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)

    (334,) (334,) (84,) (84,)
```

Prepare TF-IDF Vectorizer

```
[ ] stop_words = stopwords.words('english') + list(punctuation)
    vectorizer = TfidfVectorizer(stop_words=stop_words, tokenizer=word_tokenize, vocabulary=vocabulary)
```

Over-sampled (Balanced data):

Apply over-sampling

```
X_train_o, y_train_o = random_over.fit_resample(X_train_o, y_train_o.ravel())
```

Analyze the target class counts after over-sampling

```
[ ] print("Before UpSampling, counts of label 'I': {}".format(sum(y_train_o==1)))
    print("Before UpSampling, counts of label 'II': {}".format(sum(y_train_o==2)))
    print("Before UpSampling, counts of label 'III': {}".format(sum(y_train_o==3)))
    print("Before UpSampling, counts of label 'IV': {}".format(sum(y_train_o==4)))
    print("Before UpSampling, counts of label 'V': {}".format(sum(y_train_o==5)))

    Before UpSampling, counts of label 'I': 243
    Before UpSampling, counts of label 'II': 243
    Before UpSampling, counts of label 'III': 243
    Before UpSampling, counts of label 'IV': 243
    Before UpSampling, counts of label 'V': 243
```

Vectorize X_train_o and X_test_o

```
[ ] X_train_o = vectorizer.transform(X_train_o['description_processed_lemmatized'].values)
    X_test_o = vectorizer.transform(X_test_o['description_processed_lemmatized'].values)
```

4. Conclusion

a. We have successfully vectorized the **Description** column which can now be used with ML models.
b. We have used Vocabulary to correct grammar and also remove stop words.
c. Since this is a ChatBot application, we can also build an abbreviation list to be passed as lots of abbreviated words are used in typed applications.
d. We are considering both Balanced and Imbalanced dataset to create models.

Machine Learning Classifiers

In our project, we have implemented and built various classification models as below:

1. Multinomial NB
2. Logistic Regression
3. Gaussian NB
4. KNN
5. SVM
6. Decision Tree
7. Random Forest
8. Bagging
9. Ada Boost
10. Gradient Boosting
11. RidgeClassifier

For that, we have created a model result function, which will take the parameters X_train, y_train, X_test and y_test as below:

```
def get_ml_model_results(X_train, y_train, X_test, y_test):
    models = {
        'Multinomial NB': MultinomialNB(),
        'Logistic Regression': LogisticRegression(),
        'Gaussian NB': GaussianNB(),
        'KNN': KNeighborsClassifier(),
        'SVM': SVC(),
        'Decision Tree': DecisionTreeClassifier(criterion='entropy', max_depth=10, random_state=50, min_samples_leaf=7),
        'Random Forest': RandomForestClassifier(n_estimators=50, max_samples=7),
        'Bagging': BaggingClassifier(n_estimators=100, max_samples=10),
        'Ada Boost': AdaBoostClassifier(n_estimators=100),
        'Gradient Boosting': GradientBoostingClassifier(n_estimators=100, learning_rate=0.05),
        'RidgeClassifier': RidgeClassifier(random_state=1),
    }

    names = []
    prediction = []
    train_scores = []
    test_scores = []
    precision_scores = []
    recall_scores = []
    f1_scores = []

    for name, model in models.items():
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        train_score = model.score(X_train, y_train)
        test_score = model.score(X_test, y_test)
        ps, rs, fs, _ = precision_recall_fscore_support(y_test, y_pred, average='weighted')

        names.append(name)
        prediction.append(y_pred)
        train_scores.append(round(train_score * 100, 2))
        test_scores.append(round(test_score * 100, 2))
        precision_scores.append(round(ps * 100, 2))
        recall_scores.append(round(rs * 100, 2))
        f1_scores.append(round(fs * 100, 2))

    results = pd.DataFrame({
        'Model': names,
        'Train Accuracy': train_scores,
        'Test Accuracy': test_scores,
        'Precision': precision_scores,
        'Recall': recall_scores,
        'F1': f1_scores
    })

    return results
```

Analyze the Model scores with imbalanced data

Now let's execute the function to get the model results for Imbalanced dataset:

Analyze Model scores with Imbalanced data

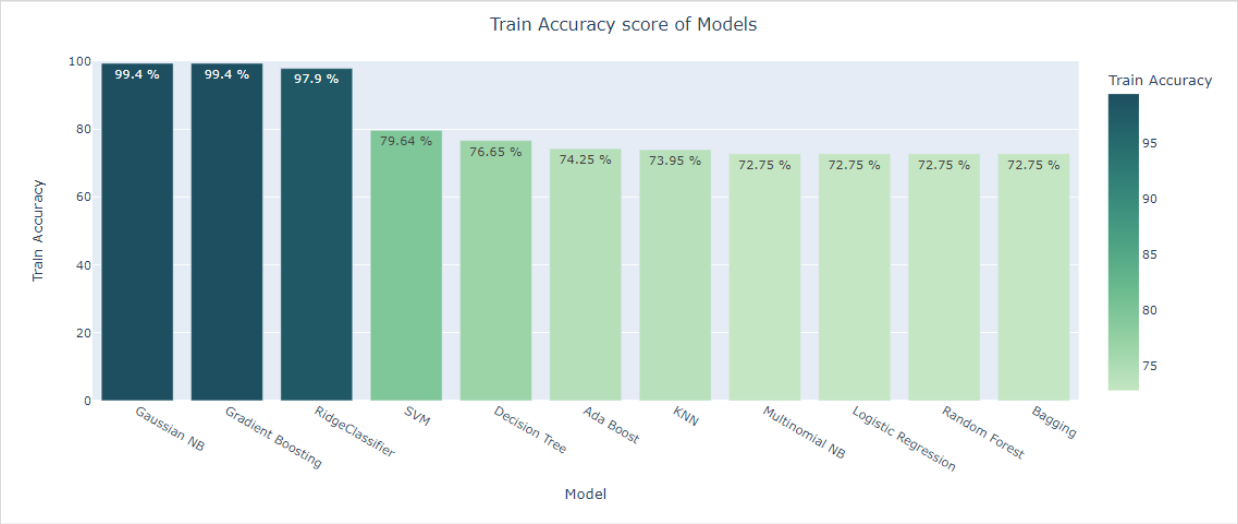
```
[ ] model_results = get_ml_model_results(X_train.todense(), y_train, X_test.todense(), y_test)
```

```
[ ] model_results.sort_values(by='Test Accuracy', ascending=False)
```

	Model	Train Accuracy	Test Accuracy	Precision	Recall	F1
0	Multinomial NB	72.75	78.57	61.73	78.57	69.14
1	Logistic Regression	72.75	78.57	61.73	78.57	69.14
2	Gaussian NB	99.40	78.57	61.73	78.57	69.14
4	SVM	79.64	78.57	61.73	78.57	69.14
6	Random Forest	72.75	78.57	61.73	78.57	69.14
7	Bagging	72.75	78.57	61.73	78.57	69.14
10	RidgeClassifier	97.90	78.57	61.73	78.57	69.14
3	KNN	73.95	76.19	62.86	76.19	68.88
8	Ada Boost	74.25	76.19	61.32	76.19	67.95
9	Gradient Boosting	99.40	76.19	62.08	76.19	68.42
5	Decision Tree	76.65	71.43	61.22	71.43	65.93

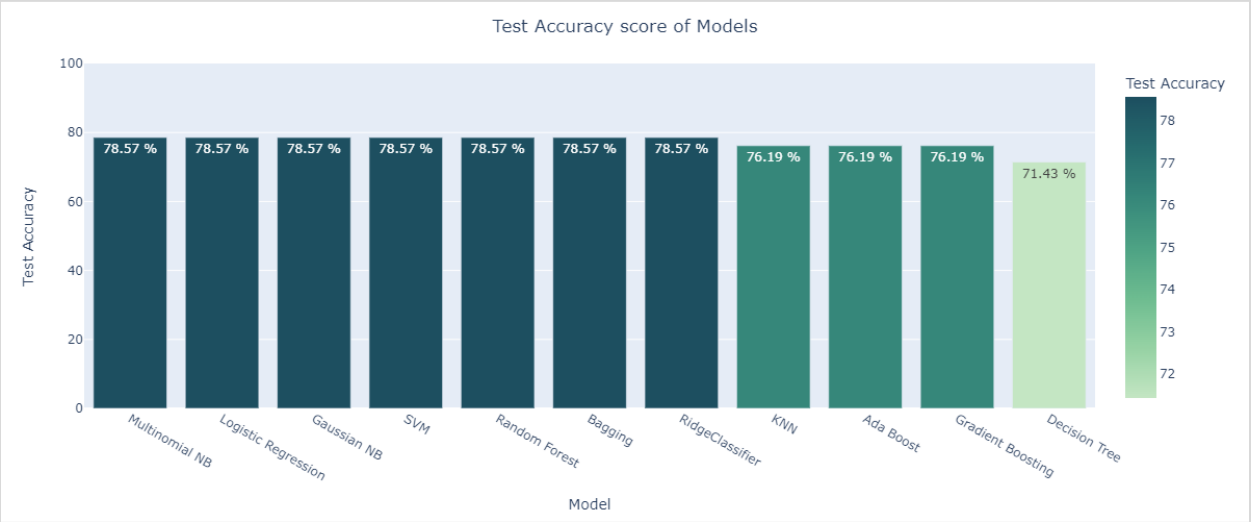
```
[ ] visualize_model_score(model_results, 'Train Accuracy')
```

Model Accuracy Analysis with Train dataset



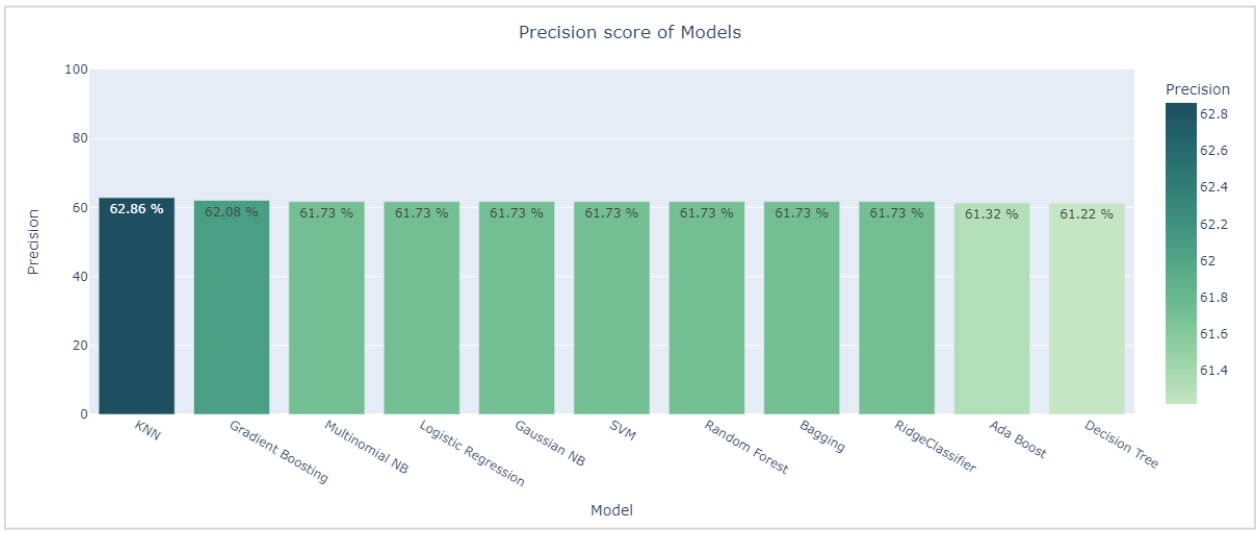
Here we can see, with the train dataset, the model accuracy is at its best for Gaussian NB and Gradient Boosting models with 99.4% accuracy. Next best performing model is Ridge Classifier with 97.9% accuracy. Other good performing models are SVM, Decision Tree, KNN and AdaBoost.

Model Accuracy Analysis with Test dataset



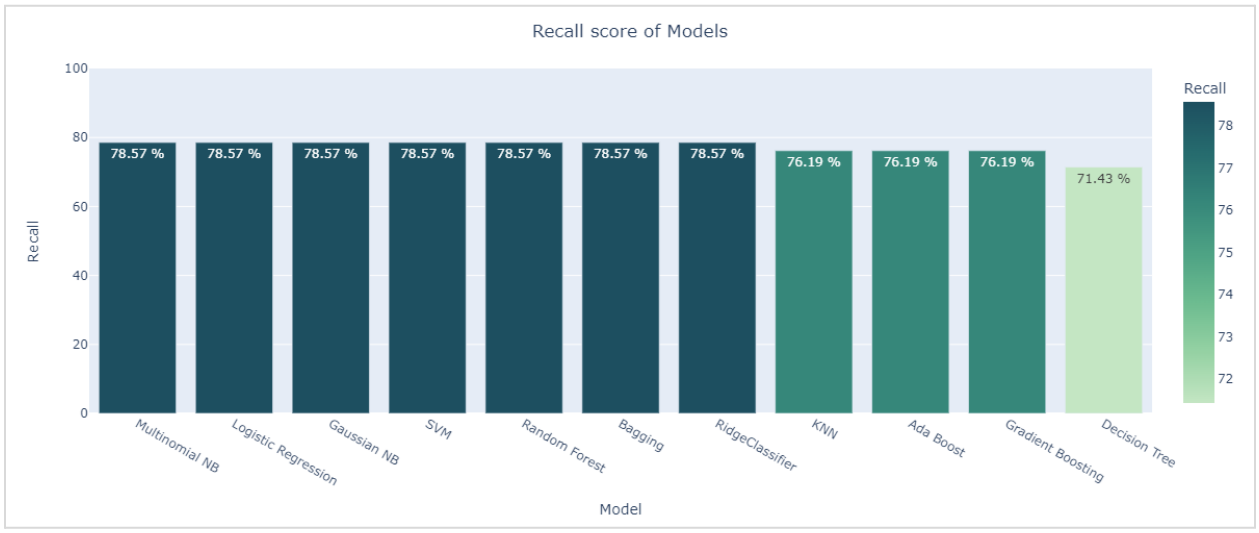
Here we can see, with the test dataset, the models Multinomial NB, Logistic Regression, SVM, Random Forest and Bagging have the same accuracy score 78.57%. Other 2 models with 77.38% accuracy are Gaussian NB and AdaBoost.

Precision Score Analysis of Models



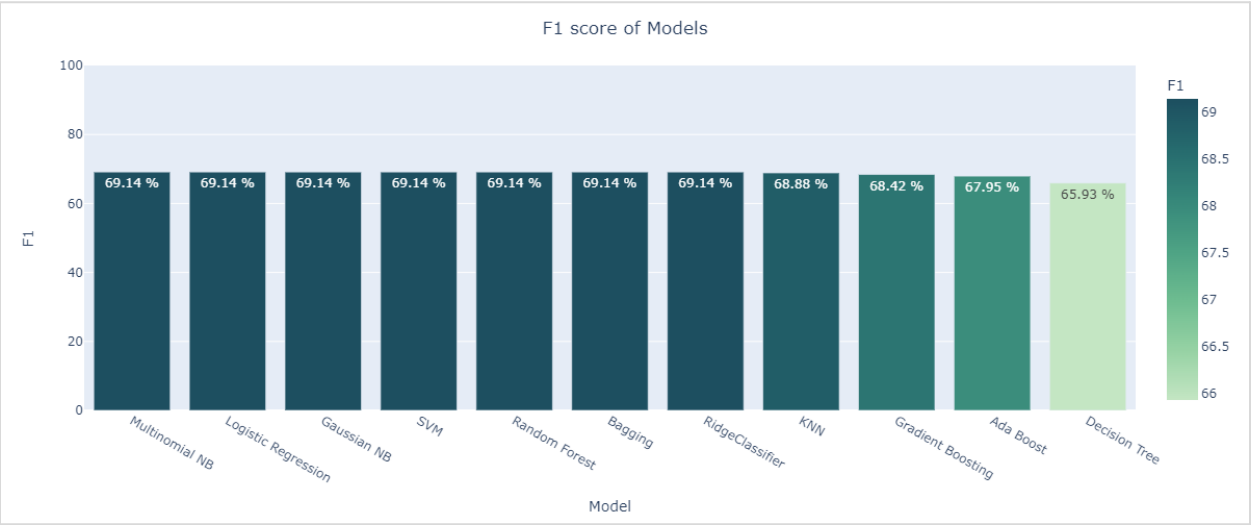
Here we can see the Precision score for almost all models has a similar score between 60 and 63 percent.

Recall Score Analysis of Models



Here we can see the Recall score for models Multinomial NB, Logistic Regression, SVM, Random Forest and Bagging have 78.57 percent.

F1 Score Analysis of Models



Here we can see the F1 score for models Multinomial NB, Logistic Regression, SVM, Random Forest and Bagging have 69.14 percent.

Analyze the Model scores with Balanced/Over-sampled data

Now let's execute the function to get the model results for Balanced dataset:

Analyze Model scores with balanced data

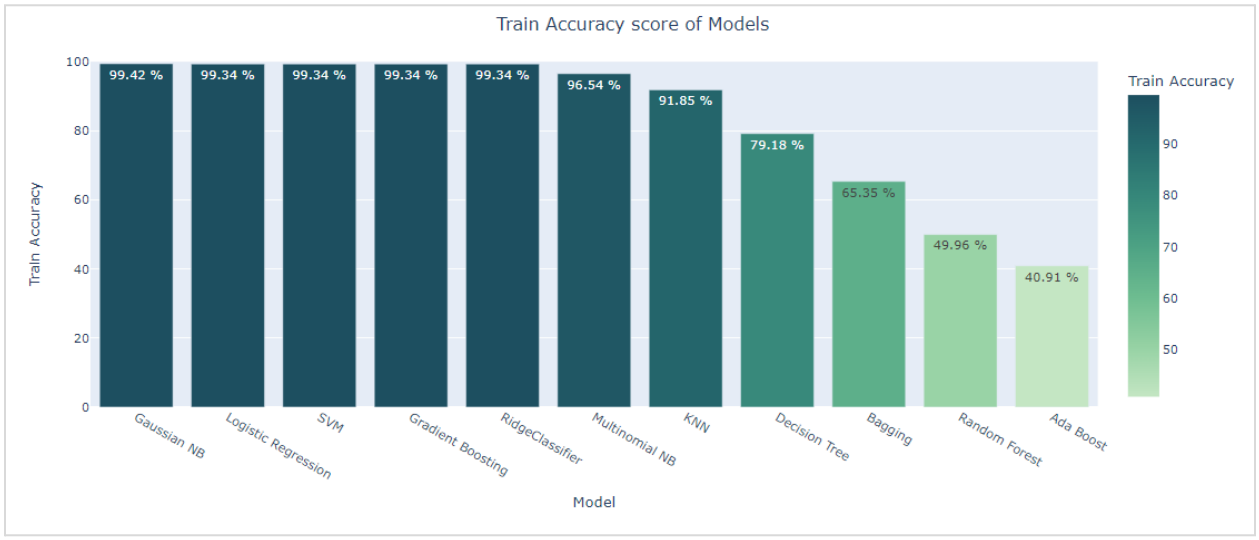
```
[ ] model_results_o = get_ml_model_results(X_train_o.todense(), y_train_o, X_test_o.todense(), y_test_o)
```

```
[ ] model_results_o.sort_values(by=['Train Accuracy', 'Test Accuracy'], ascending=False)
```

	Model	Train Accuracy	Test Accuracy	Precision	Recall	F1
2	Gaussian NB	99.42	78.57	61.73	78.57	69.14
4	SVM	99.34	78.57	61.73	78.57	69.14
1	Logistic Regression	99.34	76.19	62.08	76.19	68.42
10	RidgeClassifier	99.34	76.19	68.22	76.19	70.65
9	Gradient Boosting	99.34	65.48	66.94	65.48	66.19
0	Multinomial NB	96.54	60.71	73.83	60.71	65.93
3	KNN	91.85	40.48	64.50	40.48	49.29
5	Decision Tree	79.18	16.67	67.97	16.67	23.74
7	Bagging	65.35	30.95	58.91	30.95	37.69
6	Random Forest	49.96	47.62	64.90	47.62	54.20
8	Ada Boost	40.91	15.48	59.95	15.48	14.57

```
[ ] visualize_model_score(model_results_o, 'Train Accuracy')
```

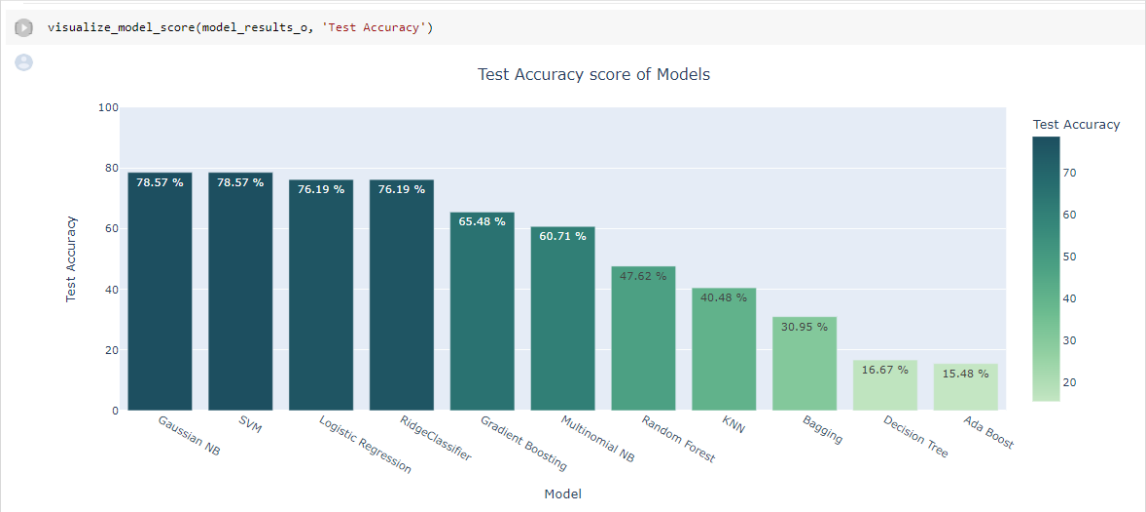
Model Accuracy Analysis with Train dataset



Here we can see, with the train dataset, the model accuracy is great for Gaussian NB and Logistics Regression. Other good performing models are SVM, Gradient Boosting and Multinomial NB.

Model Accuracy Analysis with Test dataset

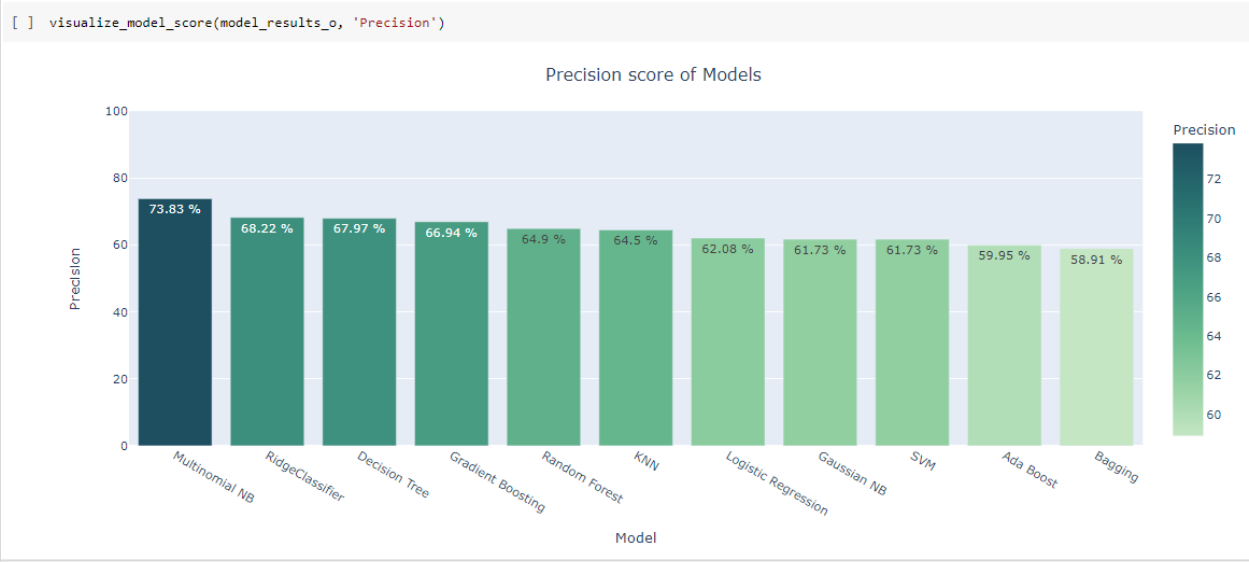
```
visualize_model_score(model_results_o, 'Test Accuracy')
```



Here we can see, with the test dataset, the model accuracy is great for Gaussian NB and SVM. Other good performing models are Logistics Regression and Ridge Classifier.

Precision Score Analysis of Model

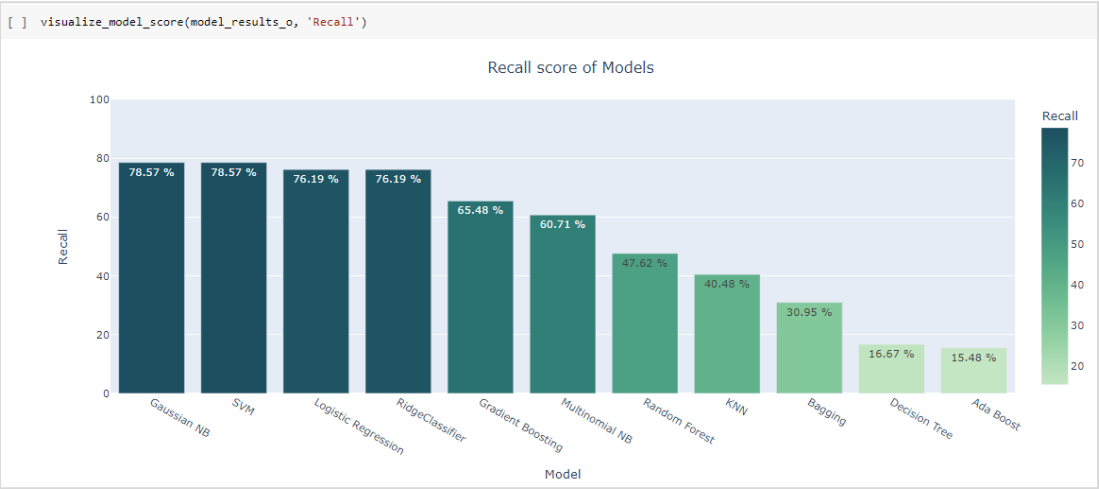
```
[ ] visualize_model_score(model_results_o, 'Precision')
```



Here we can see the precision score of all the models. Multinomial NB has the highest precision with 73.83% followed by Ridge Classifier which could only achieve 68.22%

Recall Score Analysis of Models

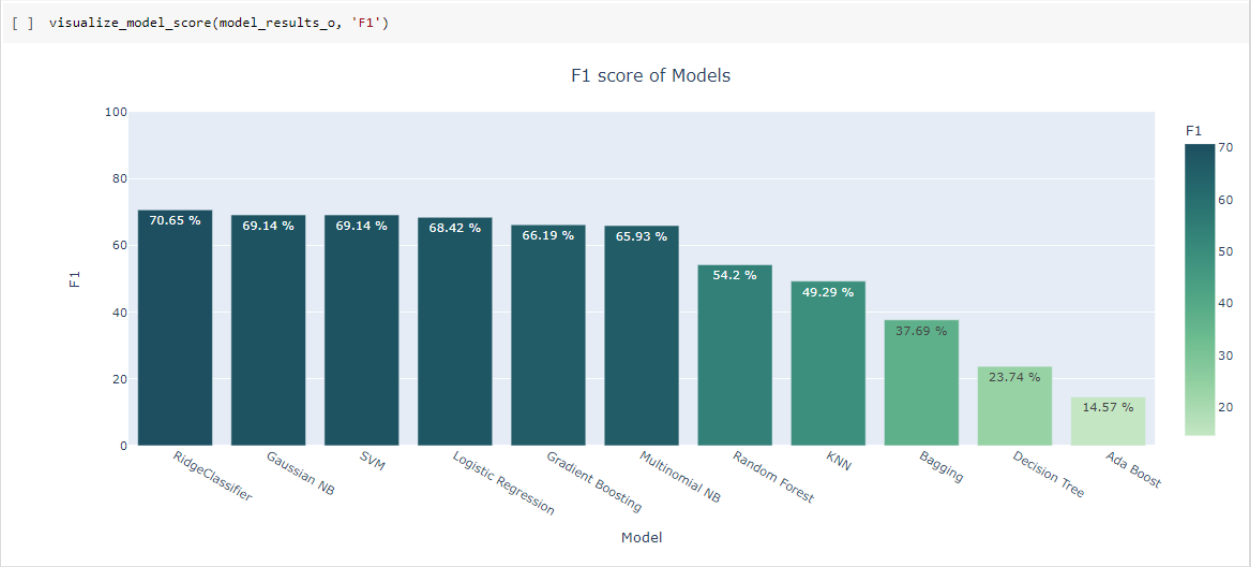
```
[ ] visualize_model_score(model_results_o, 'Recall')
```



Here we can see the Recall score for models Multinomial NB and SVM is best followed by Logistics Regression and Ridge Classifier.

F1 Score Analysis of Models

```
[ ] visualize_model_score(model_results_o, 'F1')
```



Here we can see the highest F1 score is delivered by Ridge Classifier model (70.65&) with Gaussian NB and SVM trailing not far behind with 69.14% each.

Prediction with above Models

With Imbalanced Dataset:

Predict the target class with Desciption text

```
[ ]  I_input = I['Description'].iloc[10]
     best_ml_model.predict(vectorizer.transform([I_input]))

     array([1])
```

```
[ ]  II_input = II['Description'].iloc[10]
     best_ml_model.predict(vectorizer.transform([II_input]))

     array([1])
```

Conclusion:

Since we have used imbalanced data for the creating model, the best performing model is not predicting correctly

With Balanced Dataset:

Predict the target class with Desciption text

```
[ ]  I_input = I['Description'].iloc[10]
     best_ml_model_bal.predict(vectorizer.transform([I_input]))

     array([1])
```

```
[ ]  II_input = II['Description'].iloc[10]
     best_ml_model_bal.predict(vectorizer.transform([II_input]))

     array([2])
```

```
[ ]  III_input = III['Description'].iloc[5]
     best_ml_model_bal.predict(vectorizer.transform([III_input]))

     array([3])
```

Conclusion:

This time we have used a balanced dataset for preparing the model, also we can see the model is predicting well. Going forward we will be building all the models with Upsampled(balanced) dataset, i.e. X_train_o, y_train_o, X_test_o and y_test_o

Neural Networks Classifiers

We have created 4 Neural Network sequential models using reusable functions as follows:

1. Basic Neural Network with Sigmoid and Softmax as activation
2. Basic Neural Network with Relu and Softmax as activation
3. Neural Network with Batch Normalization, Relu activation and Softmax as activation
4. Neural Network with Batch Normalization, Relu activation, Dropout layer and Softmax as activation

Reusable function to train and test all the models

```
def reset_seeds(seed):
   np.random.seed(seed)
   python_random.seed(seed)
   set_seed(seed)

def get_basic_nn_model(X_train):
  reset_seeds(0)
  clear_session()

  model = Sequential()
  model.add(Dense(64, input_shape=(X_train.shape[1], )))
  model.add(Activation('sigmoid'))
  model.add(Dense(24))
  model.add(Activation('sigmoid'))
  model.add(Dense(6))
  model.add(Activation('softmax'))
  model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='SGD')
  return model

def get_nn_model_with_weight(X_train):
  reset_seeds(0)
  clear_session()

  model = Sequential()
  model.add(Dense(64, input_shape=(X_train.shape[1], ), kernel_initializer='he_normal'))
  model.add(Activation('sigmoid'))
  model.add(Dense(24, kernel_initializer='he_normal'))
  model.add(Activation('sigmoid'))
  model.add(Dense(6, kernel_initializer='he_normal'))
  model.add(Activation('softmax'))
  model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='SGD')
  return model

def get_nn_model_with_relu(X_train):
  reset_seeds(0)
  clear_session()

  model = Sequential()
  model.add(Dense(64, input_shape=(X_train.shape[1], ), kernel_initializer='he_normal'))
  model.add(Activation('relu'))
  model.add(Dense(24))
  model.add(Activation('relu'))
  model.add(Dense(6))
  model.add(Activation('softmax'))
  model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='SGD')
  return model

def get_nn_model_with_batch_normalization(X_train):
  reset_seeds(0)
  clear_session()

  model = Sequential()
  model.add(Dense(64, input_shape=(X_train.shape[1], ), kernel_initializer='he_normal'))
```

```
  model.add(BatchNormalization())
  model.add(Activation('relu'))
  model.add(Dense(24))
  model.add(BatchNormalization())
  model.add(Activation('relu'))
  model.add(Dense(6))
  model.add(Activation('softmax'))
  model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='SGD')
  return model

def get_nn_model_with_dropout(X_train):
  reset_seeds(0)
  clear_session()

  model = Sequential()
  model.add(Dense(64, input_shape=(X_train.shape[1], ), kernel_initializer='he_normal'))
  model.add(BatchNormalization())
  model.add(Activation('relu'))
  model.add(Dropout(0.2))
  model.add(Dense(24))
  model.add(BatchNormalization())
  model.add(Activation('relu'))
  model.add(Dropout(0.2))
  model.add(Dense(6))
  model.add(Activation('softmax'))
  model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='SGD')
  return model

def get_nn_model_results(X_train, y_train, X_test, y_test):
  models = {
      'Basic NN Model': get_basic_nn_model(X_train),
      'NN Model with Weight Initialization': get_nn_model_with_weight(X_train),
      'NN Model with Relu Activation': get_nn_model_with_relu(X_train),
      'NN Model with Batch Normalization': get_nn_model_with_batch_normalization(X_train),
      'NN Model with Dropout': get_nn_model_with_dropout(X_train)
  }

  names = []
  train_scores = []
  test_scores = []
  precision_scores = []
  recall_scores = []
  f1_scores = []

  for name, model in models.items():
    model.fit(X_train, y_train, epochs=100, batch_size=8, validation_data=(X_test, y_test), verbose=False)
    train_loss, train_score = model.evaluate(X_train, y_train, verbose=0)
    test_loss, test_score = model.evaluate(X_test, y_test, verbose=0)

    y_pred = model.predict(X_test, batch_size=64, verbose=0)
    y_pred_bool = np.argmax(y_pred, axis=1)
    y_test_bool = np.argmax(y_test, axis=1)
    ps, rs, fs, _ = np.average(precision_recall_fscore_support(y_test_bool, y_pred_bool), axis=1)

    names.append(name)
    train_scores.append(round(train_score * 100, 2))
    test_scores.append(round(test_score * 100, 2))
    precision_scores.append(round(ps * 100, 2))
    recall_scores.append(round(rs * 100, 2))
    f1_scores.append(round(fs * 100, 2))


  results = pd.DataFrame({
    'Model': names,
    'Train Accuracy': train_scores,
    'Test Accuracy': test_scores,
    'Precision': precision_scores,
    'Recall': recall_scores,
    'F1': f1_scores
  })

  return results

def get_nn_model(model, X_train, y_train):
  models = {
      'Basic NN Model': get_basic_nn_model(X_train),
```

```
    'NN Model with Weight Initialization': get_nn_model_with_weight(X_train),
    'NN Model with Relu Activation': get_nn_model_with_relu(X_train),
    'NN Model with Batch Normalization': get_nn_model_with_batch_normalization(X_train),
    'NN Model with Dropout': get_nn_model_with_dropout(X_train)
}

selected_model = models[model]
if selected_model == None:
  print('Model not available')
  return

selected_model.fit(X_train, y_train, epochs=100, batch_size=8, verbose=False)
return selected_model
```

Analyze Neural Network models with balanced dataset:

Analyze NN Model scores with balanced data

```
nn_model_results = get_nn_model_results(X_train_o_vect.todense(), y_train_o_cat, X_test_o_vect.todense(), y_test_o_cat)
```

```
WARNING:tensorflow:5 out of the last 9 calls to <function Model.make_predict_function.<locals>.predict_function at 0x000001D0868F3AF0> triggered tf.function retracing. Tracing is expensive and t
he excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1),
please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.
org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

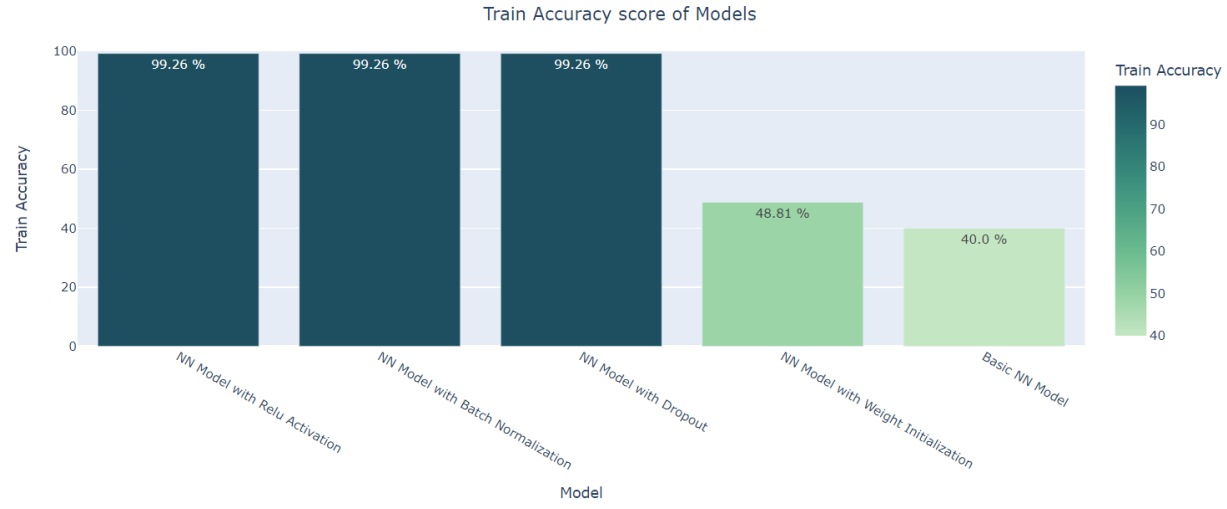
```
nn_model_results.sort_values(by=['Train Accuracy', 'Test Accuracy'], ascending=False)
```

```
nn_model_results.sort_values(by=['Train Accuracy', 'Test Accuracy'], ascending=False)
```

	Model	Train Accuracy	Test Accuracy	Precision	Recall	F1
2	NN Model with Relu Activation	99.26	77.38	35.80	21.89	21.86
3	NN Model with Batch Normalization	99.26	75.00	15.95	19.09	17.38
4	NN Model with Dropout	99.26	75.00	16.15	19.09	17.50
1	NN Model with Weight Initialization	48.81	67.86	17.31	36.97	19.21
0	Basic NN Model	40.00	78.57	15.71	20.00	17.60

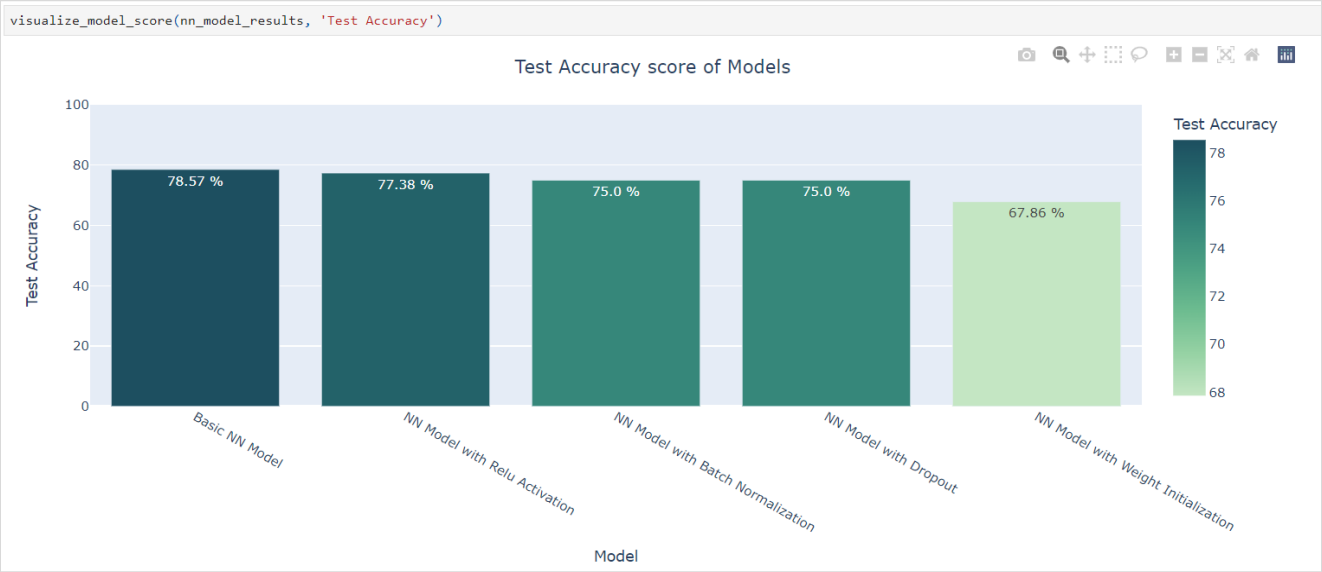
Train Accuracy Visualization:

```
visualize_model_score(nn_model_results, 'Train Accuracy')
```



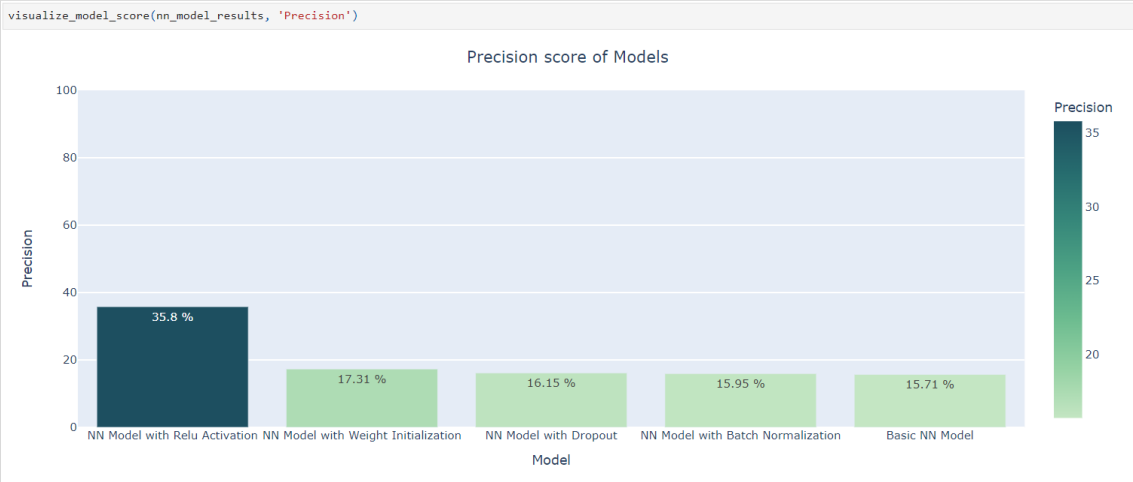
Test Accuracy visualization:

```
visualize_model_score(nn_model_results, 'Test Accuracy')
```



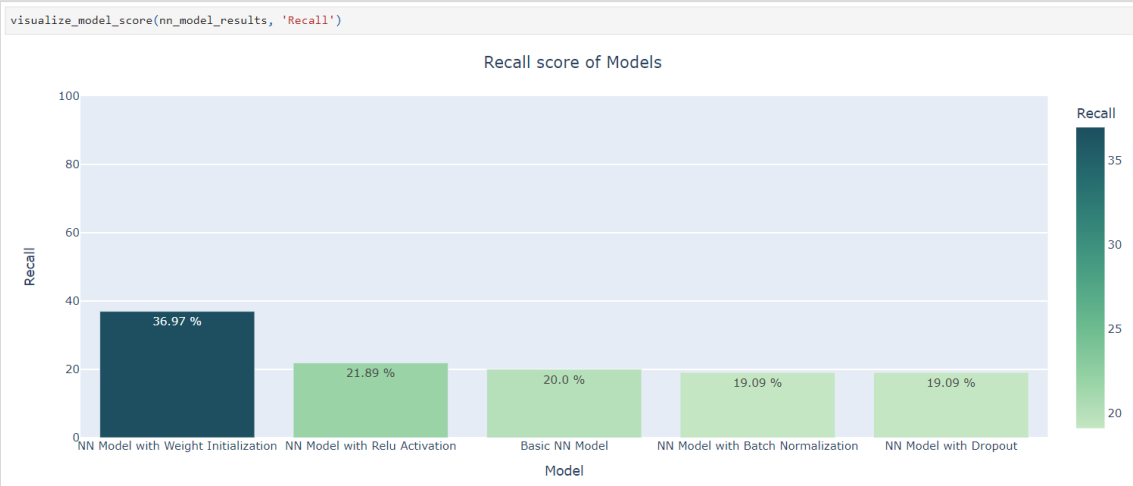
Precision score Visualization:

```
visualize_model_score(nn_model_results, 'Precision')
```



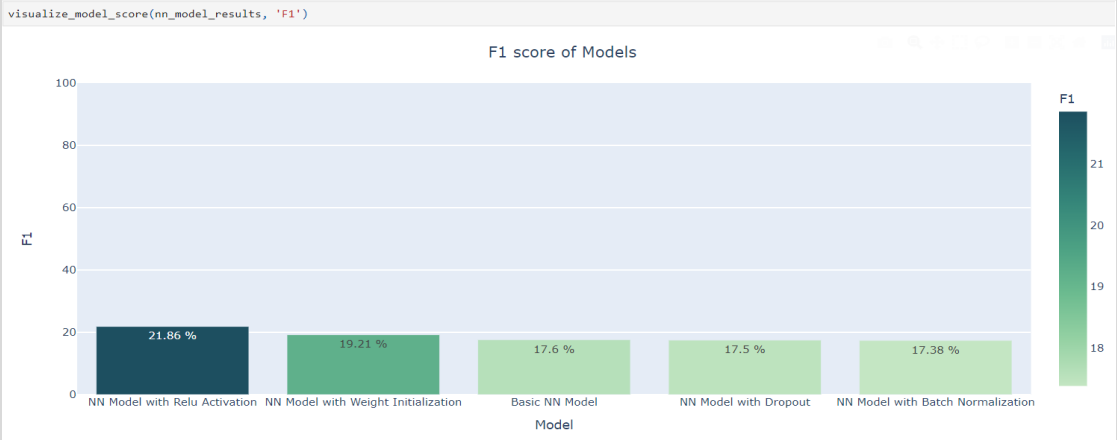
Recall Score Visualization:

```
visualize_model_score(nn_model_results, 'Recall')
```



F1 Score Visualization:

```
visualize_model_score(nn_model_results, 'F1')
```



Conclusion

From above we can see that the Neural network model with Relu Activation performed best with train accuracy of 99.26% and test accuracy of 77.38% while Model with Batch Normalization and Dropout were 99.26% in training and 75% as test accuracy. We also observed that Precision, Recall and F1 scores of all these models are rather low.

Analyze the Prediction capability of best performing model

Let's use best performing model and perform some predictions:

Analyze the prediction capability of best performing model

Get the best performing NN model

```
best_nn_model = get_nn_model('NN Model with Relu Activation', X_train_o_vect.todense(), y_train_o_ca
```

Predict the target class with Desciption text

```
np.argmax(best_nn_model.predict(vectorizer.transform([I['Description'].iloc[13]]).todense()))
```

```
WARNING:tensorflow:6 out of the last 11 calls to <function Model.make_predict_function.<locals>.pred
the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
(1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing
flow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/func
1/1 [==============================] - 0s 41ms/step
1
```

```
np.argmax(best_nn_model.predict(vectorizer.transform([III['Description'].iloc[13]]).todense()))
```

```
1/1 [==============================] - 0s 18ms/step
3
```

```
np.argmax(best_nn_model.predict(vectorizer.transform([V['Description'].iloc[7]]).todense()))
```

```
1/1 [==============================] - 0s 18ms/step
5
```

This model works rather well and is able to predict as expected.

Save & Load Model and use it for prediction:

Save the best performing model

```
best_nn_model.save('best_nn_model.h5')
```

Load back the best performing model

```
loaded_model = load_model('best_nn_model.h5')
```

Predict the target class with Desciption text

```
np.argmax(loaded_model.predict(vectorizer.transform(['Hand cut off']).todense()))
```

```
1/1 [==============================] - 0s 45ms/step
1
```

```
np.argmax(loaded_model.predict(vectorizer.transform([V['Description'].iloc[7]]).todense()))
```

```
1/1 [==============================] - 0s 17ms/step
5
```

It seems that the model is performing well.

RNN or LSTM Classifiers

We created a reusable function to train and test all the models as shown below.

```
def get_simple_lstm_model(max_len, top_words):
  clear_session()
  embedding_vecor_length = 32

  model = Sequential()
  model.add(Embedding(top_words, embedding_vecor_length, input_length=max_len))
  model.add(LSTM(100))
  model.add(Dense(6, activation='softmax'))
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
  return model

def get_lstm_with_dropout_model(max_len, top_words):
  clear_session()
  embedding_vecor_length = 32

  model = Sequential()
  model.add(Embedding(top_words, embedding_vecor_length, input_length=max_len))
  model.add(Dropout(0.2))
  model.add(LSTM(100))
  model.add(Dropout(0.2))
  model.add(Dense(6, activation='softmax'))
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
  return model

def get_bidirectional_lstm_model(max_len, top_words):
  clear_session()
  embedding_vecor_length = 32

  model = Sequential()
  model.add(Embedding(top_words, embedding_vecor_length, input_length=max_len))
  model.add(Dropout(0.2))
  model.add(Bidirectional(LSTM(100)))
  model.add(Dropout(0.2))
  model.add(Dense(6, activation='softmax'))
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
  return model

def get_lstm_and_cnn_model(max_len, top_words):
  clear_session()
  embedding_vecor_length = 32

  model = Sequential()
  model.add(Embedding(top_words, embedding_vecor_length, input_length=max_len))
  model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
  model.add(MaxPooling1D(pool_size=2))
  model.add(LSTM(100))
  model.add(Dense(6, activation='softmax'))
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
  return model

def get_rnn_lstm_model_results(X_train, y_train, X_test, y_test, max_len, top_words):
  models = {
      'Simple LSTM Model': get_simple_lstm_model(max_len, top_words),
      'LSTM with Droput': get_lstm_with_dropout_model(max_len, top_words),
      'Bidirectional LSTM': get_bidirectional_lstm_model(max_len, top_words),
      'LSTM and CNN': get_lstm_and_cnn_model(max_len, top_words)
  }

  names = []
  train_scores = []
  test_scores = []
  precision_scores = []
  recall_scores = []
  f1_scores = []

  for name, model in models.items():
    print('Preparing model ', name)
    model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=64, verbose=False)
    train_loss, train_score = model.evaluate(X_train, y_train, verbose=0)
    test_loss, test_score = model.evaluate(X_test, y_test, verbose=0)

    y_pred = model.predict(X_test, batch_size=64, verbose=0)
    y_pred_bool = np.argmax(y_pred, axis=1)
    y_test_bool = np.argmax(y_test, axis=1)
    ps, rs, fs, _ = np.average(precision_recall_fscore_support(y_test_bool, y_pred_bool), axis=1)
```

```
    names.append(name)
    train_scores.append(round(train_score * 100, 2))
    test_scores.append(round(test_score * 100, 2))
    precision_scores.append(round(ps * 100, 2))
    recall_scores.append(round(rs * 100, 2))
    f1_scores.append(round(fs * 100, 2))


  results = pd.DataFrame({
    'Model': names,
    'Train Accuracy': train_scores,
    'Test Accuracy': test_scores,
    'Precision': precision_scores,
    'Recall': recall_scores,
    'F1': f1_scores
  })

  return results

def get_rnn_lstm_model(model, X_train, y_train, max_len, top_words):
  models = {
      'Simple LSTM Model': get_simple_lstm_model(max_len, top_words),
      'LSTM with Droput': get_lstm_with_dropout_model(max_len, top_words),
      'Bidirectional LSTM': get_bidirectional_lstm_model(max_len, top_words),
      'LSTM and CNN': get_lstm_and_cnn_model(max_len, top_words)
  }

  selected_model = models[model]
  if selected_model == None:
    print('Model not available')
    return

  selected_model.fit(X_train, y_train, epochs=10, batch_size=64, verbose=False)
  return selected_model

def prepare_input(text):
    input_seq = tokenizer.texts_to_sequences([text])
    input_pad = pad_sequences(input_seq, maxlen=max_len)
    return input_pad
```

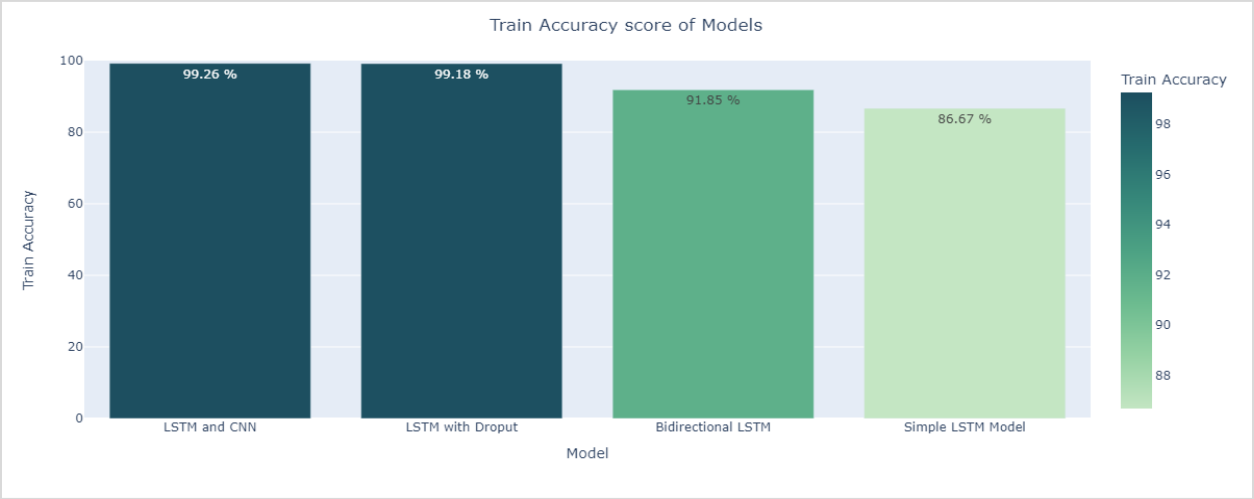
Analyze RNN or LSTM Model scores with balanced data

In [179…
```
rnn_lstm_model_results.sort_values(by=['Train Accuracy', 'Test Accuracy'], ascending=False)
```

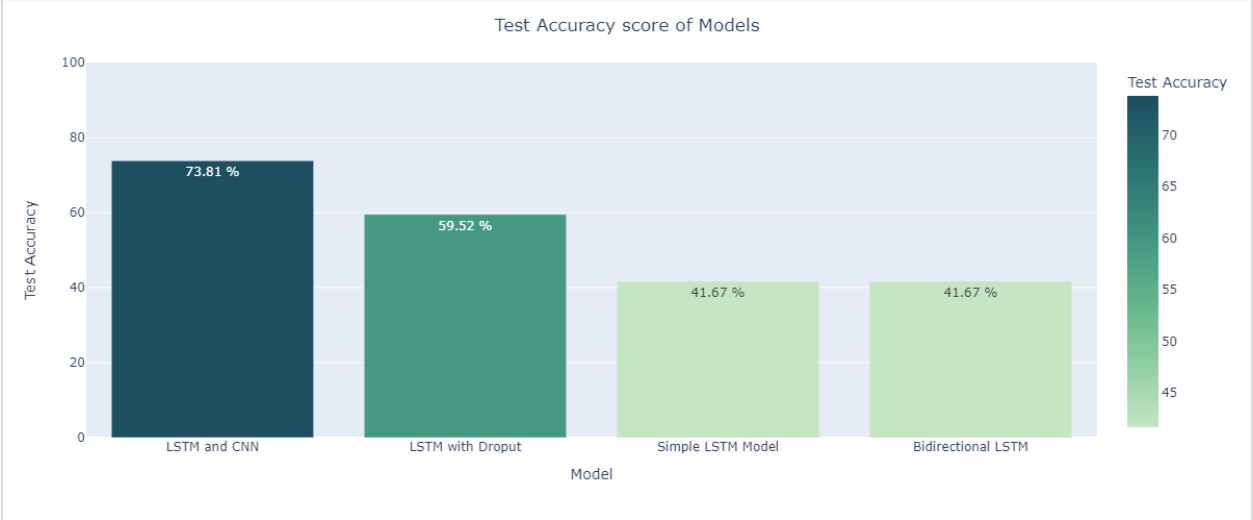
Out[179]:

	Model	Train Accuracy	Test Accuracy	Precision	Recall	F1
3	LSTM and CNN	99.26	73.81	29.12	23.18	24.17
1	LSTM with Droput	99.18	59.52	24.95	21.74	21.88
2	Bidirectional LSTM	91.85	41.67	21.26	23.79	19.40
0	Simple LSTM Model	86.67	41.67	36.64	32.50	17.83

Train Accuracy Visualization of LSTM models

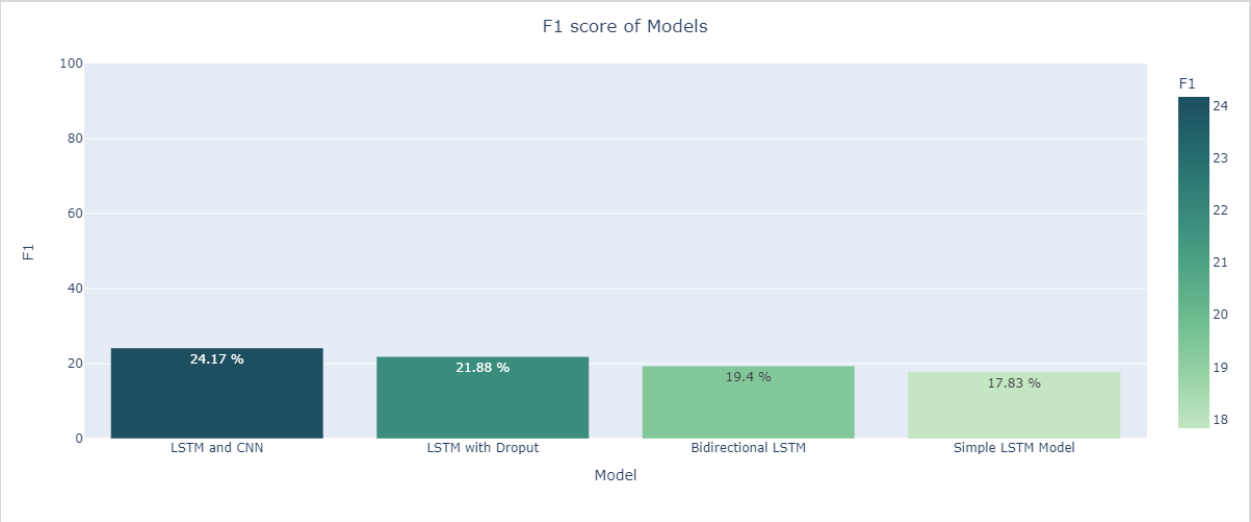


Test Accuracy Visualization of LSTM models



Precision score visualization of the LSTM models

F1 score visualization of the LSTM models



Conclusion

From above we can see that the LSTM and CNN model performed best with train accuracy of 99.26% and test accuracy of 77.31% while Bidirectional LSTM was 91.85% in training and 41.67 % as test accuracy. We also observed that Precision, Recall and F1 scores of all these models are rather low.

Choose Best Performing Model

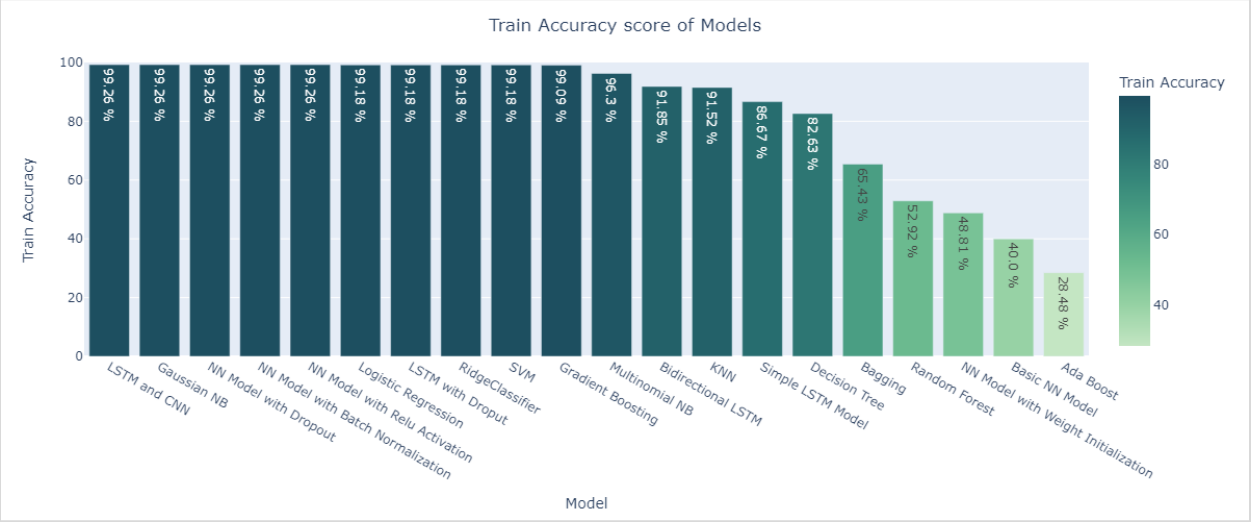
Combine all the model results

```
all_model_scores = pd.concat([model_results_o, nn_model_results, rnn_lstm_model_results])
```

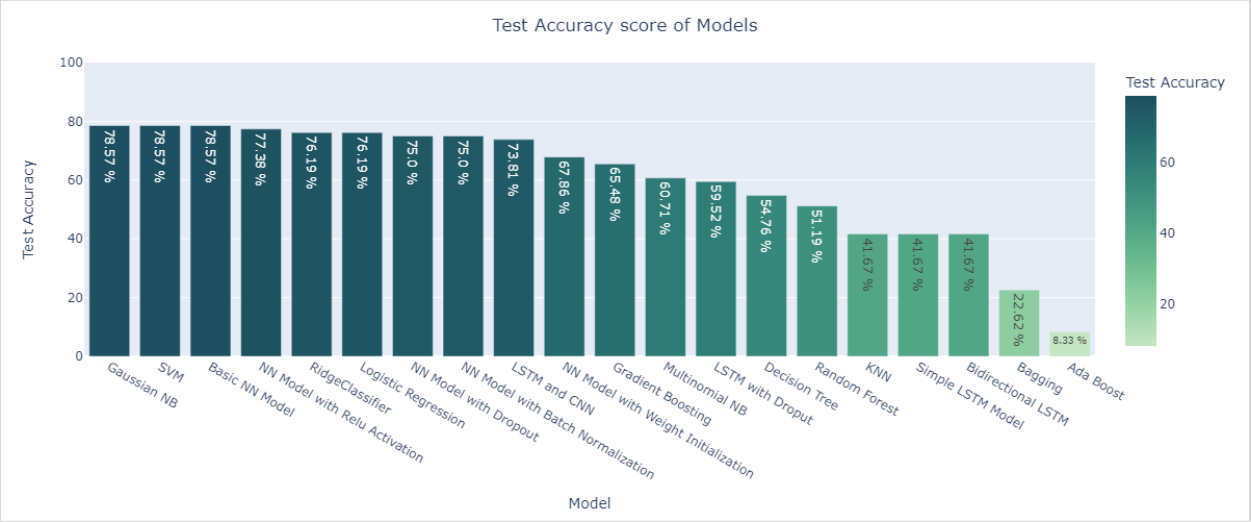
```
all_model_scores.sort_values(by=['Test Accuracy'], ascending=False)
```

	Model	Train Accuracy	Test Accuracy	Precision	Recall	F1
2	Gaussian NB	99.26	78.57	61.73	78.57	69.14
4	SVM	99.18	78.57	61.73	78.57	69.14
0	Basic NN Model	40.00	78.57	15.71	20.00	17.60
2	NN Model with Relu Activation	99.26	77.38	35.80	21.89	21.86
10	RidgeClassifier	99.18	76.19	67.42	76.19	70.18
1	Logistic Regression	99.18	76.19	62.08	76.19	68.42
4	NN Model with Dropout	99.26	75.00	16.15	19.09	17.50
3	NN Model with Batch Normalization	99.26	75.00	15.95	19.09	17.38
3	LSTM and CNN	99.26	73.81	29.12	23.18	24.17
1	NN Model with Weight Initialization	48.81	67.86	17.31	36.97	19.21
9	Gradient Boosting	99.09	65.48	61.73	65.48	63.55
0	Multinomial NB	96.30	60.71	75.38	60.71	66.10
1	LSTM with Droput	99.18	59.52	24.95	21.74	21.88
5	Decision Tree	82.63	54.76	64.97	54.76	59.38
6	Random Forest	52.92	51.19	74.59	51.19	58.65
3	KNN	91.52	41.67	64.91	41.67	50.34
0	Simple LSTM Model	86.67	41.67	36.64	32.50	17.83
2	Bidirectional LSTM	91.85	41.67	21.26	23.79	19.40
7	Bagging	65.43	22.62	66.91	22.62	29.20
8	Ada Boost	28.48	8.33	0.81	8.33	1.48

Train Accuracy Visualization of all models



Test Accuracy Visualization of all models



Pickle the Best performing model for integrating with Chatbot

Get the best model

```
In [194…  best_model = get_ml_model('SVM', X_train_o_vect, y_train_o)
```

Save the best model

```
In [197…  joblib.dump(best_model, 'best_model.pkl')
Out[197]: ['best_model.pkl']
```

Load the saved model

```
In [200…  loaded_best_model = joblib.load('best_model.pkl')
```

Predict Accident Level using Loaded best model

```
In [ ]:   II_input = II['Description'].iloc[10]
In [202…  loaded_best_model.predict(vectorizer.transform([II_input]))
Out[202]: array([2], dtype=int64)
In [203…  IV_input = IV['Description'].iloc[10]
In [204…  loaded_best_model.predict(vectorizer.transform([IV_input]))
Out[204]: array([4], dtype=int64)
```

We find that the top performing models are as below:

1. Gaussian NB
2. SVM
3. Basic NN Model

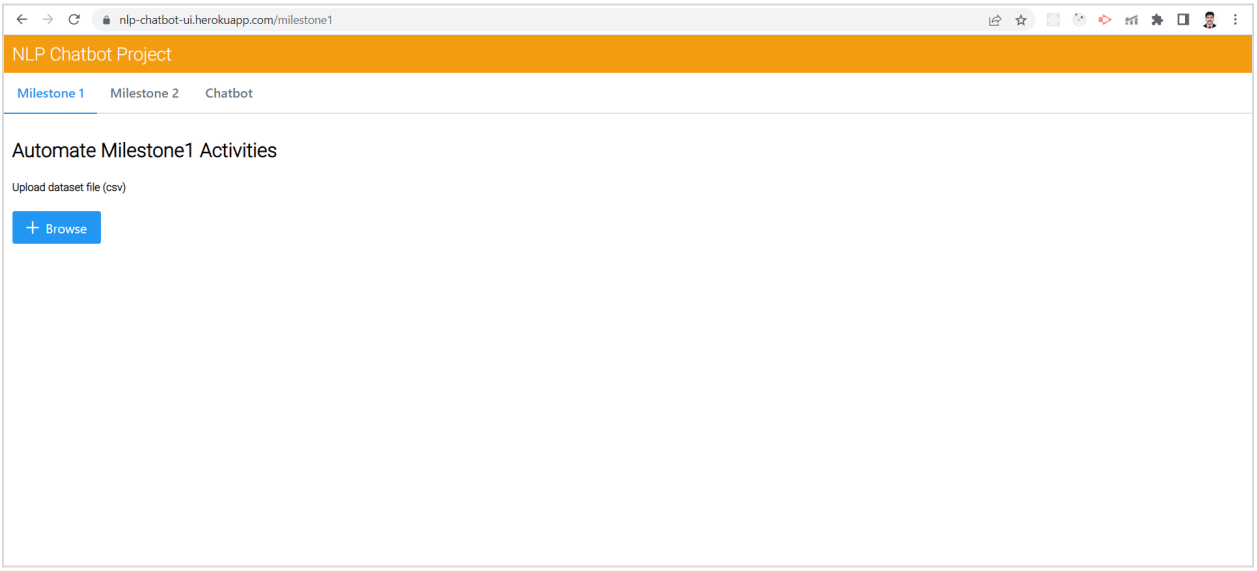
All of these models have a Test Accuracy of 78.57%. The next best performing model id **NN Model with Relu Activation** which having 77.38% accuracy.

Milestone 3

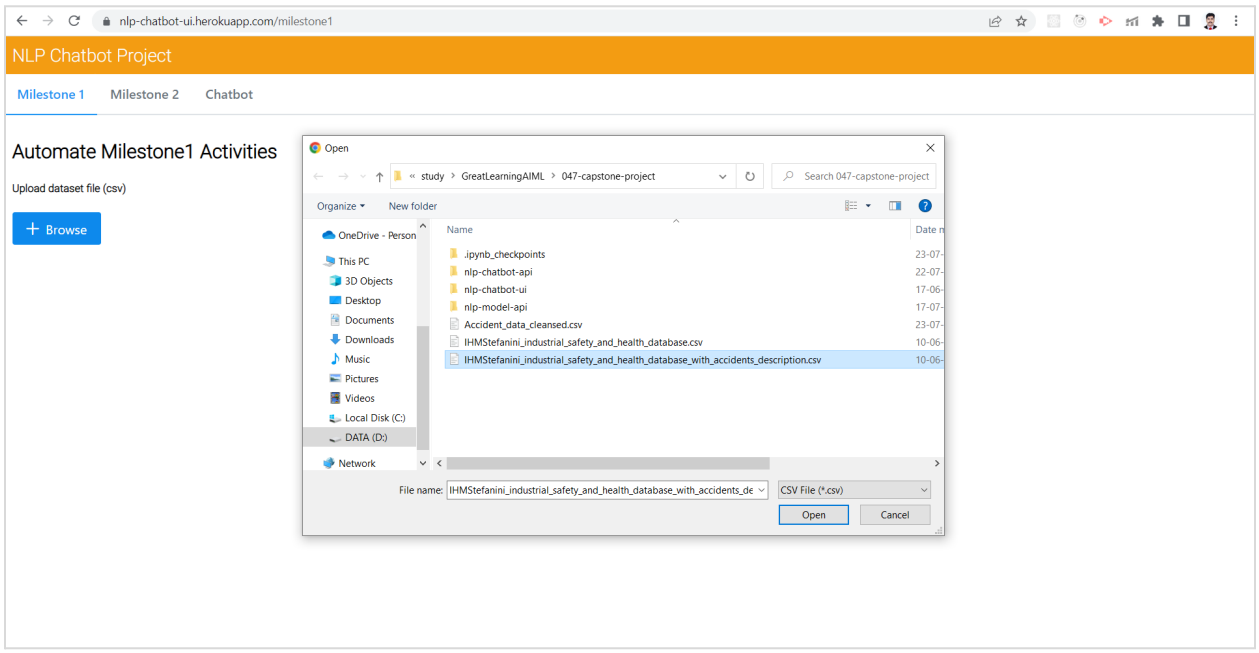
Automate Milestone1 Activities

We have automated Milestone 1 activity which is capable of performing various analysis and data cleansing tasks. The UI can be accessed from the URL: https://nlp-chatbot-ui.herokuapp.com/milestone1. Below are some of the screenshots with descriptions:

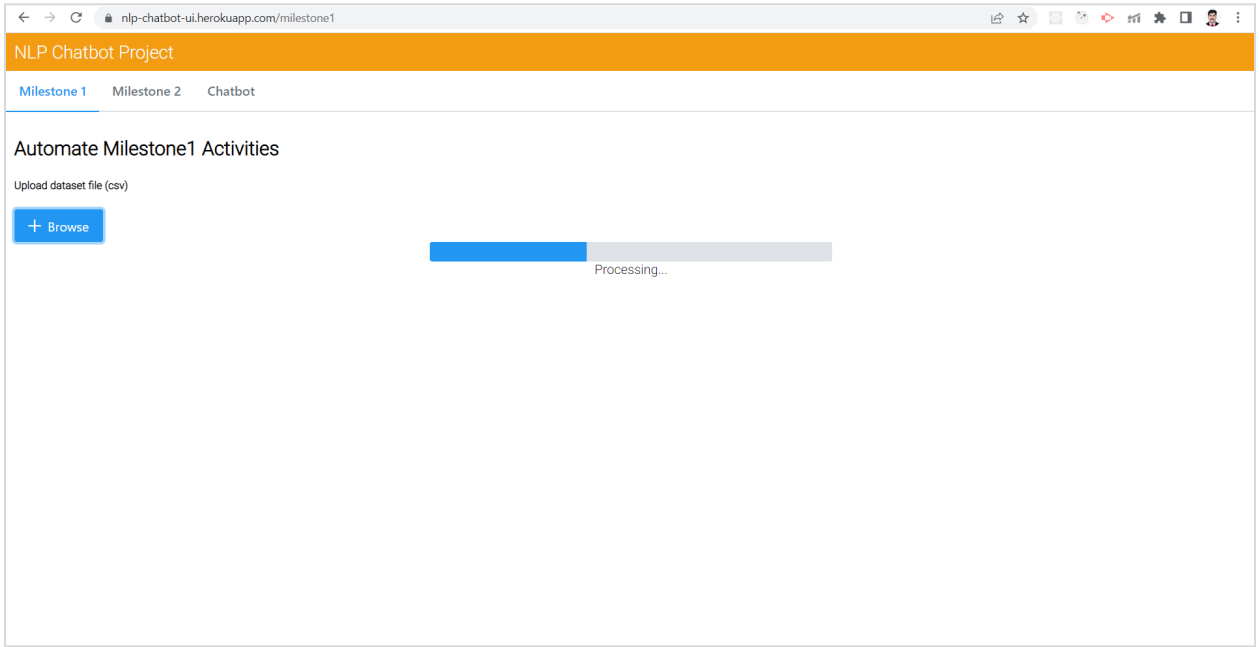
This is the initial UI of the Milestone1 automation where we have provided a **Browse** button to select the input dataset file.



Whenever the user clicks on the Browse button, it will open a file explorer window. The browse feature is smart enough to list and select only CSV files. Here it is recommended to upload the input CSV file with description or text data. Once the input file has been selected, the user can click on the Open button available on the file explorer window.

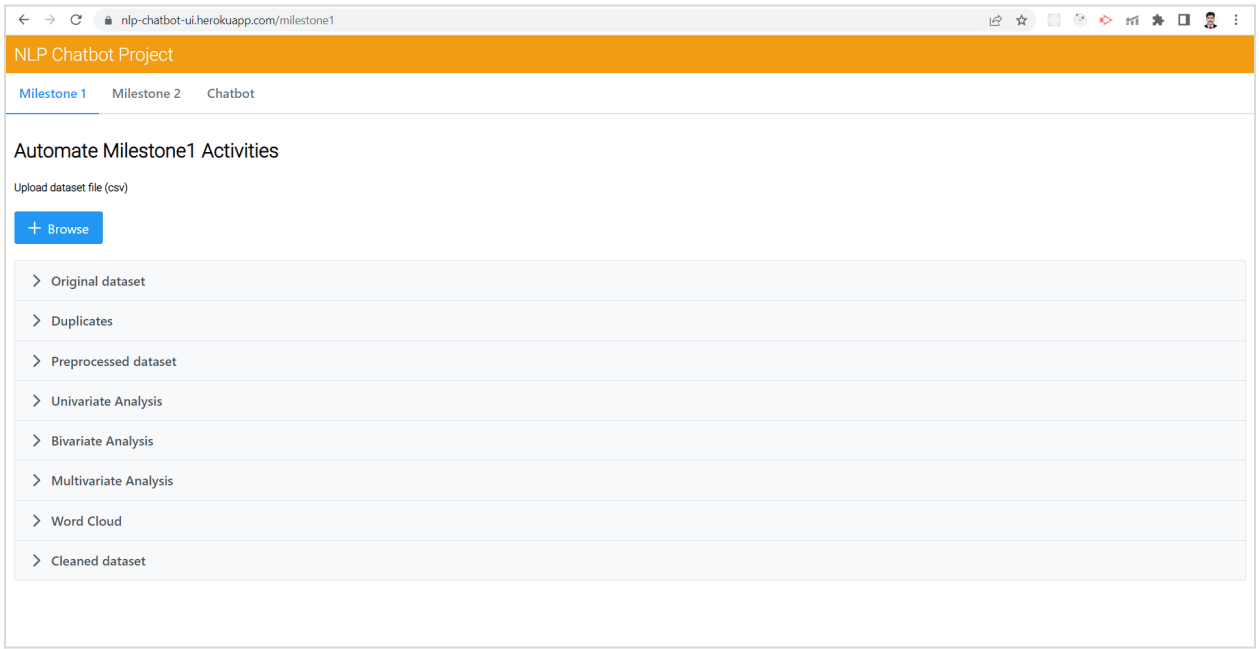


Once the user clicks on the Open button from the file explorer window, the file will get uploaded to the server and start processing it. This processing may take a few minutes since it is doing various data cleaning processes and analysis.

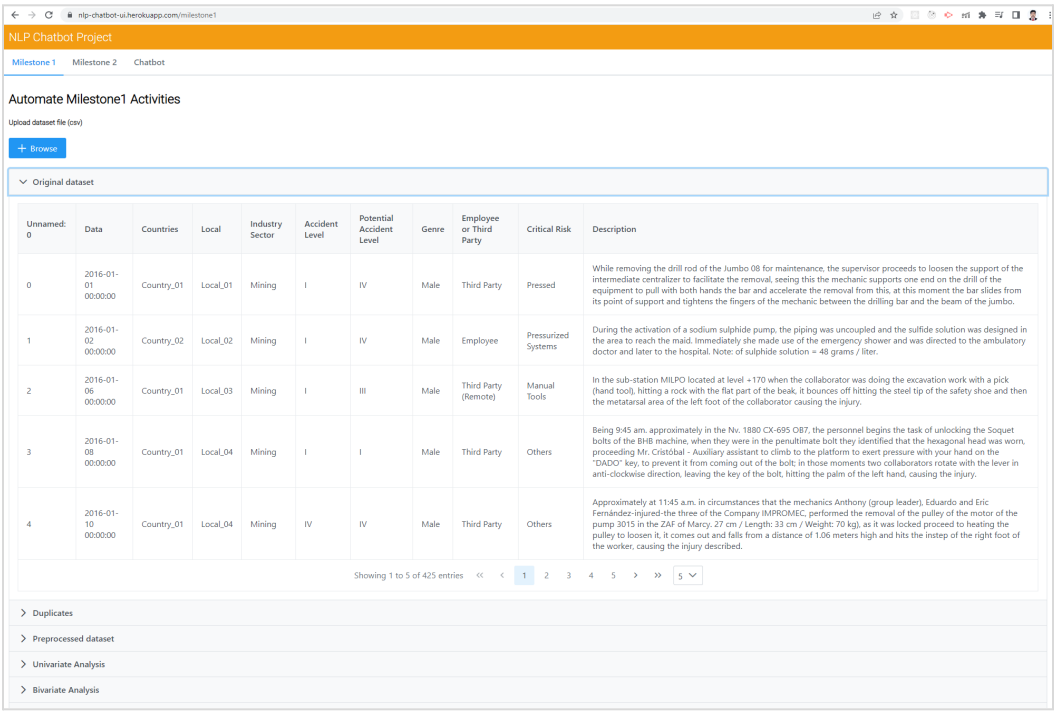


Once the processing has been completed, the user is able to see various result sets on the UI, which are:

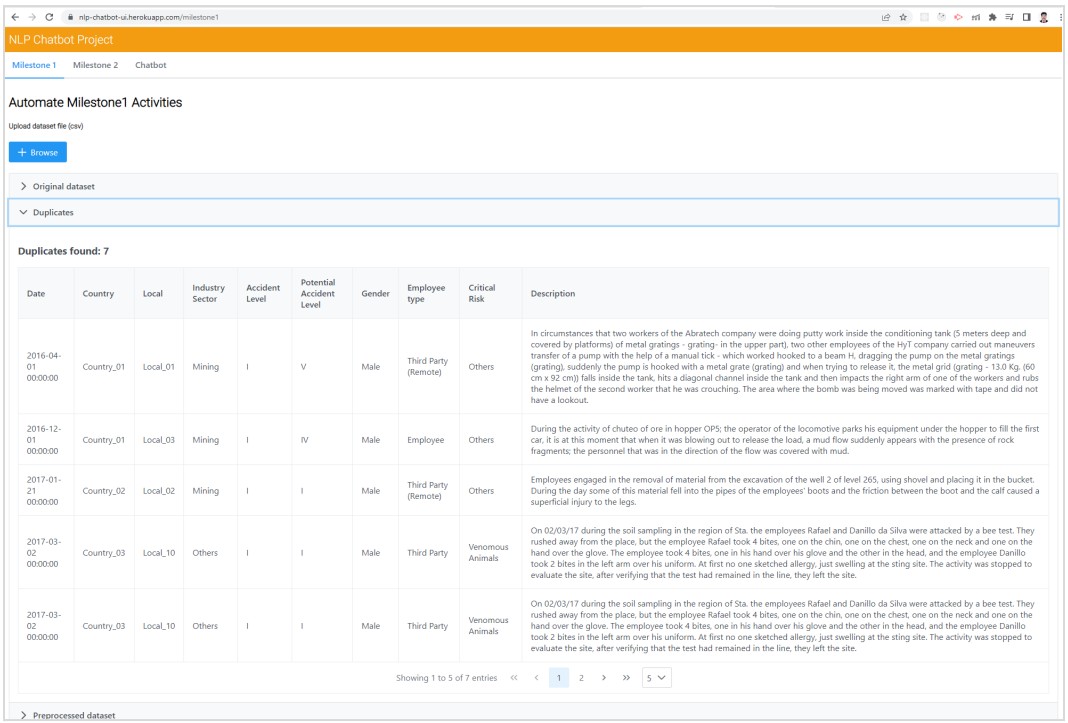
1. **Original dataset** - This shows the uploaded data in grid format with pagination.
2. **Duplicates** - Shows the duplicate records with its count.
3. **Preprocessed dataset** - Shows the preprocessed dataset prepared for various kinds of analysis like Univariate, Bivariate, Multivariate, Word Cloud etc.
4. **Univariate analysis** - Shows the visualizations generated as part of the Univariate analysis.
5. **Bivariate analysis** - Shows the visualizations generated as part of the Bivariate analysis.
6. **Multivariate analysis** - Shows the visualizations generated as part of the Multivariate analysis.
7. **Word cloud** - Shows the Word cloud visualization where the word size will be based on the number of occurrences.
8. **Clean Dataset** - Shows the Processed dataset which can be used for Milestone 2 activities.



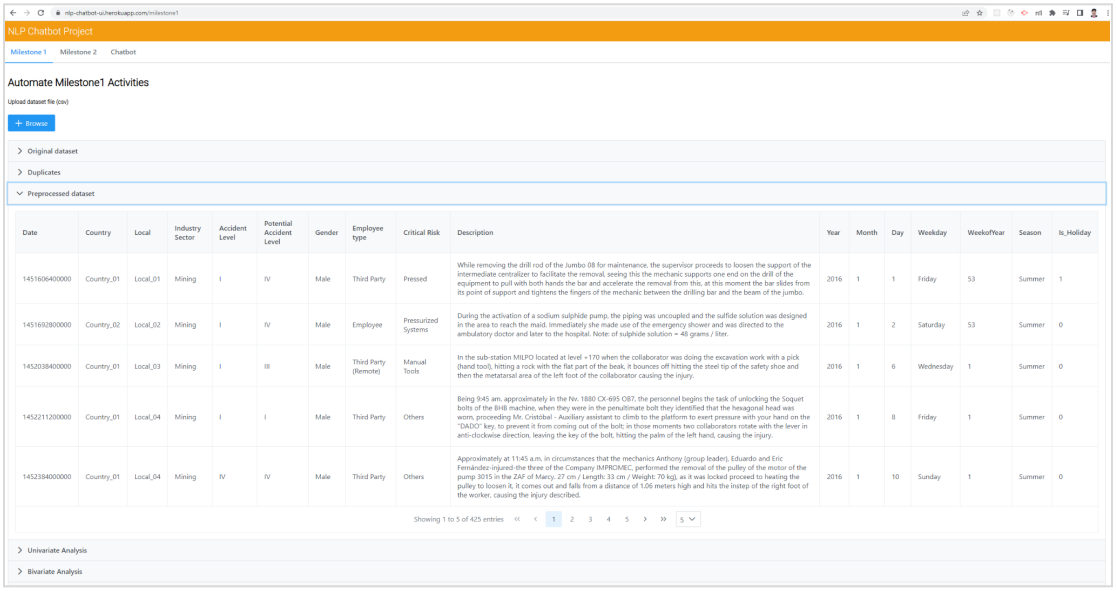
Original dataset



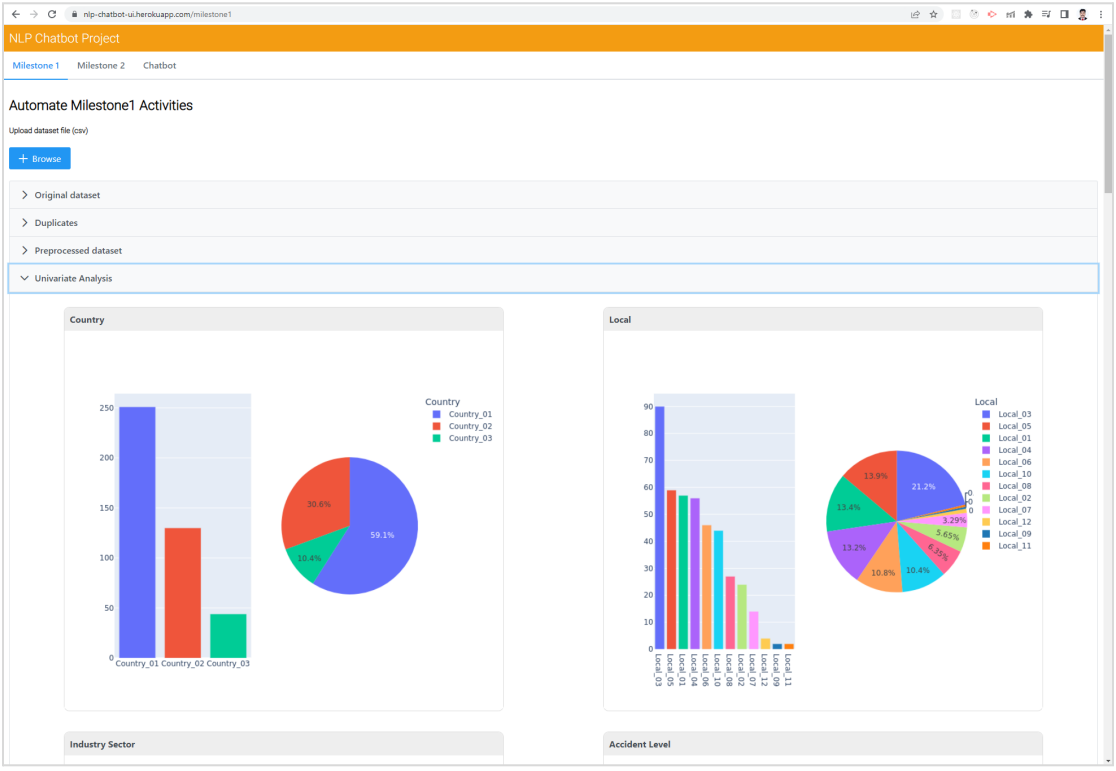
Duplicates



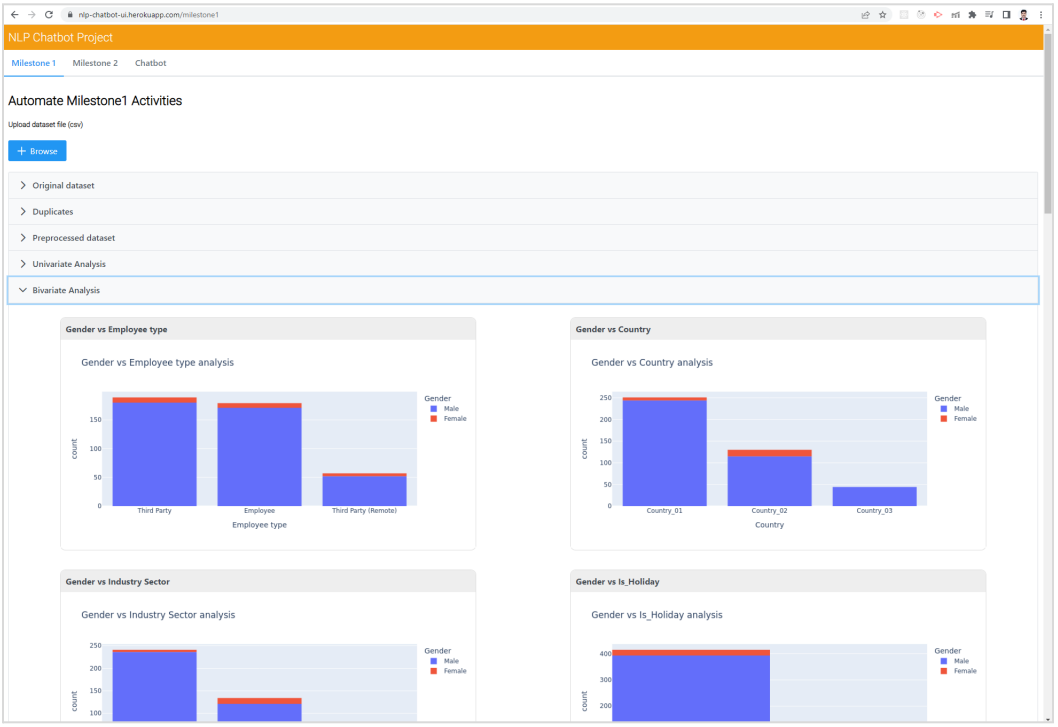
Preprocessed dataset



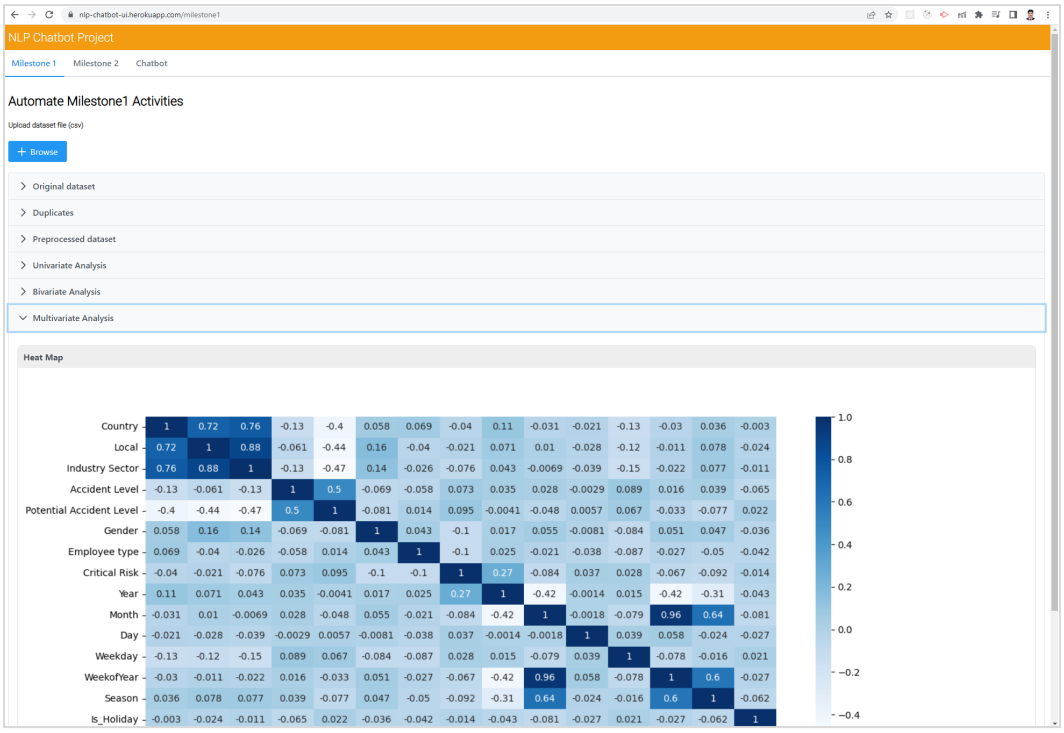
Univariate Analysis



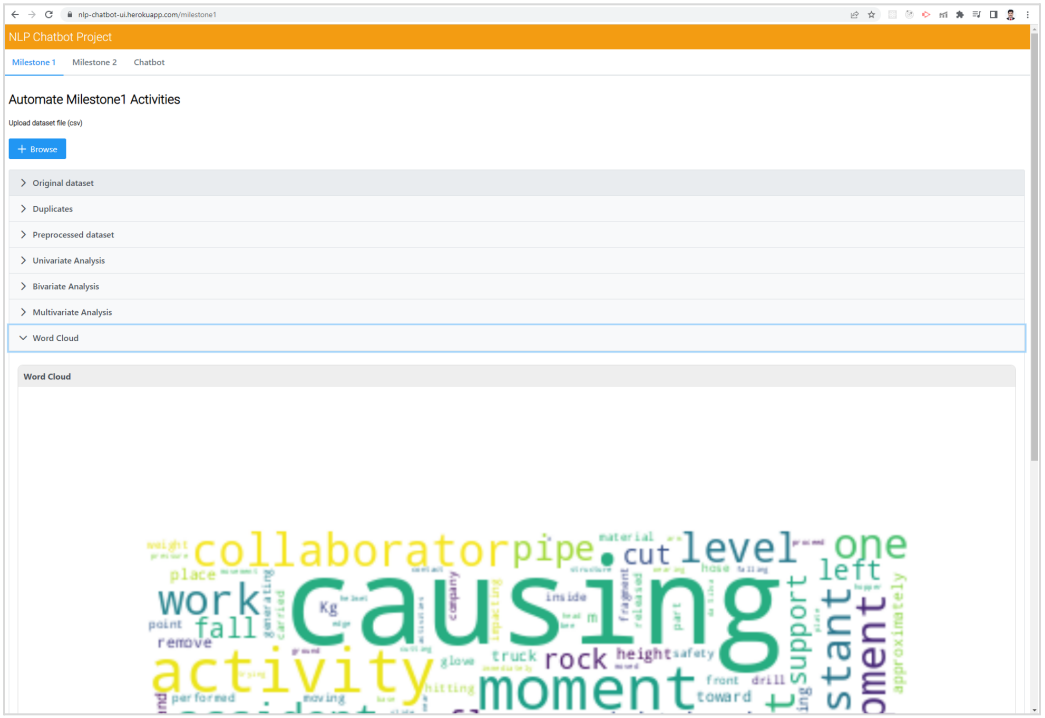
Bivariate Analysis



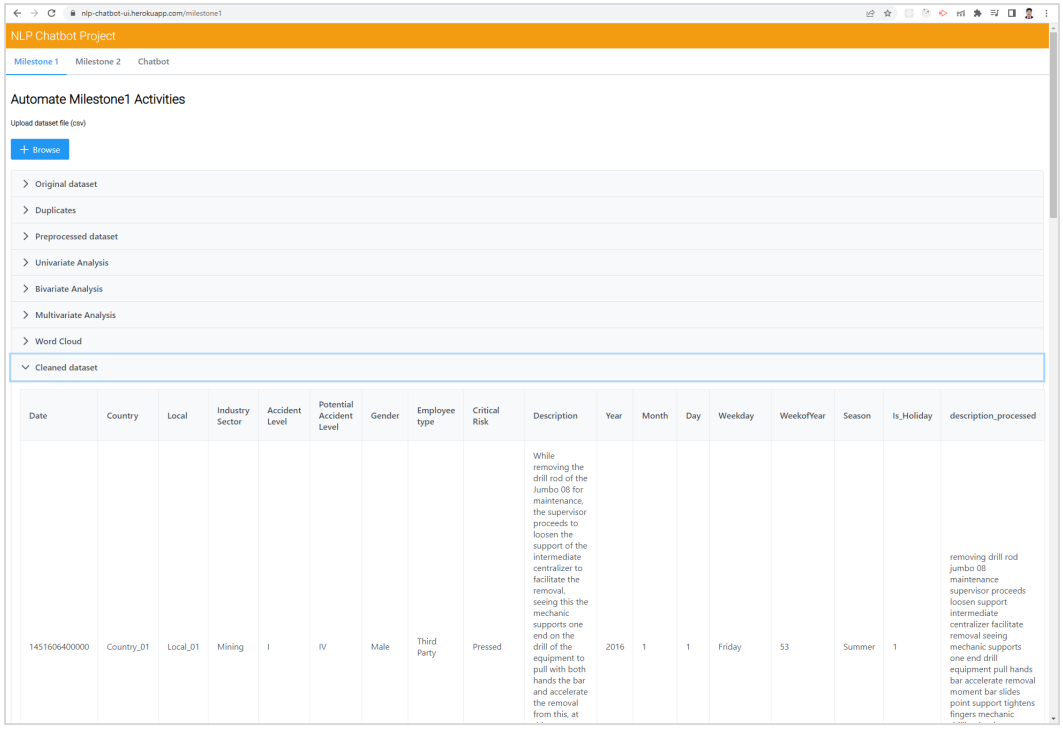
Multivariate Analysis



Word Cloud



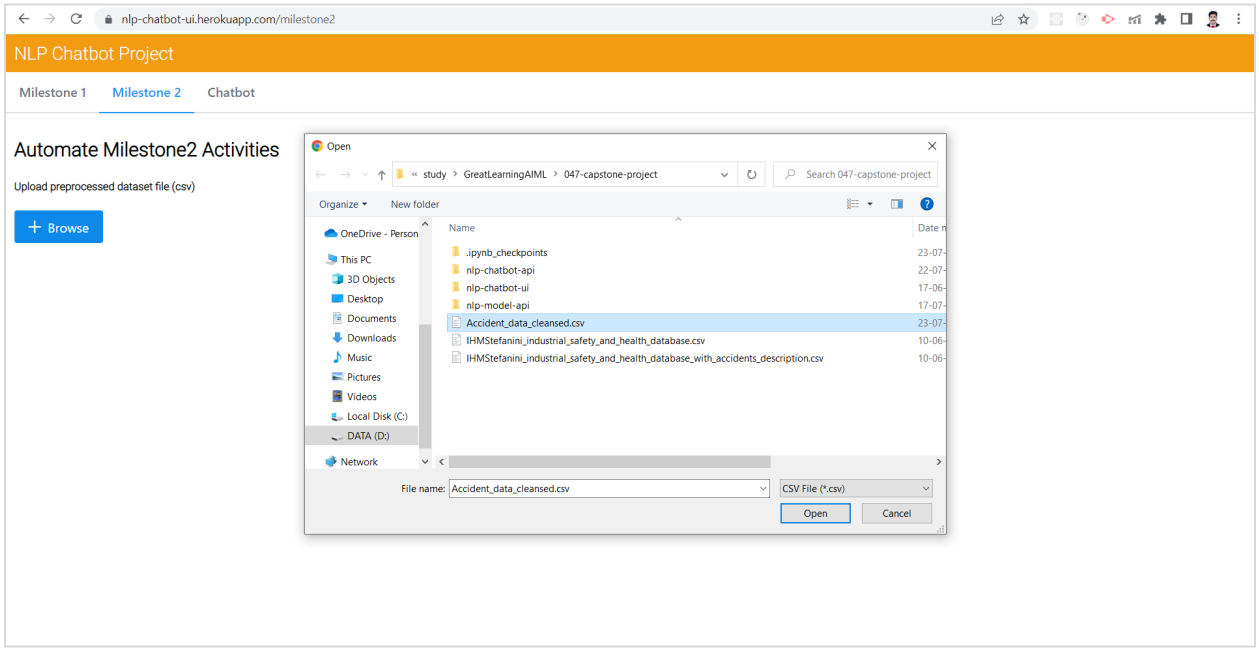
Cleaned Dataset



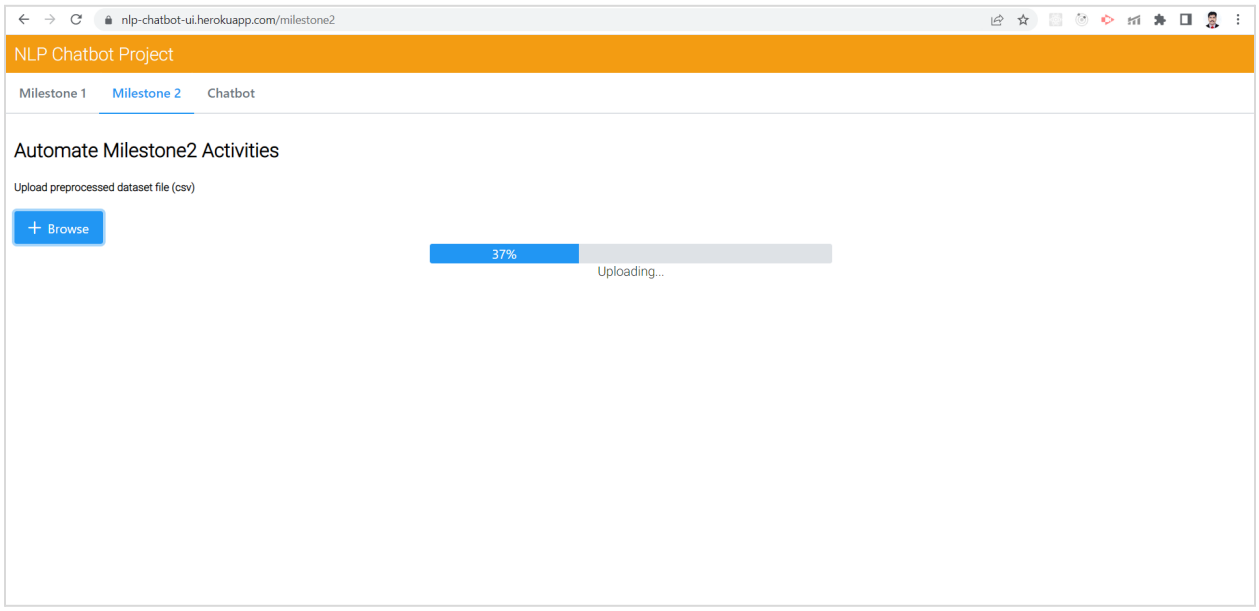
Automate Milestone 2 Activities

In Milestone 2, we again have an option to upload an ".csv" file. This file should be processed already. Idea is to use the data from the file and train different types of models. The chatbot UI can be accessed from the URL: https://nlp-chatbot-ui.herokuapp.com/milestone2

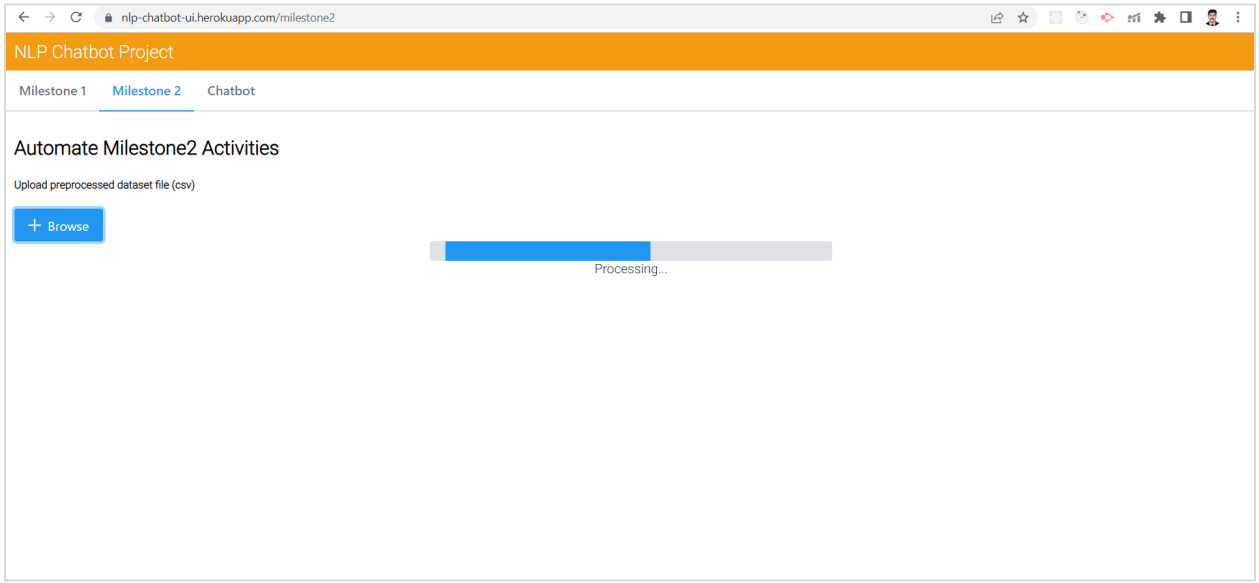
The below screen shows the file being selected for upload.



Once the user clicks on the Open button from the file explorer window, the file will get uploaded to the server and start processing it. This processing may take a few minutes since it is doing various model preparations.



File is being processed:

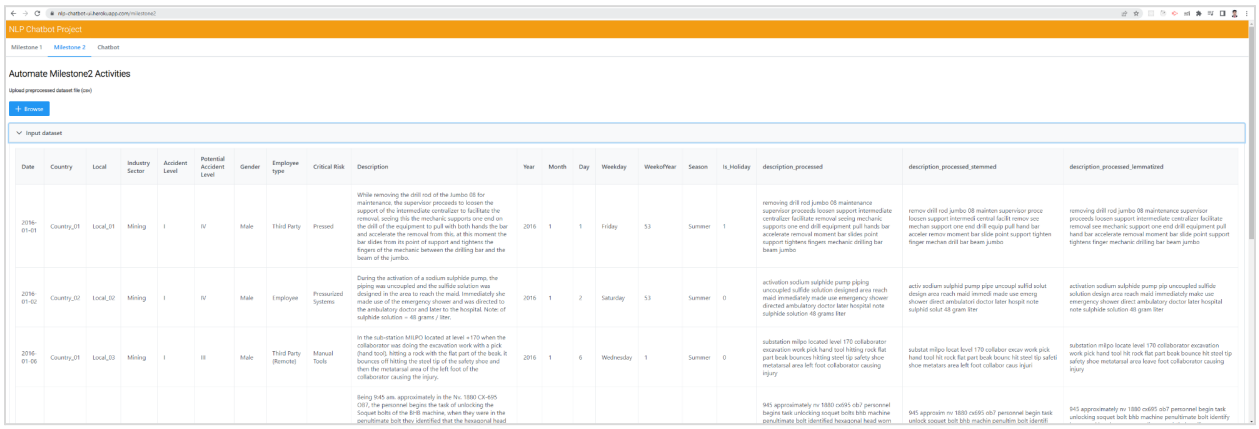


Once the processing has been completed, the user is able to see various result sets on the UI, which are:

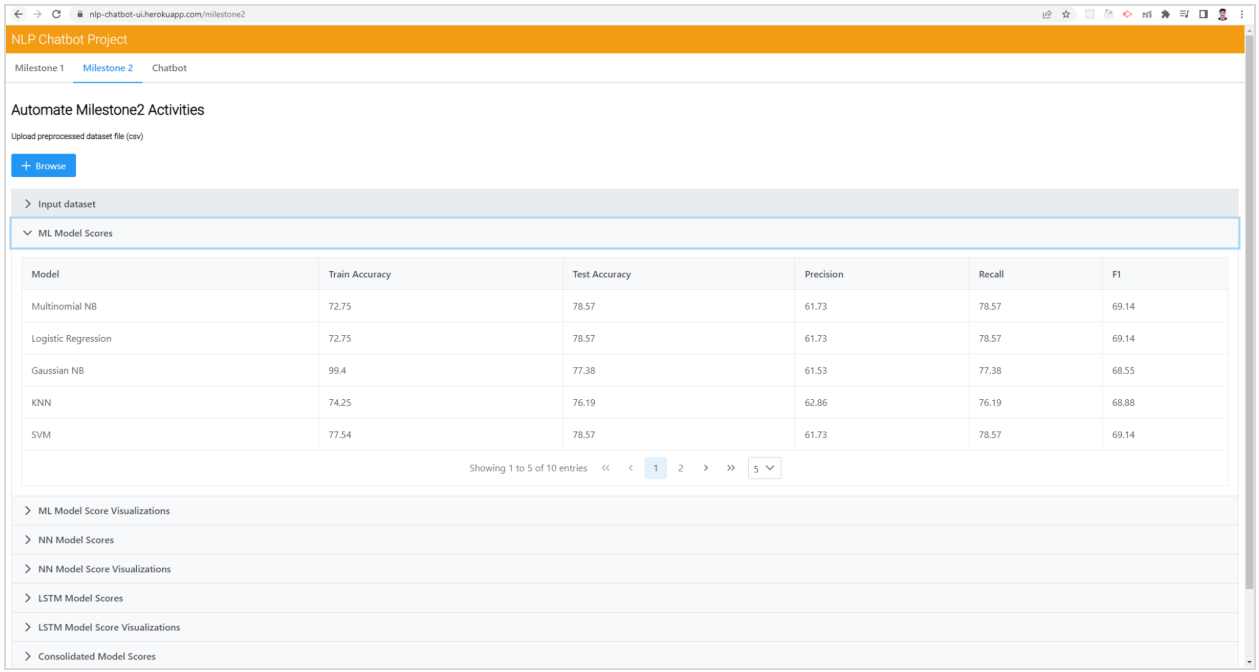
1. Input dataset - This shows the uploaded data in grid format with pagination.
2. ML Model Scores - Shows the Machine Learning model scores in grid format..
3. ML Molde Score Visualizations - Shows the visualizations generated using ML Model scores
4. NN Model Scores - Shows the Neural Networks model scores in grid format..
5. NN Molde Score Visualizations - Shows the visualizations generated using NN Model scores
6. LSTM Model Scores - Shows the LSTM model scores in grid format..
7. LSTM Molde Score Visualizations - Shows the visualizations generated using LSTM Model scores
8. Consolidated Model Scores - Shows the All the model scores in grid format..
9. Consolidated Molde Score Visualizations - Shows the visualizations generated using All the model scores
10. Best Performing Model - Show the best performing model from the list of prepared models.

Few of the result screens are listed below:

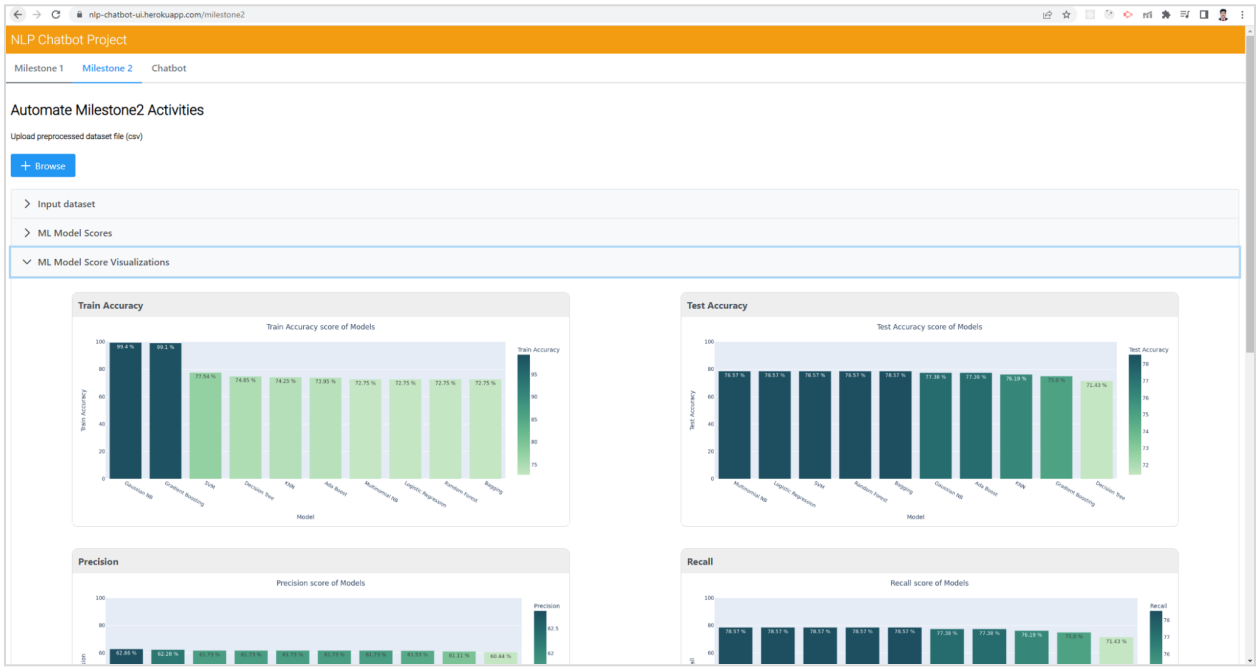
Input dataset



ML Model Scores



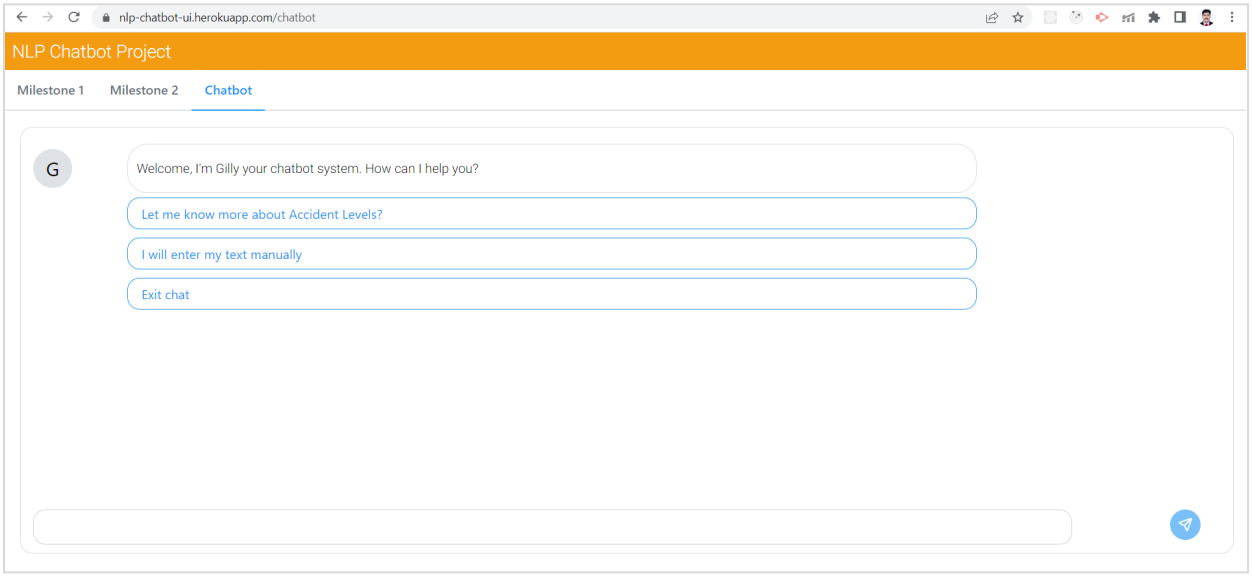
ML Model Score Visualizations



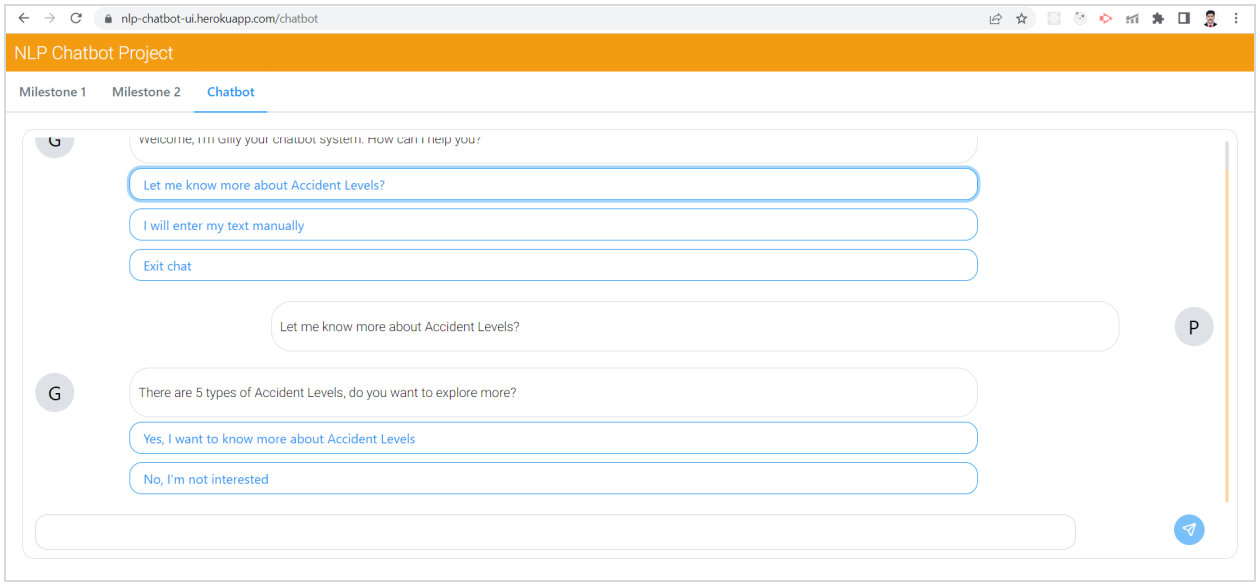
Milestone 3 : Chatbot

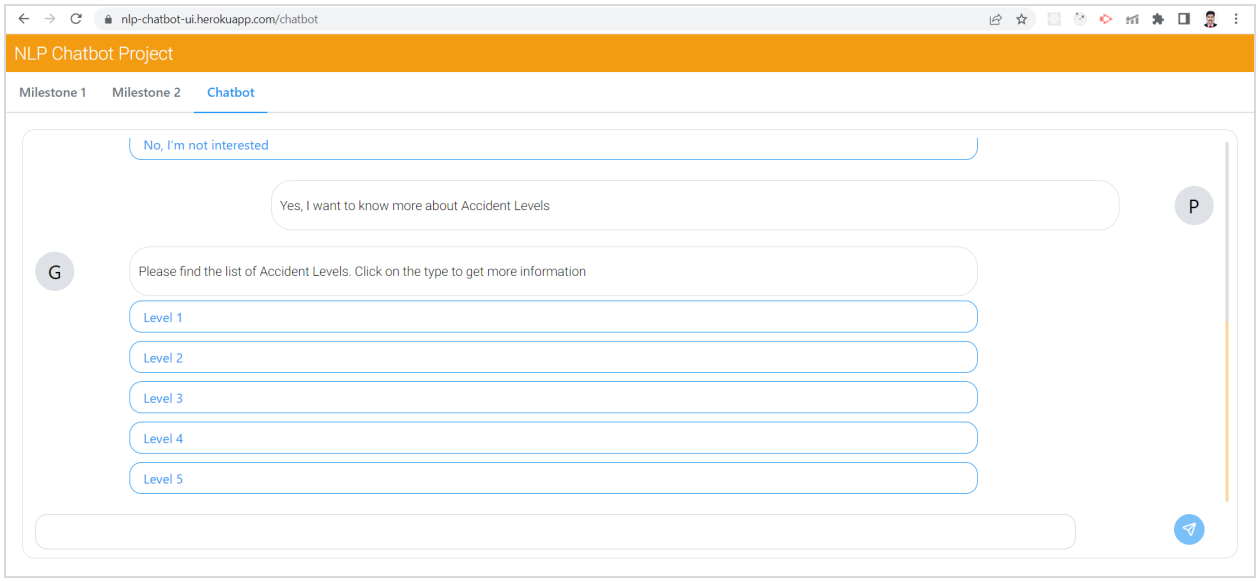
After designing various models using a balanced dataset, we picked the best performing mode and integrated it with our chatbot. As a result we have an intelligent chatbot named Gilly. The UI can be accessed from the URL: https://nlp-chatbot-ui.herokuapp.com/chatbot

Chatbot UI (Gilly)

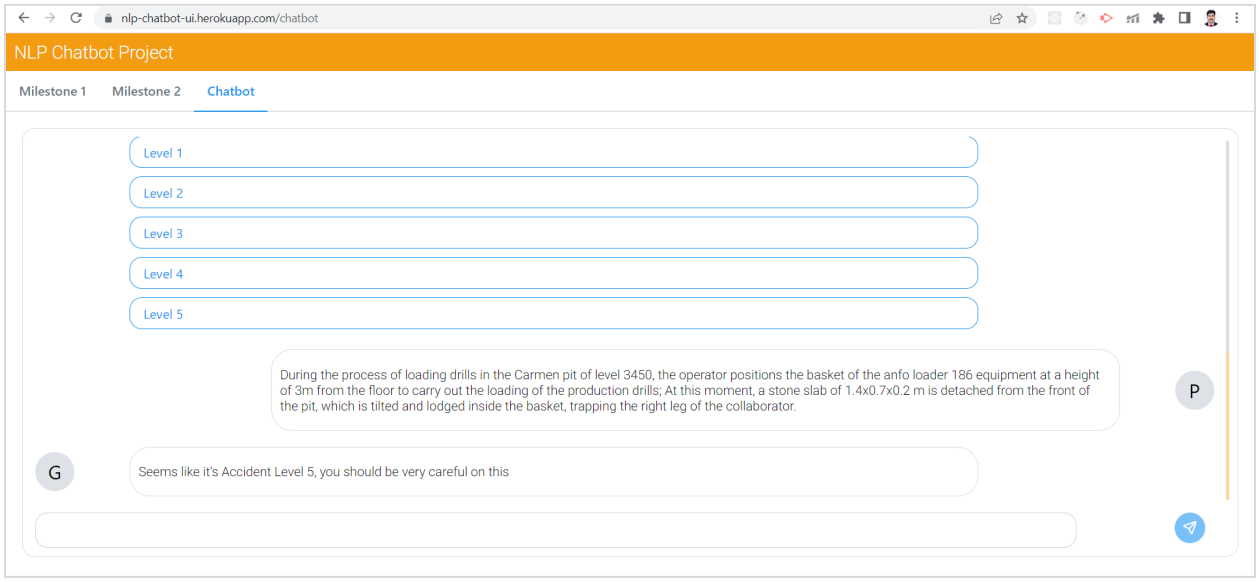


Gilly is smart enough to interpret what is being type or what is being selected from the context menu.

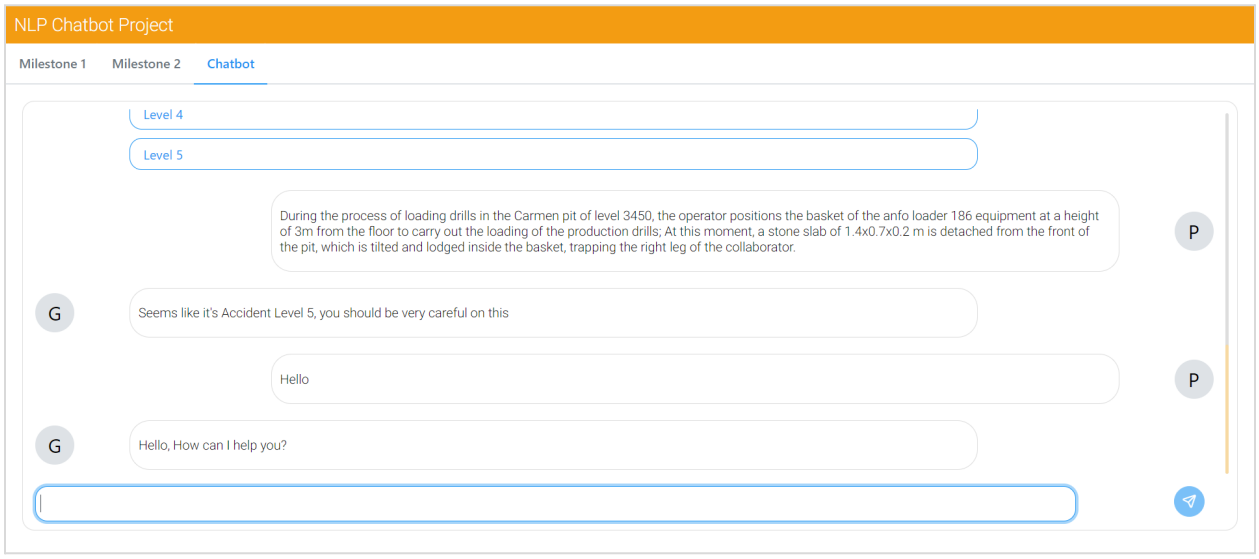




We added a description to test the capability of Gilly. She not only gives us accurate predictions but also urges us to be careful.



We have an option to end a chat or maybe continue. Gilly understands the context reset and responds accordingly.

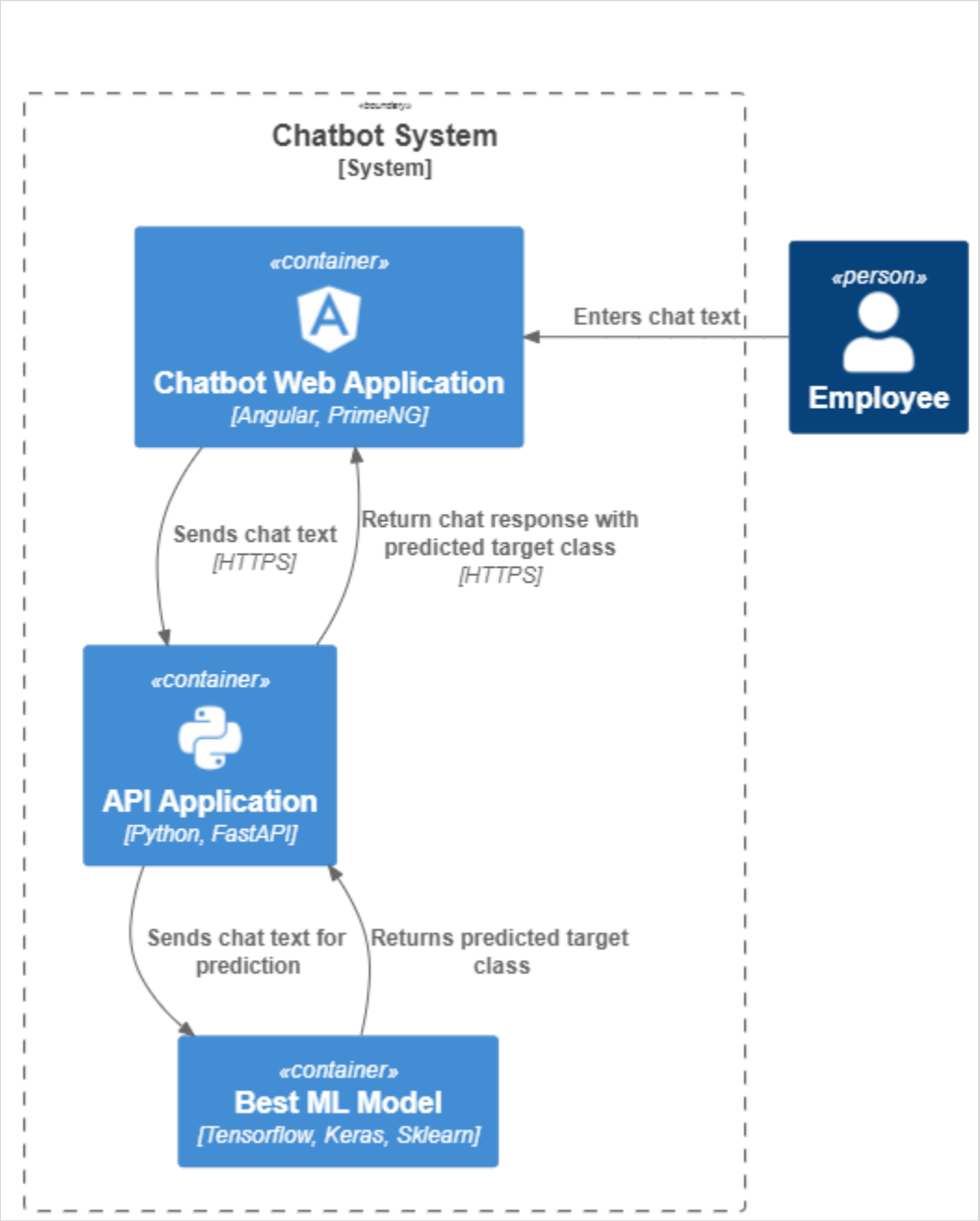


Conclusion:

After all the Milestone we have a working chatbot which works with reduced human intervention. In future we can even expand the capability to include SOS messages being passed on to health services based on the interaction with Gilly.

Technical Details

Context Diagram



The above context diagram displays the Chatbot system context and its related components. Each component is explained in detail below.

Employee

Here the employee is the end user who interacts with the Chatbot application through chat texts.

Chatbot Web Application

The chatbot web application is the user interface where the end user can enter their chat text and view the chat responses. Apart from entering some manual text, the chatbot application is capable of populating some predefined questions and answers, then the end user can simply select one of the populated options. This Web application has been developed using Angular framework and PrimeNG components library.

API Application

The API application will act as a bridge between the ML model and the Web application. Whatever text message the end user sends from the Web application will reach the API application through HTTPs protocol. The text messages then passed to the ML model and responded back to the Web application with the predicted target class value. The API application has been developed using Python and FastAPI.

Best ML Model

The best ML model is the pickle file created by the ML team which will be used to predict the target class over the chat text. The model will be returning only the predicted target class.

Application Deployment

Github

Github is a distributed version control and source code management system where all the allocated developers can work on the source code. Here we have maintained all our application codes on different Github repositories.

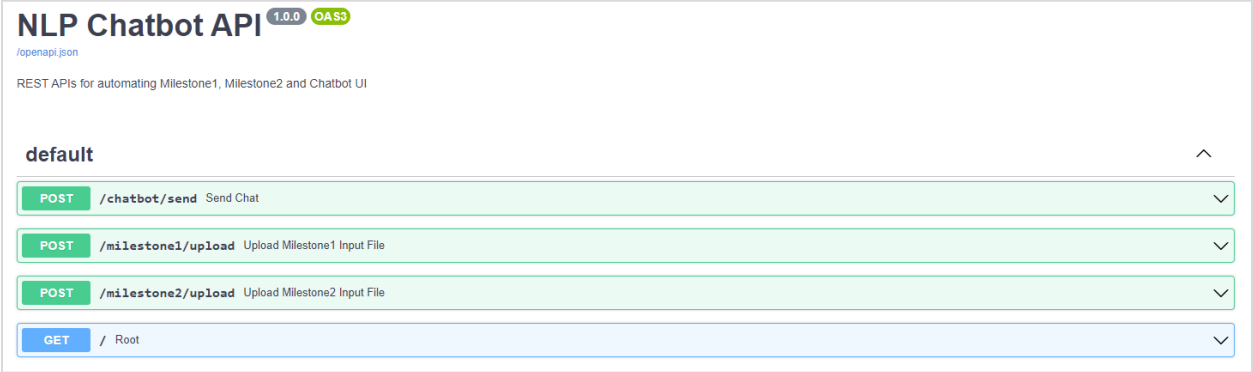
Heroku

Heroku is a cloud application platform where we can build and deploy our application codes. We have deployed our frontend web application and API application on Heroku.

CICD Pipeline

Nowadays most people don't prefer manual deployment of applications, instead they use CICD Pipelines. CICD stands for Continuous Integration and Continuous Delivery/Deployment. Heroru provides a CICD pipeline provision on their system where the application source code can be linked from a Github repository. Whenever a new code change is pushed on to the Github repository, the CICD pipeline will initiate the build and deployment process automatically without any manual interventions. This helps the developers a lot and the new code changes will be available on the hosted application URLs within minutes.

API Documentation



As part of the application, we have developed 3 major APIs and 1 root API which can be used as an health check endpoint to make sure the API is up and running. The other 3 APIs are listed below:

1. Upload Milestone1 Input file (/milestone1/upload)

   This API has been used for uploading the milestone 1 file from the frontend application.

2. Upload Milestone2 Input file (/milestone2/upload)

   This API has been used for uploading the milestone 2 file from the frontend application.

3. Send Chat (/chatbot/send)

   This API has been used for sending the chat messages from the frontend application.

Conclusion

1. Gaussian Naive Bayes and Support Vector Machine algorithms performed well in test accuracy on text classification of accident data comparatively to bidirectional LSTM. Hence, the application of traditional ML classification algorithms are better on the data extracted from real world situations.

2. Insights performed on exploratory data analysis show the mining industry has serious occupational accidents. Industrial engineers can design safer workplaces at mining sites by consulting with tech companies and Data Scientists.

Future prospects

To get more insights on this accident data, application of Retrieval Enhanced TRansfOrmers (RETRO) might be a good idea as the LSTM models were not effective in gaining the test accuracy. The RETRO model has access to the entire dataset while training using a retrieval mechanism. Comparing this to a normal Transformer with the same number of parameters yields notable performance gains.

From the Frontend application perspective we can accept voice commands from the end user and respond to it with both voice and text messages.