FINAL REPORT

AI Enabled IT Ticketing Service Tool

Submitted By

Disha Palan

Parita Desai

Gloria Preet

Table of Contents

Summary of problem statement, data and findings

Problem Statement

In any of the IT industry, incident management plays an important role in delivering quality and timely support to its customers across the globe.

The incidents are generally created by various stakeholders like end users, vendors, IT users, etc. They might not have right information as to which team the ticket should go to. Hence, to improve and retain customer satisfaction, it is very important that the ticket is assigned to the right group of people for faster and appropriate resolution. In many Organizations this is still a manual process. There are few problems with the manual process:

1. Manual assignment of incidents is time consuming
2. It requires human efforts
3. There may be mistakes due to human errors and resource consumption is carried out ineffectively because of the misaddressing
4. Manual assignment increases the response and resolution times which result in user satisfaction deterioration / poor customer service

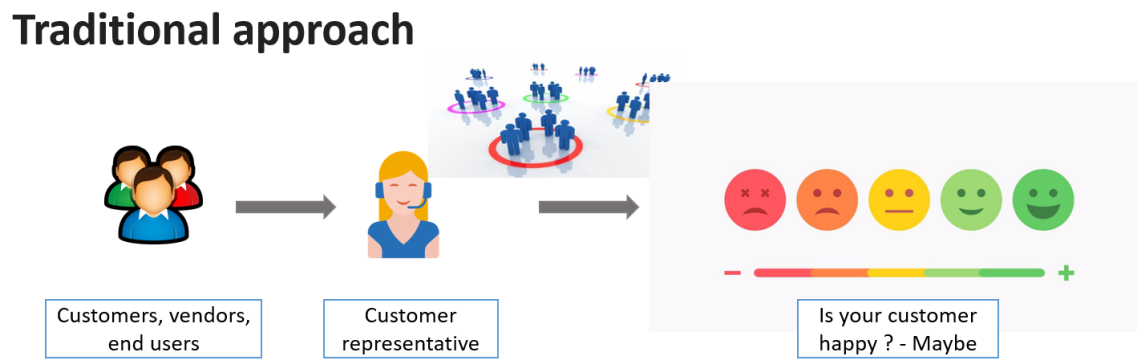
L1 / L2 needs to spend time to review Standard Operating Procedures (SOPs) before assigning to Functional teams (Minimum 25–30% of incidents needs to be reviewed for SOPs before ticket assignment).

15 mins are being spent for SOP review for each incident. Minimum of 1 FTE effort needed only for incident assignment to L3 teams.

During the process of incident assignments by L1 / L2 teams to functional groups, there were multiple instances of incidents getting assigned to wrong functional groups.

Around 25% of Incidents are wrongly assigned to functional teams. Additional effort needed for Functional teams to re-assign to right functional groups

During this process, some of the incidents are in queue and not addressed timely resulting in poor customer service and loss of business.

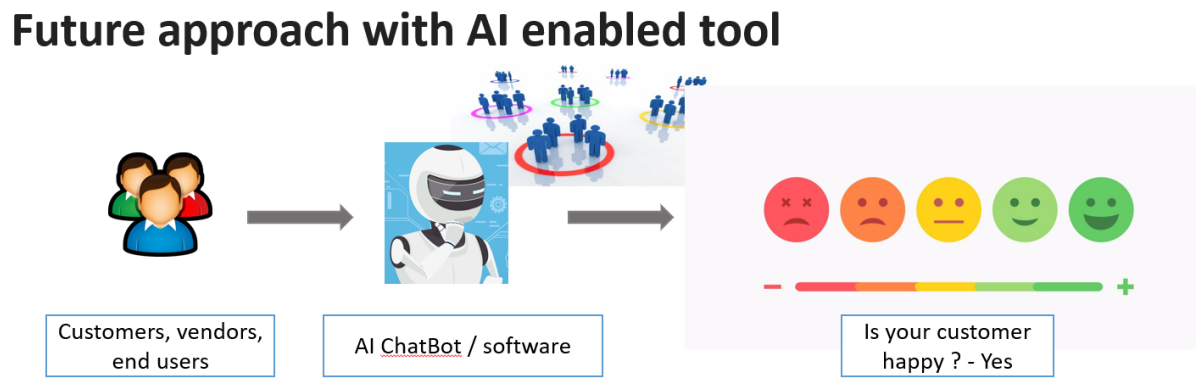


Objective

We are building an AI solution which will enable organizations to classify incidents to the right functional group by implementing the best suited machine learning model and leading to customer satisfaction.

Guided by AI, organizations can reduce the resolution time and focus on more productive tasks. This will overcome and save time with below losses:

1. Time latency due to review of SOPs before assigning to right functional group
2. Incorrect assignments to functional groups



Solution

Below approach is taken to determine the assignment groups based on the ticket description and short description.

- Exploratory data analysis
- Visualized different patterns in the dataset and performed further analysis
- Performed target column i.e. assignment group analysis
- Dealt with inconsistencies like treating the missing values and merging the short description and long description to determine the assignment group for ticket
- Text pre-processing done by fixing the encoding, detecting different languages and translating to English
- Performed data cleaning by removing stop words, lemmatization and creating tokens
- Performing Named Entity recognition and find POS tags for description
- Modelling : Modelling was performed in two rounds:
    - Round I - Before treating class imbalance : Below Models are employed :

Classifier	Accuracy	F1-Score	Precision	Recall	Time taken
MultinomialNB Classifier - Round 1	0.612002	0.703090	0.533347	0.612002	0.676971s
SVC Classifier - Round 1	0.666667	0.707262	0.643468	0.666667	1.527838s
KNN Classifier - Round 1	0.525253	0.495434	0.688724	0.525253	86.517506s
SGD Classifier - Round 1	0.620915	0.705889	0.565885	0.620915	0.542858s
Random Forest Classifier - Round 1	0.647059	0.717783	0.601619	0.647059	8.965739s
XgBoost Classifier - Round 1	0.654783	0.705976	0.622069	0.654783	181.355291s

- Round II - After treating class imbalance : Below Models are employed :

Evaluation Metrics for Traditional models: -

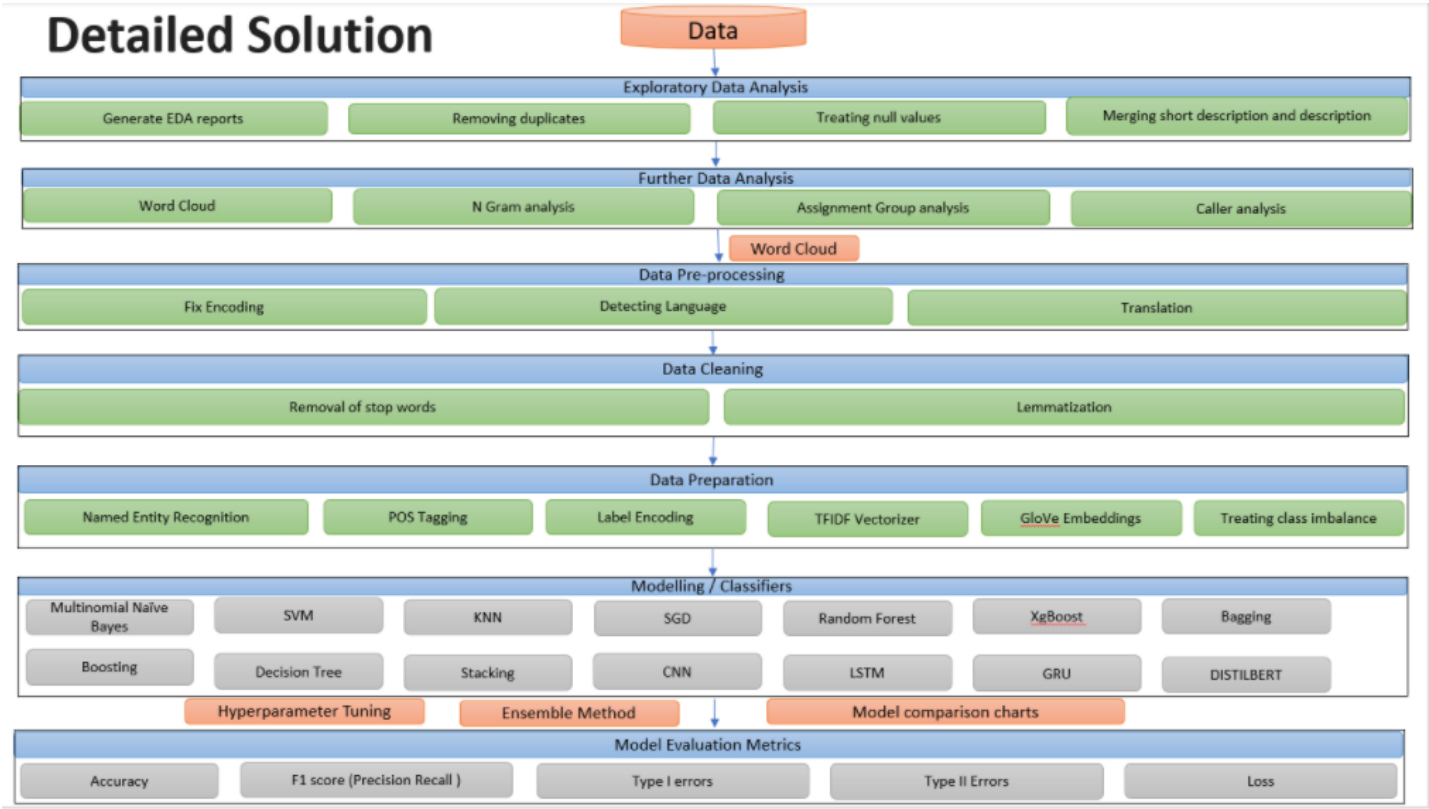
Classifier	Accuracy	F1-Score	Precision	Recall	Time taken
MultinomialNB Classifier - Round 2	0.902472	0.902598	0.921811	0.902472	3.713569s
SVC Classifier - Round 2	0.948670	0.946304	0.965018	0.948670	7.533591s
SGD Classifier - Round 2	0.869265	0.879354	0.891305	0.869265	3.873436s
Random Forest Classifier - Round 2	0.957783	0.954622	0.975372	0.957783	42.754390s
XgBoost Classifier - Round 2	0.956631	0.953569	0.974189	0.956631	856.407620s
Bagging Classifier - Round 2	0.952965	0.950734	0.970131	0.952965	54.931186s
Boosting Classifier - Round 2	0.836371	0.826056	0.884197	0.836371	1401.513942s
DecisionTree Classifier - Round 2	0.953384	0.950845	0.970456	0.953384	7.126146s

Evaluation Metrics for Sequential models: -

Classifier	Accuracy	F1-Score	Precision	Recall	Time taken
LSTM	0.920386	0.918289	0.934078	0.920386	3576.689095s
CNN - Round 2	0.935156	0.932964	0.953313	0.935156	238.385721s
GRU	0.943956	0.940658	0.963943	0.943956	5117.891881s

- Hyperparameter Tuning was done for above models
- Transfer learning is employed with DISTILBERT with LinearSVC with resampled data
- DistilBERT, used to create the features and labels using pretrained model gives an accuracy of 92.82%. This can be further improved by fine tuning

Based on above Evaluation metrics we select either Random Forest as this gives accuracy of 95.77% and takes much lesser time compared to other models with almost similar accuracy and also the precision and recall value are also the highest



Data

References : https://drive.google.com/open?id=1OZNJm81JXucV3HmZroMq6qCT2m7ez7IJ

The given dataset has below four columns:

1. Short description
2. Description
3. Caller
4. Assignment group

Out of above four columns we have 3 features namely, short description, description and caller and one target group namely assignment group

Top 10 records from the dataset:

	Short description	Description	Caller	Assignment group
0	login issue	-verified user details.(employee# & manager na...	spxjnwir pjlcoqds	GRP_0
1	outlook	\n\nreceived from: hmjdrvpb.komuaywn@gmail.com...	hmjdrvpb komuaywn	GRP_0
2	cant log in to vpn	\n\nreceived from: eylqgodm.ybqkwiam@gmail.com...	eylqgodm ybqkwiam	GRP_0
3	unable to access hr_tool page	unable to access hr_tool page	xbkucsvz gcpydteq	GRP_0
4	skype error	skype error	owlgqjme qhcozdfx	GRP_0
5	unable to log in to engineering tool and skype	unable to log in to engineering tool and skype	eflahbxn ltdgrvkz	GRP_0
6	event: critical:HostName_221.company.com the v...	event: critical:HostName_221.company.com the v...	jyoqwxhz clhxsoqy	GRP_1
7	ticket_no1550391- employment status - new non-...	ticket_no1550391- employment status - new non-...	eqzibjhw ymebpoih	GRP_0
8	unable to disable add ins on outlook	unable to disable add ins on outlook	mdbegvct dbvichlg	GRP_0
9	ticket update on inplant_874773	ticket update on inplant_874773	fumkcsji sarmtlhy	GRP_0

Last 5 records from the dataset:

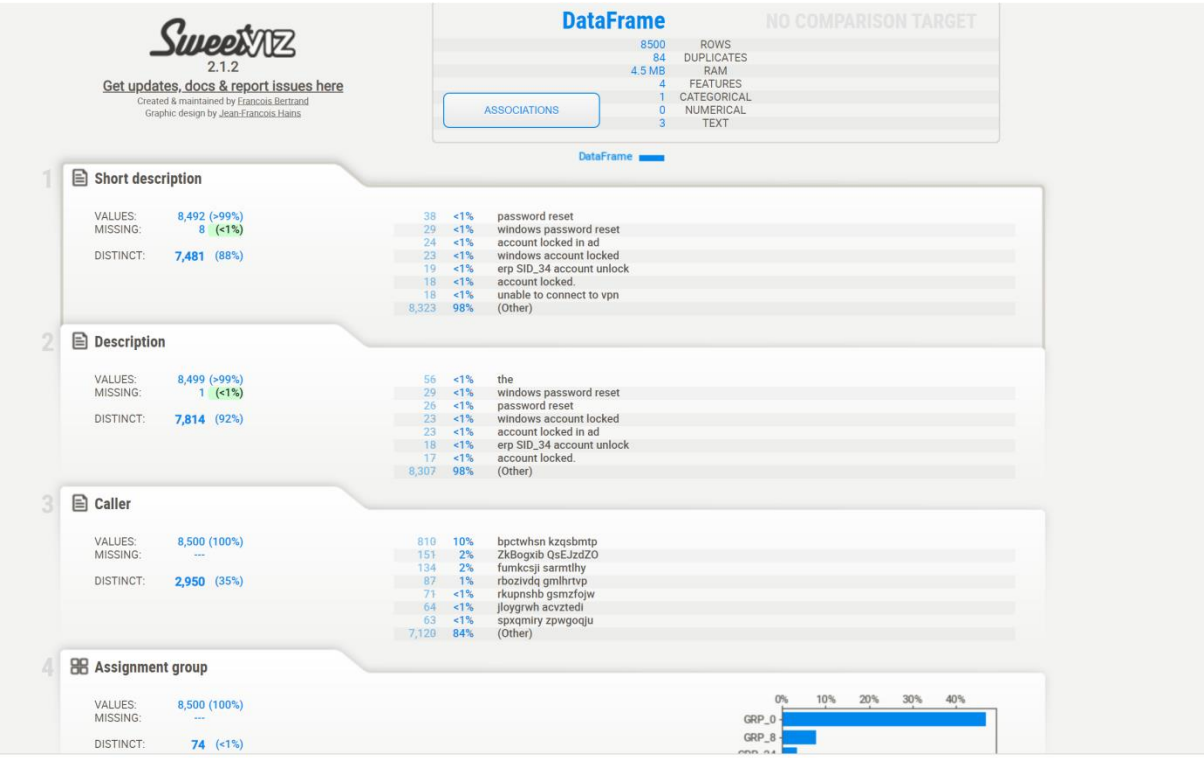
	Short description	Description	Caller	Assignment group
8495	emails not coming in from zz mail	\n\nreceived from: avglmrts.vhqmtiua@gmail.com...	avglmrts vhqmtiua	GRP_29
8496	telephony_software issue	telephony_software issue	rbozivdq gmlhrtvp	GRP_0
8497	vip2: windows password reset for tifpdchb pedx...	vip2: windows password reset for tifpdchb pedx...	oybwdsgx oxyhwrfz	GRP_0
8498	machine nÃ£o estÃ¡ funcionando	i am unable to access the machine utilities to...	ufawcgob aowhxjky	GRP_62
8499	an mehreren pc`s lassen sich verschiedene prgr...	an mehreren pc`s lassen sich verschiedene prgr...	kqvbrspl jyzoklfx	GRP_49

Summary of the Approach to EDA and Pre-processing

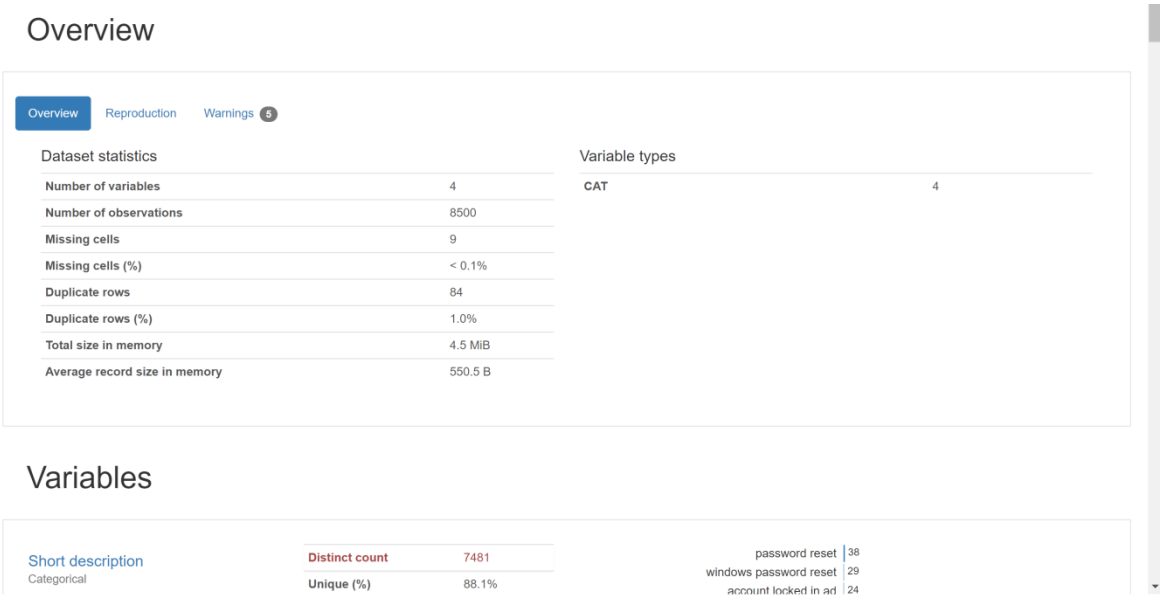
EDA

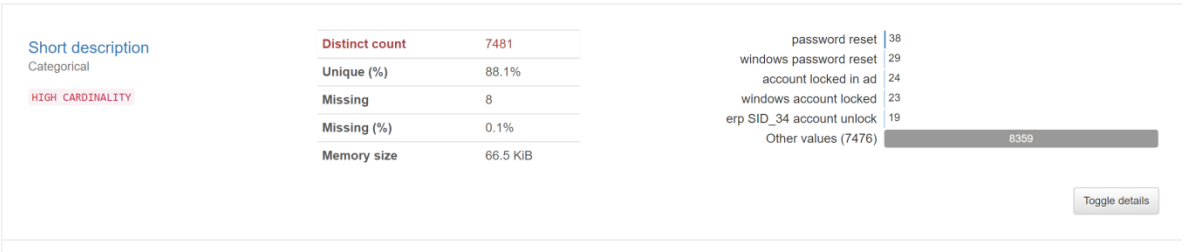
We have used SweetViz and Panda profiling to Visualize and analyse our dataset. Below are attached reports for and observations from reports are in next sections. The reports can be viewed in GitHub link - https://github.com/dishapalan02/AI-Enabled-IT-Service-Ticketing-tool/tree/main

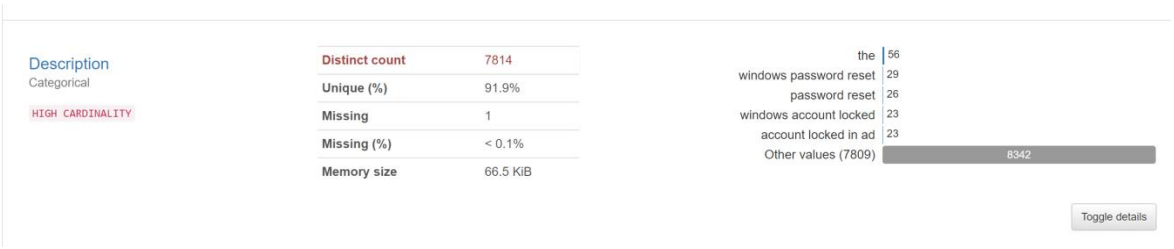
SweetViz:

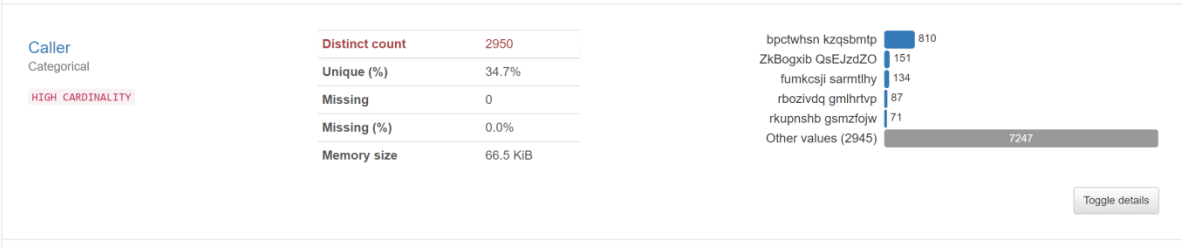


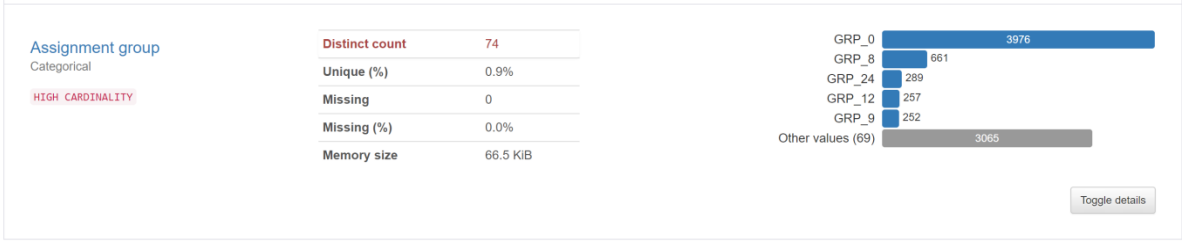
Pandas Profiling:

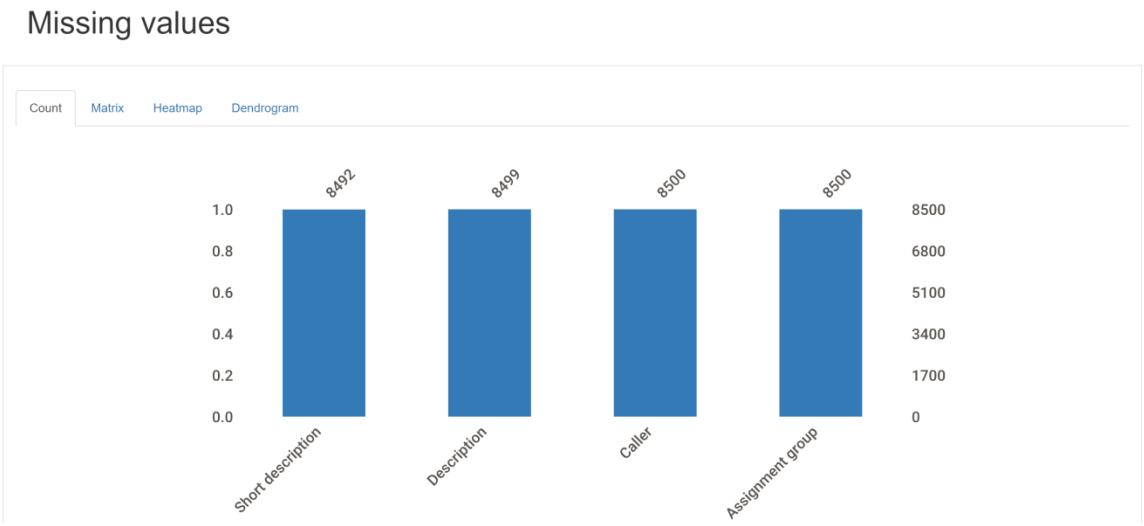












Findings

From Above two reports we have below observations:

1. Shape of the data - { Rows : 8500, Columns : 4 }
2. Total features - 3

   2.1. Short Description - Text

   2.2. Description - Text

   2.3. Caller - Text
3. Target Column - 1

   3.1 Assignment Group - Categorical
4. There are 84 duplicate records in total. Strategy to handle duplicates and the approach taken is defined in the pre-processing section below.
5. New features are required or not needs to be analysed further and also to check if below hidden patterns can be figured out:

   A. Common Issues -> user can be trained if possible
   B. Common Caller -> May be user needs training or help with hardware or software
   C. #ToDo To find if issue is controllable or not --> Check if possible.
   D. To find if customer is happy with service or needs further improvement and assistance

Now let's have a look at individual features:

1. Short description

   A. Total values - 8492 (> 99%)

   B. Missing values - 8 (< 1%)

   C. Distinct values - 7481 (88%)

   D. Mostly occurring value - password reset (0.4%)

   E. We can also see the number of times each value is being repeated

   F. Max length of statement - 159

G. It contains:

Characters -> Lowercase Letter, Punctuation,
Uppercase Letter, Decimal Number,
Math Symbol, Math Symbol,
Modifier Symbol, Other Number,
Other Symbol, Currency Symbol

Scripts -> Common(ASCII) and Latin

H. Point G indicates that we have to translate the texts in the dataset based on the scripts as part of data pre-processing.

2. Description

A. Total values - 8499 (> 99%)

B. Missing values - 1 (< 1%)

C. Distinct values - 7817 (92%)

D. Mostly occurring value - it shows "the" (0.7%) but will analyse further after the removal of stop words. But we consider the next which is windows password reset (0.3%)

E. We can also see the number of times each value is being repeated

F. Max length of statement - 13001

G. It contains:

Characters -> Lowercase Letter, Punctuation,
Uppercase Letter, Decimal Number,
Math Symbol, Math Symbol,
Modifier Symbol, Other Number,
Other Symbol, Currency Symbol

Scripts -> Common(ASCII) and Latin

H. Point G indicates that we have to translate the texts in the dataset based on the scripts as part of data pre-processing.

3. Caller

A. Total values - 8500 (100%)

B. Missing values - no missing value

C. Distinct values - 2950 (35%)

D. Mostly occurring value - bpctwhsn kzqsbmtp (10%)

E. We can also see the number of times each value is being repeated

F. Max length of statement - 30

G. It contains:

Characters -> Lowercase Letter, Space Separator,
Uppercase Letter, Connector Punctuation

Scripts -> Common(ASCII) and Latin

H. Point G indicates that we have to translate the texts in the dataset based on the scripts as part of data pre-processing.

4. Assignment Group

   A. Total values - 8500 (100%)

   B. Missing values - no missing value

   C. Distinct values - 74

   D. Mostly occurring value - GRP_0 (47% ~ nearly half of the data --> Hence we can say that target class is highly imbalanced, so needs a strategy to be employed to reduce the bias here)

   E. We can also see the number of times each value is being repeated

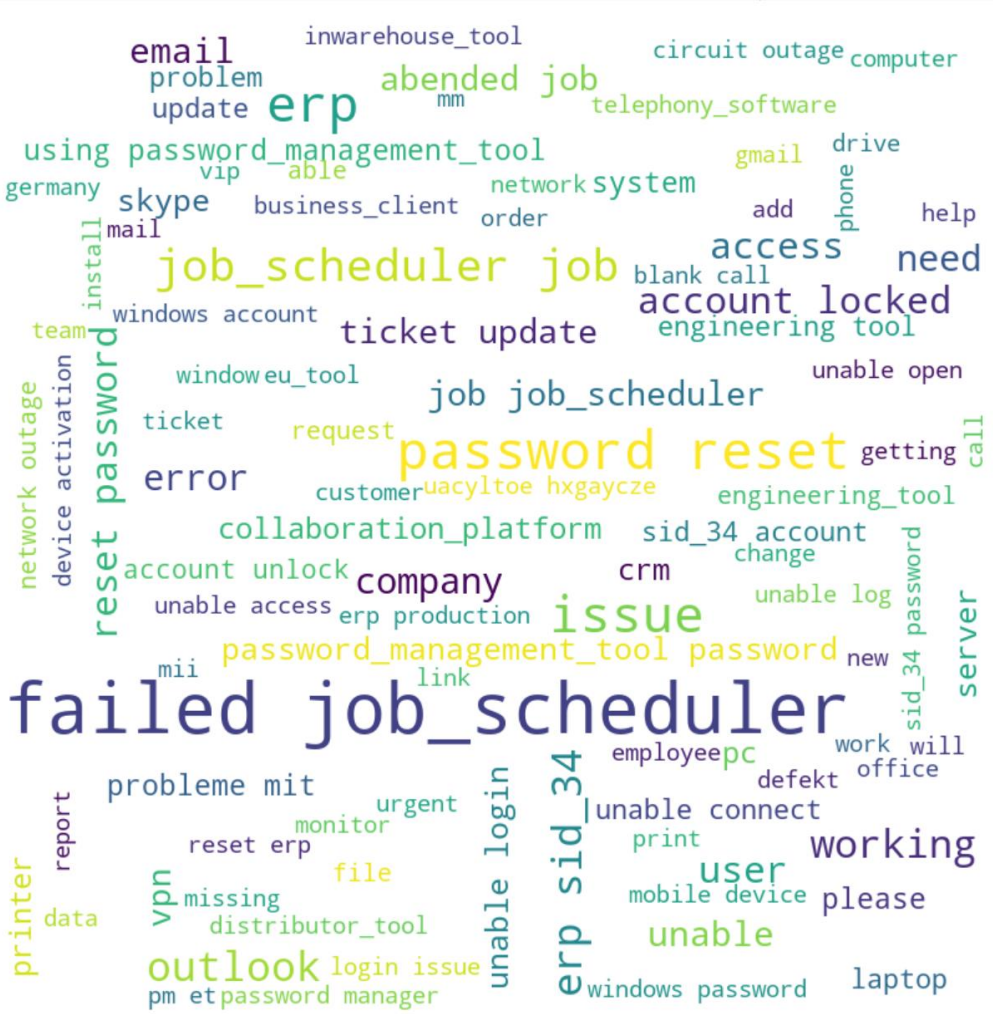
   G. This indicates we can merge few assignment groups with smaller percentage to reduce overall number of categories.

Other findings

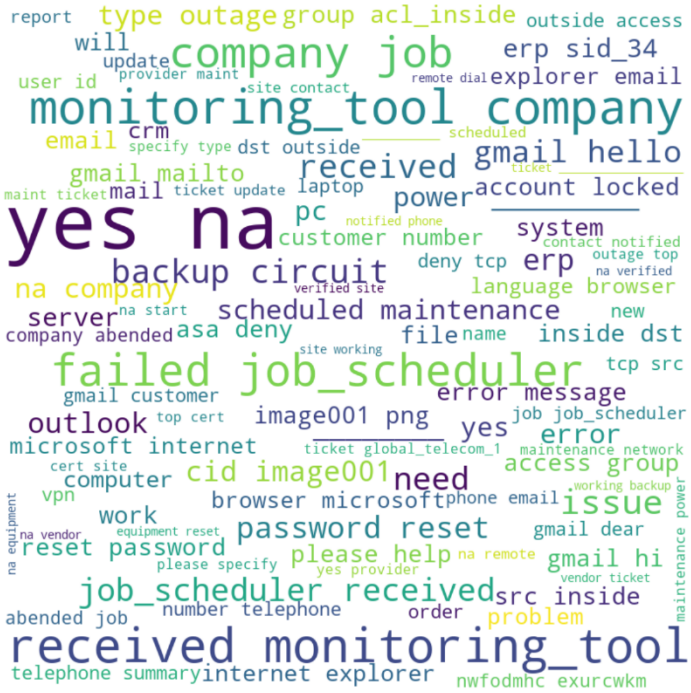
1. There are duplicates which needs to be tackled
2. There are mojibake texts in the description and short description which needs to be processed
3. There are texts belonging to different languages which needs translations
4. There are email ids, blank spaces, dates, numbers which needs to be processed
5. There are missing values to be treated

Further Data Analysis

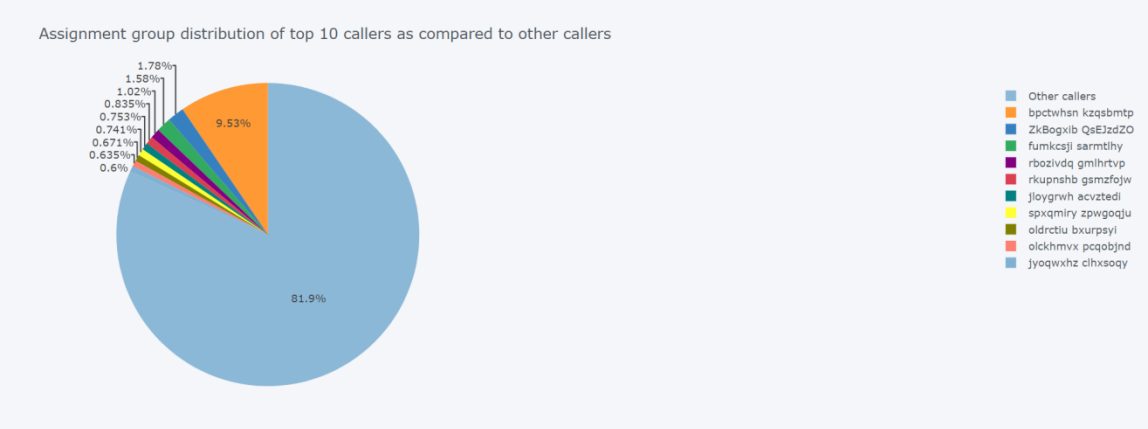
Word cloud for Short description



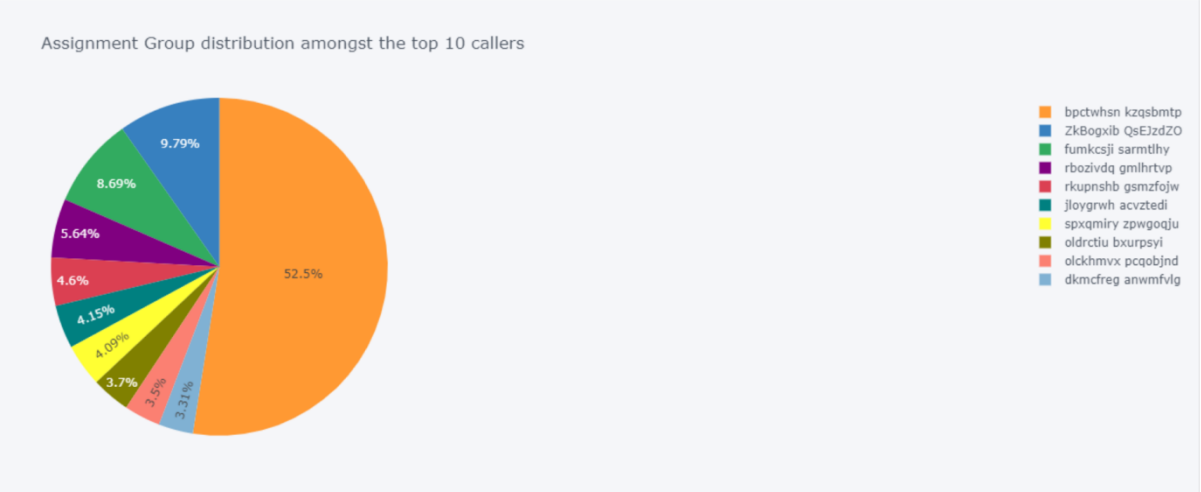
Word cloud for Description



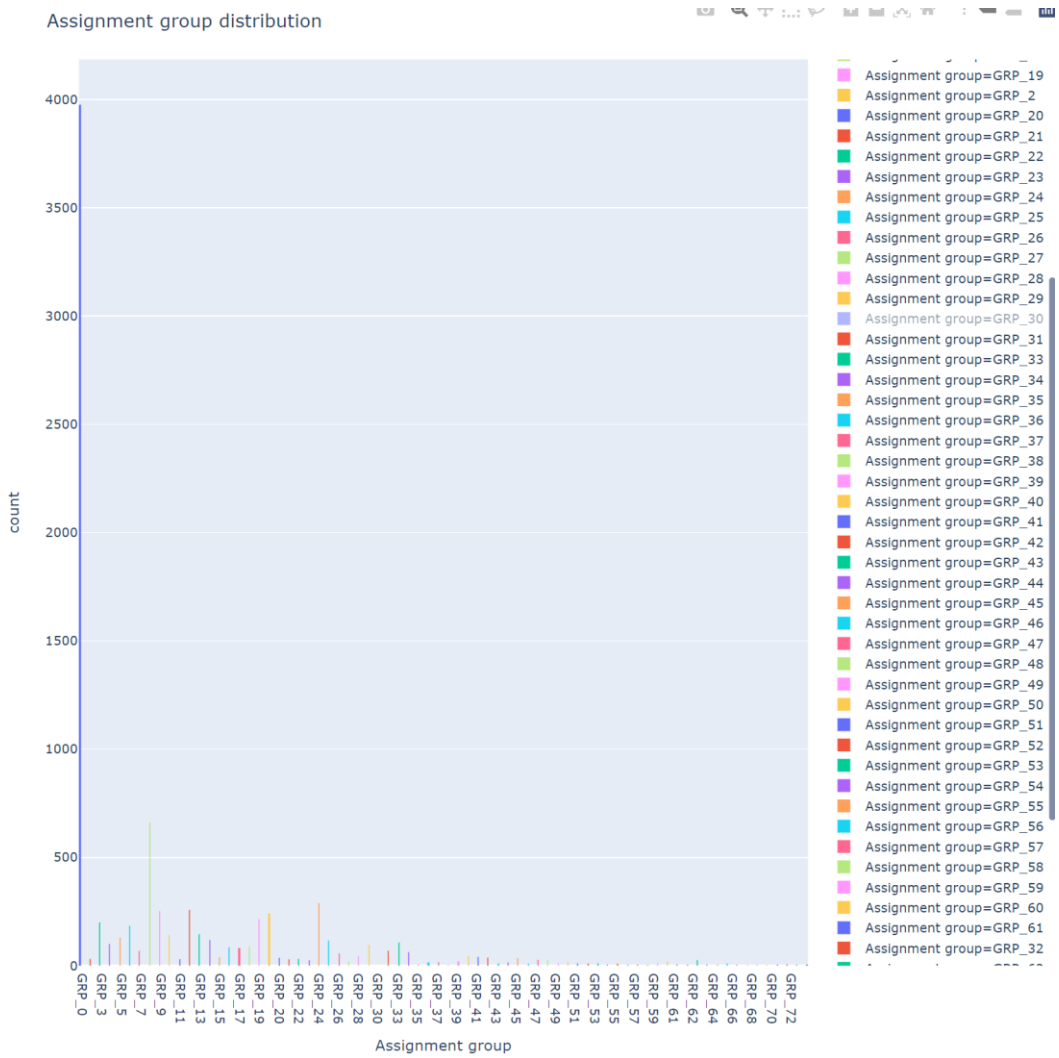
Assignment group distribution of top 10 callers as compared to other callers

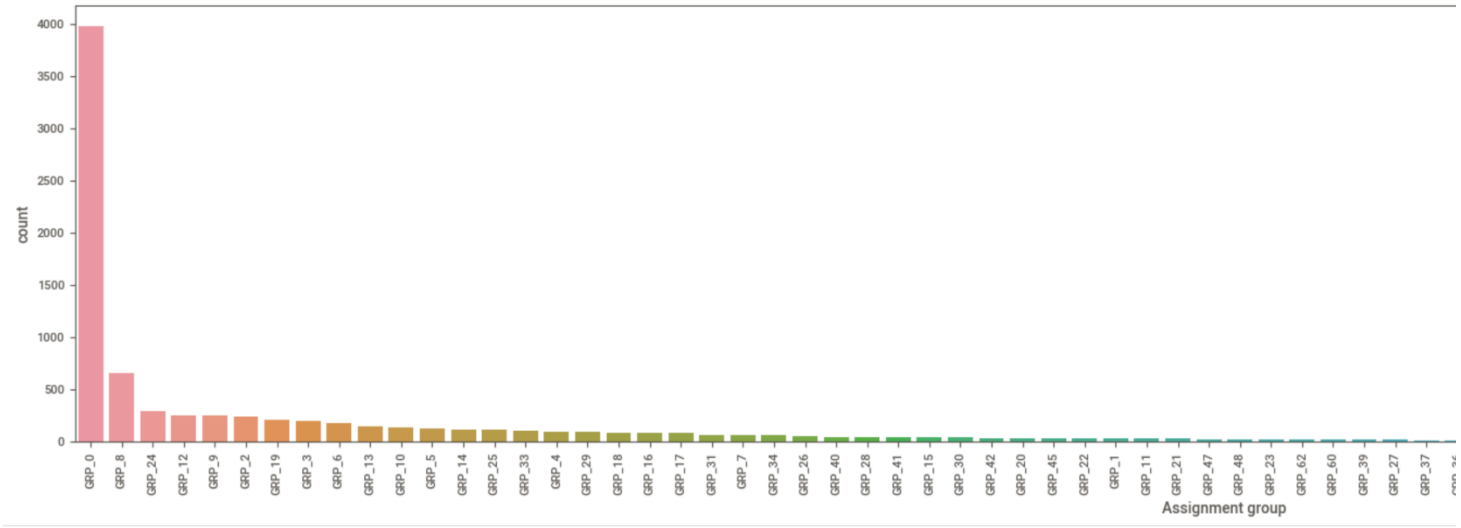


Assignment group distribution amongst top 10 callers

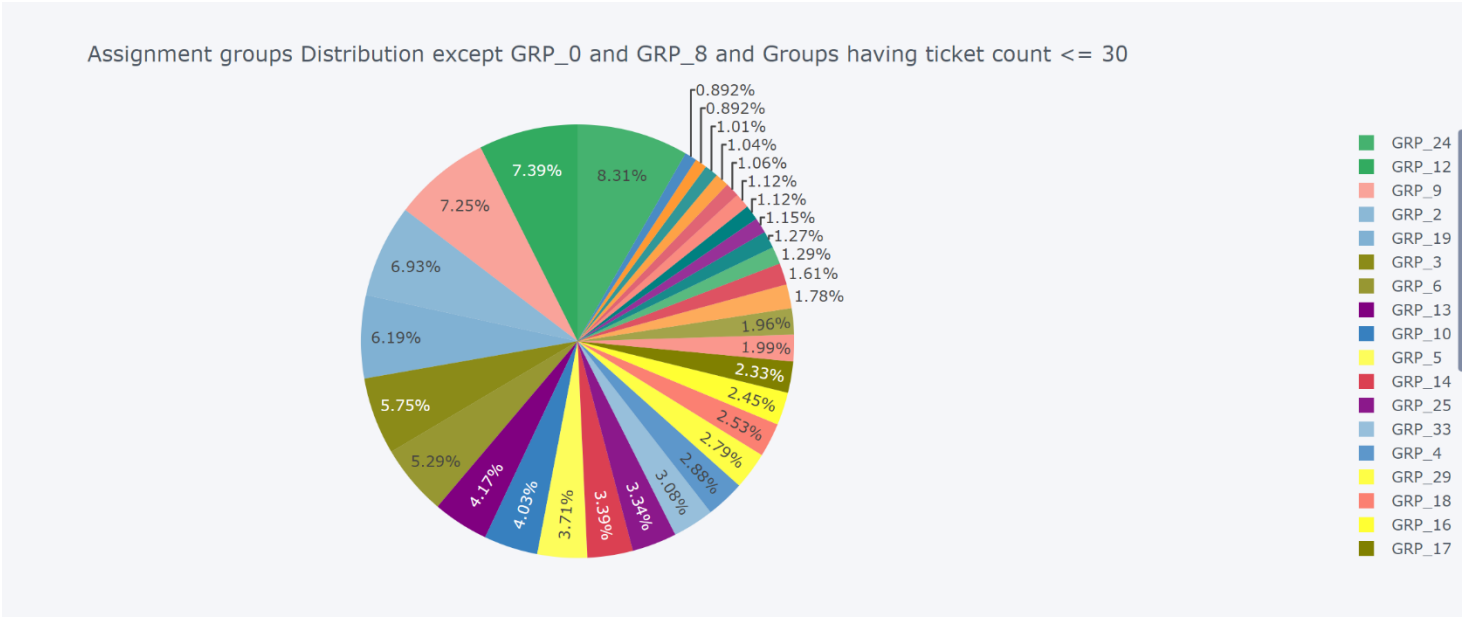


Assignment group distribution

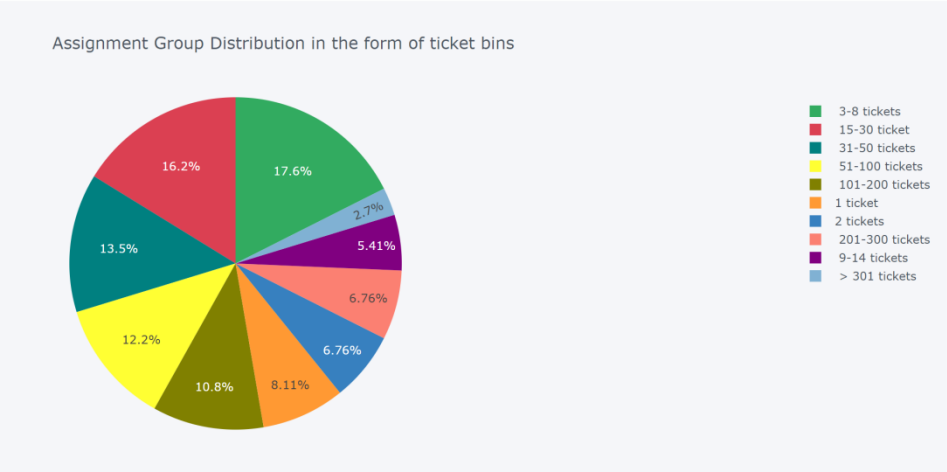




From above we see that there is significant class imbalance with GRP_0 being the majority class. Now let's see the distribution of classes other than two majority classes GRP_0 and GRP_8 an d also for those classes which have ticket count less than or equal to 30 (Number 30 is chosen using central limit theorem here)



Now we will see if we can ignore some classes or merge them into some other assignment group considering that other group has capability to resolve the ticket for this assignment group. For this we first divided the classes in some ticket bins and found the distribution as



We see above that majority of assignment groups are one with tickets between 3 and 8 i.e. 17.6%. Also, from the above chart, we can see that Assignment group <= 2 tickets contributes to 14.87% i.e. (8.11 + 6.76%).

Also after comparing the descriptions and understanding that other groups have capability to resolve same tickets. We found that we can either merge or ignore below assignment groups :

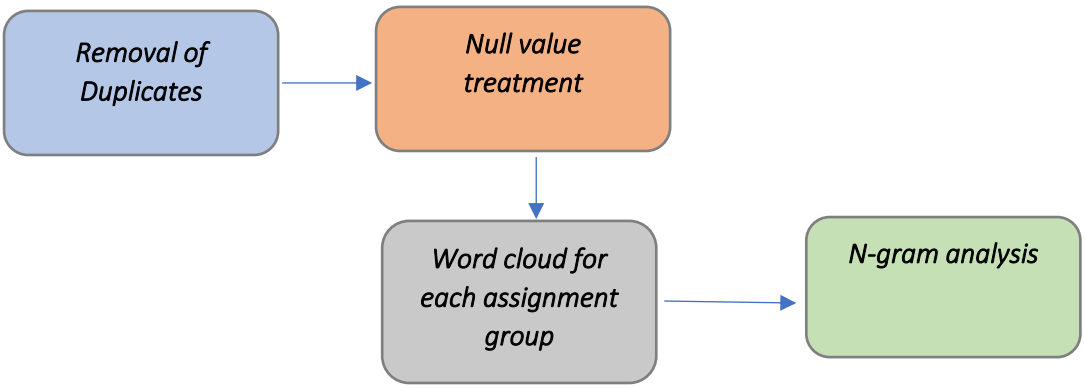
So finally, we see that below groups can be ignored based on above analysis. We will decide this later post feature engineering

1. GRP_72
2. GRP_54
3. GRP_57
4. GRP_69
5. GRP_67
6. GRP_35
7. GRP_73

But, we shall not ignore below assignment groups:

1. GRP_71
2. GRP_70
3. GRP_61
4. GRP_64

Feature Engineering



Removal of Duplicates: - Total of 83 duplicate records are found in the dataset.
Below approach was taken to remove them.

```
[ ] incidents_data.drop_duplicates(inplace=True, ignore_index=True)
```

```
[ ] print("Shape of the data post removal of duplicates")
    incidents_data.shape
```

```
Shape of the data post removal of duplicates
(8417, 4)
```

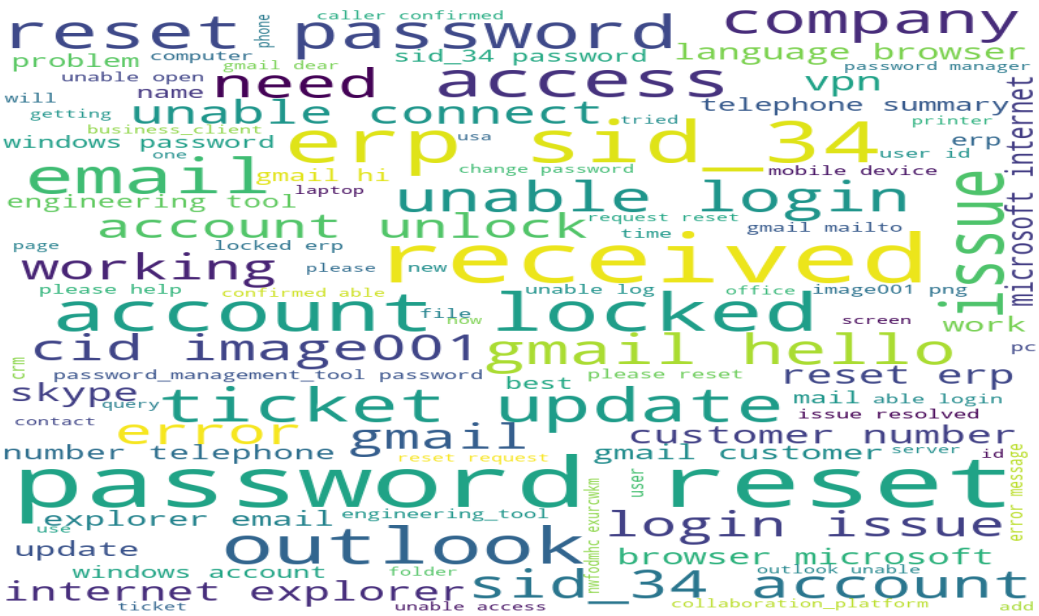
Treating of Null values: - Total of 8 null values were found for Short description and 1 for Description.

1. Drop the records with null values
2. Replace null value (NaN) with empty string
3. Look for same description in any other record and then replace the corresponding shor description or in case of description has null value then search for same short description and replace description with corresponding description

Approach-2 was taken to replace NULL values with empty strings as with approach 1 we end up losing descriptions and with approach 3 we end up with creating duplicate records.

WordCloud analysis of Assignment group: -

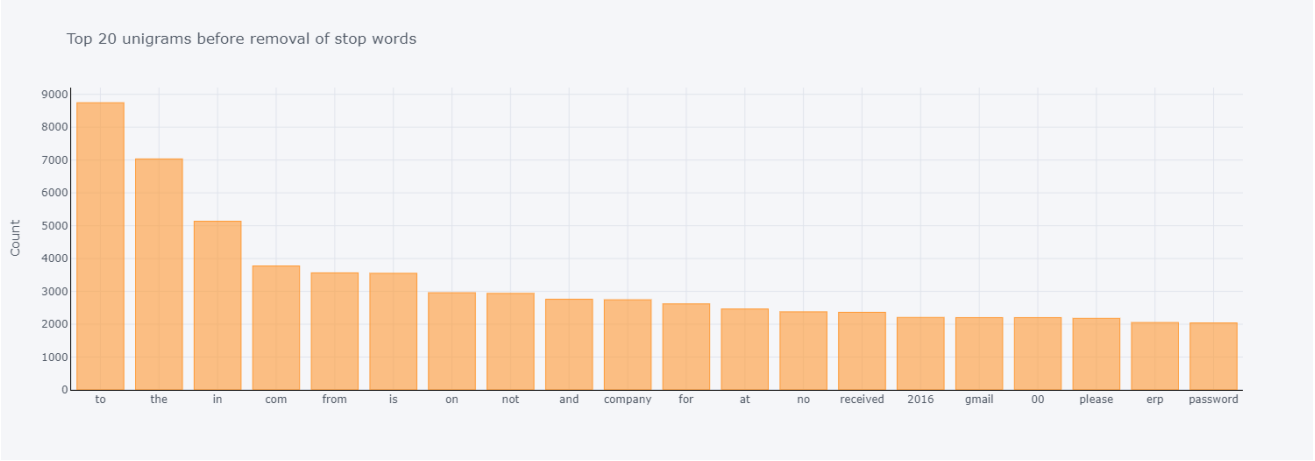
Assignment Group 0:



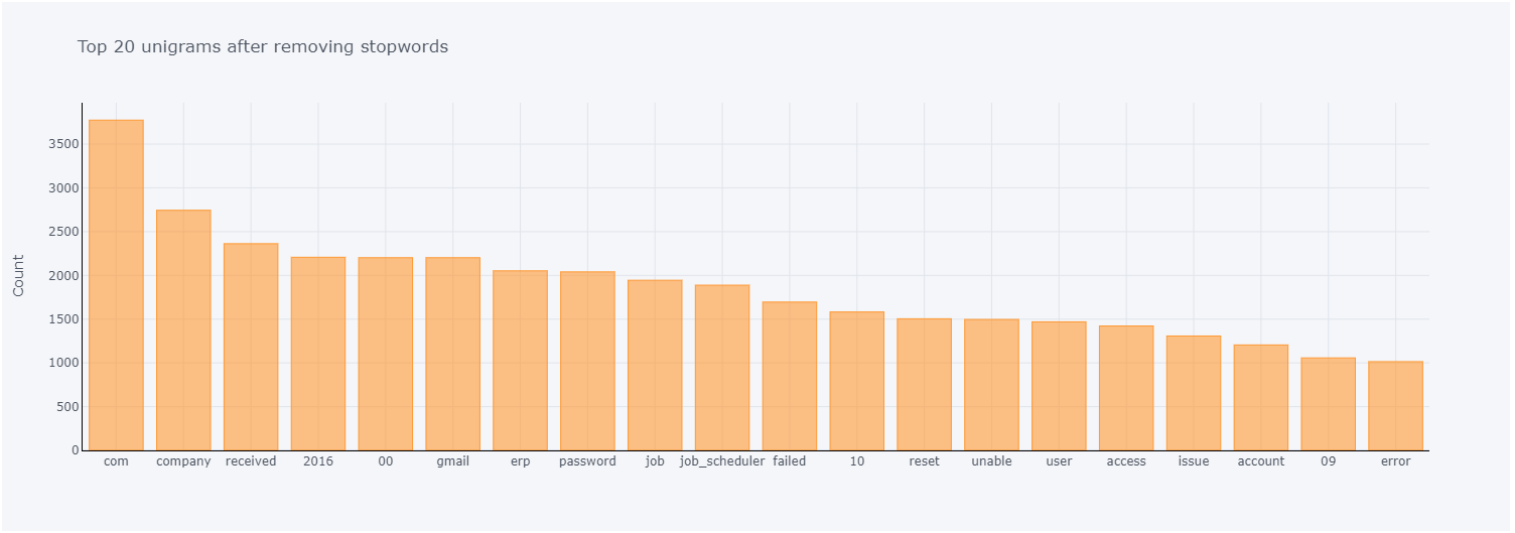
N-gram Analysis: -

N-gram, Unigram and Bigram analysis N grams are used to describe number of words as observations as unigram means singly-worded, bigram means 2-worded phrase, and trigram means 3-worded phrase.

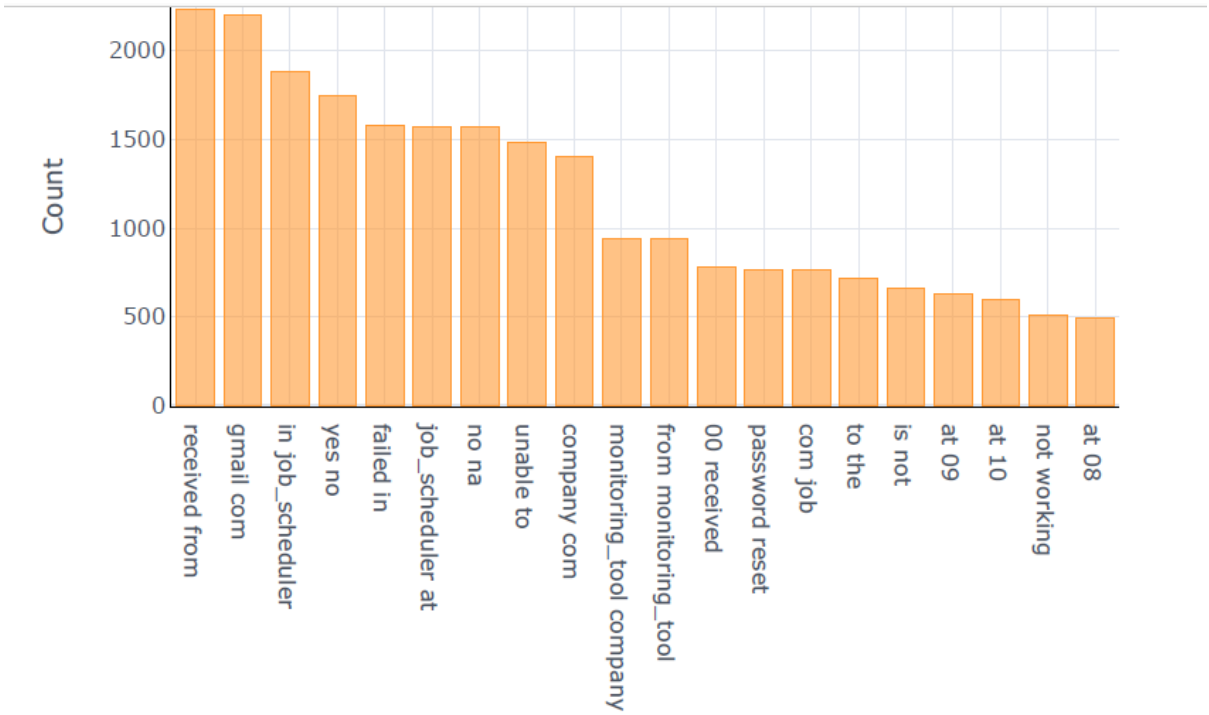
Top 20 unigrams before removal of stopwords



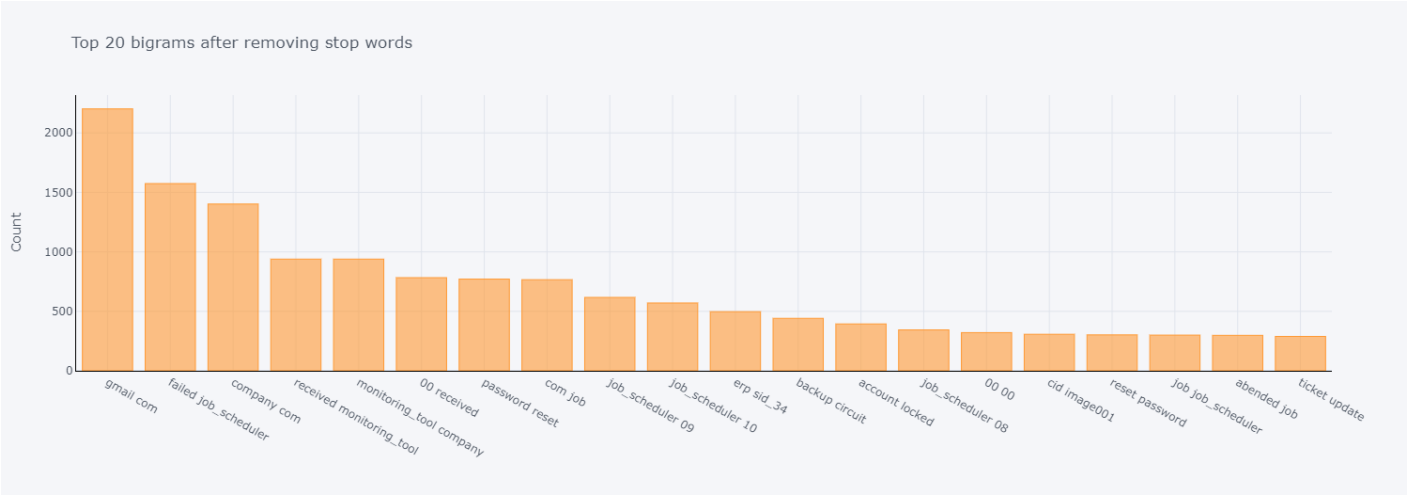
Top 20 unigrams after removal of stopwords



Top 20 bigrams before removal of stopwords



Top 20 bigrams after removal of stopwords



Data Pre-processing

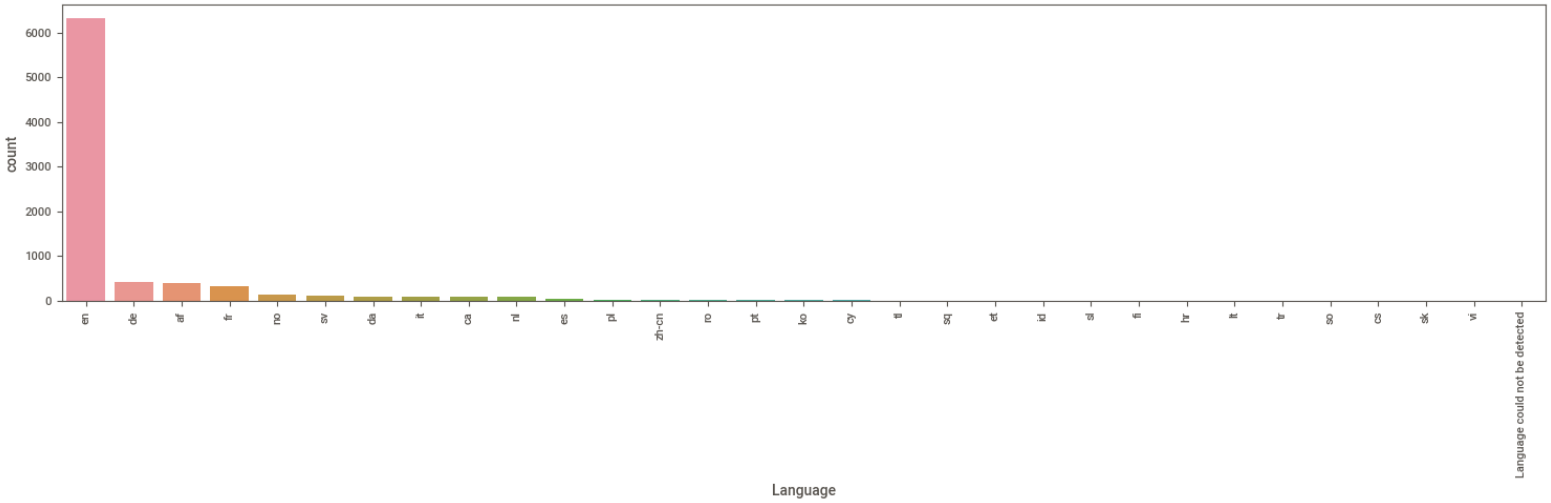
Fix Encoding: - FTFY (Fixes Text For You) library is used to detect and fix Mojibake texts

```
def fix_mojibake_text(dataframe,description_column):
  fixed_text = []
  for column in dataframe[[description_column]]:
    # Select column contents by column name using [] operator
    Text = dataframe[column]
    print('Text : ', len(Text))
    for i in range(len(Text)) :
      #print(i)
      fix_text_i = ftfy.fix_text(Text.values[i])
      fixed_text.append(fix_text_i)
    return fixed_text
```

Detection of Different Languages: - LangDetect library is used to detect different languages

```
def language_detector(dataframe):
  try:
    language = detect(dataframe)
    return language
  except:
    return 'Language could not be detected'
```

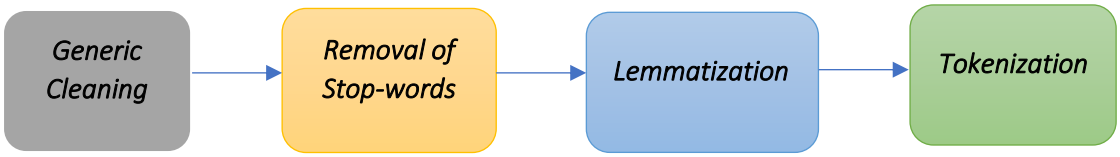
Using this, it was found that most of the descriptions within the dataset are in English language followed by German.



Translation: - TRANSLATE-API library is used to translate texts into English language.

Attached is the pickled file containing English texts post translation.

Data Cleaning



Generic

1. Convert the text to lowercase
2. Removing punctuation marks and other special characters
3. Removing numbers as converting them into corresponding words will result into dominance otherwise
4. Removing blank spaces, horizontal tab spaces, new line breaks with single space.
5. Removing stop words, parse terms and other particular words.
6. Removing email addresses as it will not add any significant value into the expected analysis.

Removal of Stop words: -

From n-gram analysis, we see that there are few english stopwords in the Ticket_Description feature other than the default ones provided by NLTK library, default list was extended as below:-

```
stopwords = stopwords.union({'yes','na','hi',
                             'hello','no',
                             'regards','thanks',
                             'from','greeting',
                             'reply',
                             'will','please',
                             'see'})
```

```
incidents_data['cleaned_description'] = incidents_data['Translated_text']
                                        .apply(lambda x: " ".join(x for x in str(x).split() if x not in stopwords))
incidents_data['cleaned_description'].head()
```

Lemmatization: -

Stemming and Lemmatization are Text Normalization (or sometimes called Word Normalization) techniques in the field of Natural Language Processing that are used to prepare text, words, and documents for further processing.

Stemming is the process of reducing inflection in words to their root forms such as mapping a group of words to the same stem even if the stem itself is not a valid word in the Language.

Lemmatization, unlike Stemming, reduces the inflected words properly ensuring that the root word belongs to the language. In Lemmatization root word is called Lemma.

```
## Lemmatization
incidents_data['cleaned_description']= incidents_data['cleaned_description']
                                       .apply(lambda x: " ".join([Word(word).lemmatize() for word in str(x).split()]))
incidents_data['cleaned_description'].head()
```

```
0    login issue verified user detail employee mana...
1    outlook received hmjdrvpb komuaywn gmail com t...
2    cant log vpn received eylqgodm ybqkwiam gmail ...
3    unable access hr tool page unable access hr to...
4                          skype error skype error
Name: cleaned_description, dtype: object
```

Tokenization: -

```
[ ]  tokenizer = nltk.tokenize.RegexpTokenizer(r'\w+')
     incidents_data['token_desc'] = incidents_data['cleaned_description'].apply(lambda x: tokenizer.tokenize(x))
```

```
[ ]  incidents_data['token_desc'].head()
```

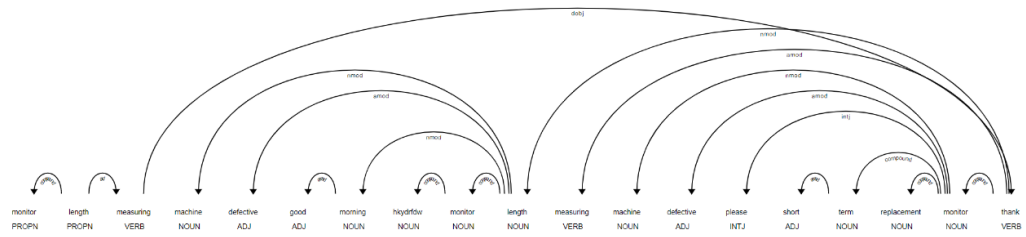
```
0    [login, issue, verified, user, detail, employe...
1    [outlook, received, hmjdrvpb, komuaywn, gmail,...
2    [cant, log, vpn, received, eylqgodm, ybqkwiam,...
3    [unable, access, hr, tool, page, unable, acces...
4                        [skype, error, skype, error]
Name: token_desc, dtype: object
```

NER and POS Tagging

```
Named Entity Recognition for Row 900
--------------------------------------------------------------------------
--------------------------------------------------------------------------
```

monitor length measuring machine defective good morning TIME hkydrfdw monitor length measuring machine defective short term replacement monitor thank

```
Dependencies for Row 900
--------------------------------------------------------------------------
--------------------------------------------------------------------------
```

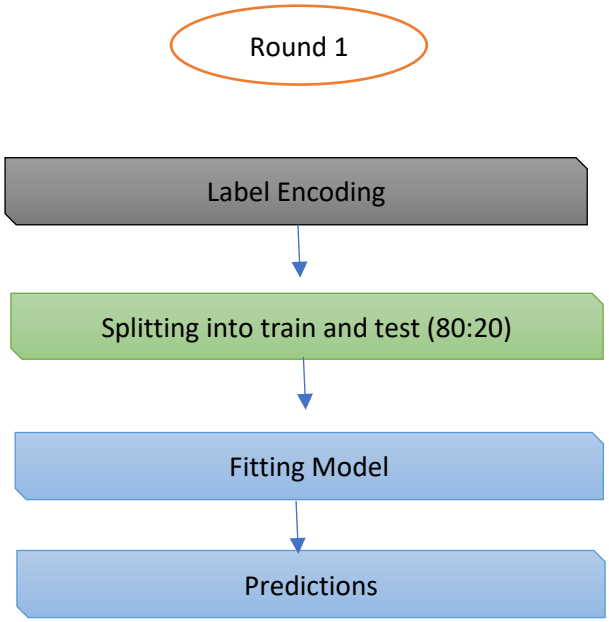


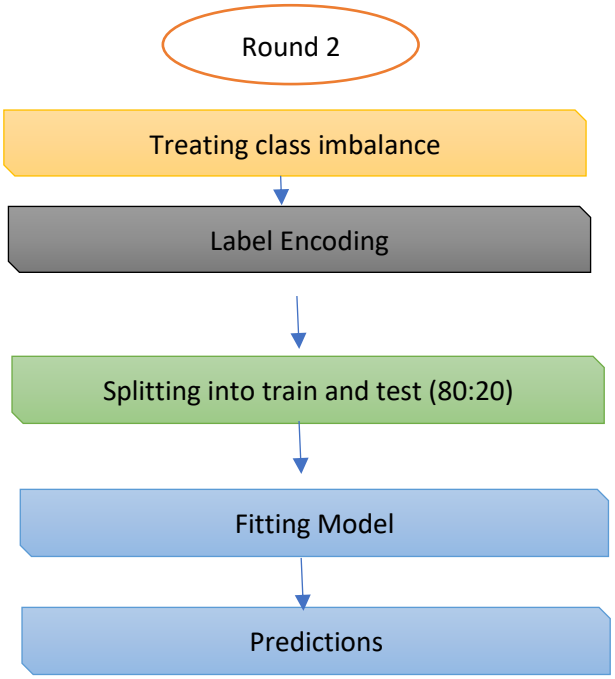

```
Word in sentence, Parts-of-speech and Lemmatized sentence 900
--------------------------------------------------------------------------
--------------------------------------------------------------------------
[('monitor', 'PROPN', 'monitor'), ('length', 'PROPN', 'length'), ('measuring', 'VERB', 'measure'), ('machine', 'NOUN', 'machine'), ('defe
ctive', 'ADJ', 'defective'), ('good', 'ADJ', 'good'), ('morning', 'NOUN', 'morning'), ('hkydrfdw', 'NOUN', 'hkydrfdw'), ('monitor', 'NOU
N', 'monitor'), ('length', 'NOUN', 'length'), ('measuring', 'VERB', 'measure'), ('machine', 'NOUN', 'machine'), ('defective', 'ADJ', 'def
ective'), ('short', 'ADJ', 'short'), ('term', 'NOUN', 'term'), ('replacement', 'NOUN', 'replacement'), ('monitor', 'NOUN', 'monitor'),
('thank', 'VERB', 'thank')]

[(monitor, 'O', ''), (length, 'O', ''), (measuring, 'O', ''), (machine, 'O', ''), (defective, 'O', ''), (good, 'O', ''), (morning, 'B',
'TIME'), (hkydrfdw, 'O', ''), (monitor, 'O', ''), (length, 'O', ''), (measuring, 'O', ''), (machine, 'O', ''), (defective, 'O', ''), (ple
ase, 'O', ''), (short, 'O', ''), (term, 'O', ''), (replacement, 'O', ''), (monitor, 'O', ''), (thank, 'O', '')]

{'morning': 'TIME'}
```

	cleaned_description	tag_counts
0	[(login, JJ), (issue, NN), (verified, VBD), (u...	{'JJ': 5, 'NN': 13, 'VBD': 4, 'VB': 1, 'VBN': 1}
1	[(outlook, NN), (received, VBD), (hmjdrvpb, JJ...	{'NN': 10, 'VBD': 1, 'JJ': 3, 'NNS': 1, 'VBP':...
2	[(cant, JJ), (log, NN), (vpn, NN), (received, ...	{'JJ': 3, 'NN': 7, 'VBD': 1, 'JJS': 1}
3	[(unable, JJ), (access, NN), (hr, NN), (tool, ...	{'JJ': 2, 'NN': 8}
4	[(skype, JJ), (error, NN), (skype, NN), (error...	{'JJ': 1, 'NN': 3}

Deciding Models and Model Building

Two rounds were carried out as below:





Label Encoding: -

```
[ ] X = final_df['token_description']
    y = final_df['Assignment group']
```

```
encoder = preprocessing.LabelEncoder()
# encoding train labels
encoder.fit(y)
y = encoder.transform(y)
```

Splitting of data into train and test sets: -

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2, random_state=13)
```

Treating class imbalance: -

Strategies Tried:

1. **SMOTE(Synthetic Minority Oversampling Technique):**

   Approach involves duplicating examples in the minority class, although these examples don't add any new information to the model. Instead, new examples can be synthesized from the existing examples. This is a type of data augmentation for the minority class and is referred to as the **Synthetic Minority Oversampling Technique, or SMOTE** for short.

   Limitations: - Given the large dataset, with SMOTE it was getting timed-out as it uses KNN technique in the background.

2. **Random Over-sampler and Under-sampler:**

The two main approaches to randomly resampling an imbalanced dataset are to delete examples from the majority class, called undersampling, and to duplicate examples from the minority class, called oversampling.

Limitations: - On implementing the technique, balancing accuracy was on the lower side resulted in less improvement in the model performance.

Possible Reasons: -

1. Random oversampling duplicates examples from the minority class inthe training dataset and results in overfitting.
2. Random undersampling deletes examples from the majority class and results in losing information valuable to a model.

3. **Random Over-sampler:**

Approach:

1. Convert Tokens into corresponding TF-IDF vectors using TFIDFVectorizer.

```
# word level tf-idf for ticket
tfidf = TfidfVectorizer(analyzer = 'word', min_df=2, max_df=0.95, ngram_range=(1, 2))
incidents_tfidf = tfidf.fit_transform(incidents_data_final_df['token_description'])
```

2. Convert TFIDF into Dataframe

```
# collect the tfid matrix in numpy array
incident_tfidf_array = incidents_tfidf.todense()

# store the tf-idf array into pandas dataframe
incident_tfidf_df = pd.DataFrame(incident_tfidf_array)
```

3. Apply ROS

```
ros = RandomOverSampler(random_state=777)
X_resmapled, y_resmapled = ros.fit_sample(incidents_tfidf, y)
```

Limitations: -

1. This approach worked well with Traditional models but didn't worked for Sequential models.

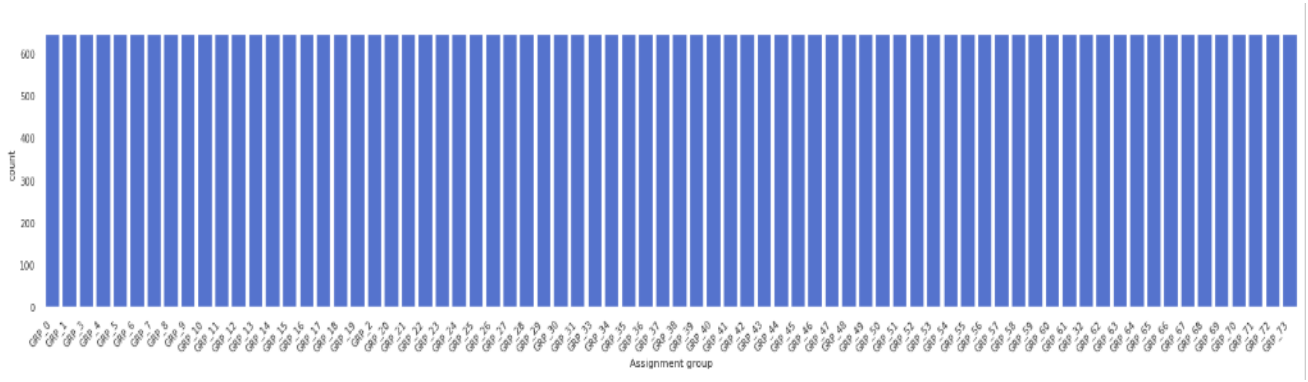
2. As for Sequential models, we need embeddings for the texts. But with TFIDF vectors created, we don't have texts that can be used for generating the word embeddings. Also, Re-engineering is not feasible here as we will not get back the texts in original sequence.

4. Resample (from sklearn.utils) : -

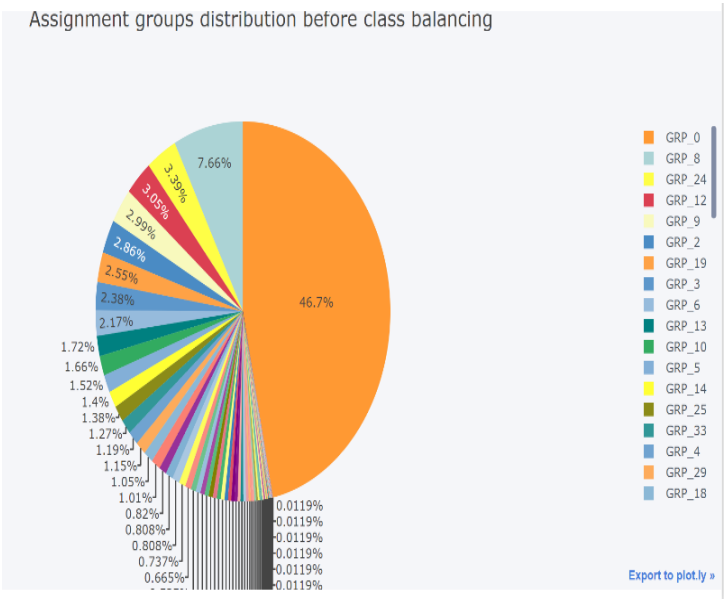
The idea is to oversample the data related to minority class using replacement. One of the parameters is replace and other one is n_samples which relates to number of samples to which minority class will be oversampled. Once the sampling is done, the balanced dataset is created by appending the oversampled data.

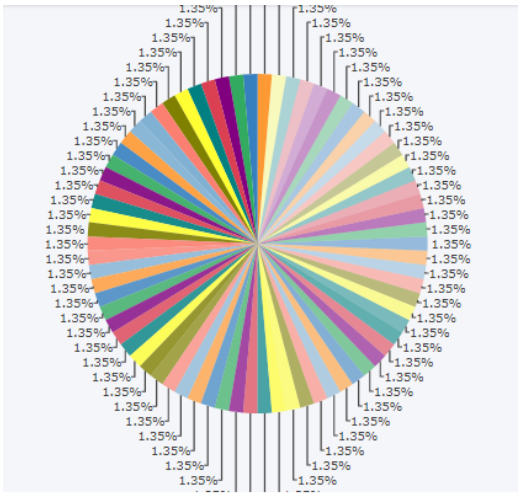
With this approach, each Group is now balanced with total of 645 tickets assigned.

Below are the diagrammatic representations of the solution implemented: -



The Assignment group before and after class balancing is shown as below:





Benefits: -

Worked well with both Traditional and Sequential models resulting in improved performance.

Models

Vectorized data has been used to train the models. Traditional and Sequential models have been evaluated with various metrices such as Accuracy, Precision-score, F1-score and Recall-score.

Approach: -

Generic function has been built with pipeline to be executed for each of the models with the output comprising of Evaluation metrics, Execution time, Accuracy and Loss plots over epochs.

Set of two iterations are executed with Round –1 and Round-2 with hyperparameter tuning.

Traditional Models

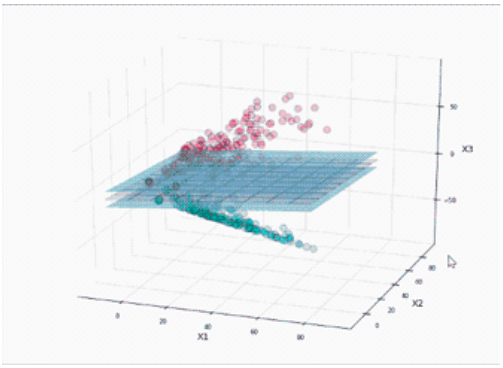
1. **Multinomial NB Classifier:**

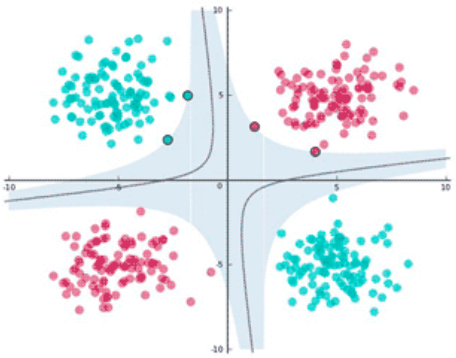
Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. Naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.

```
multi_nb_clf = MultinomialNB(alpha=0.25)
multi_nb_clf = OneVsRestClassifier(multi_nb_clf)
```

2. **SVC Classifier:**

Linear SVC (Support Vector Classifier) fits the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data. From there, after getting the hyperplane, you can then feed some features to your classifier to see what the "predicted" class is.

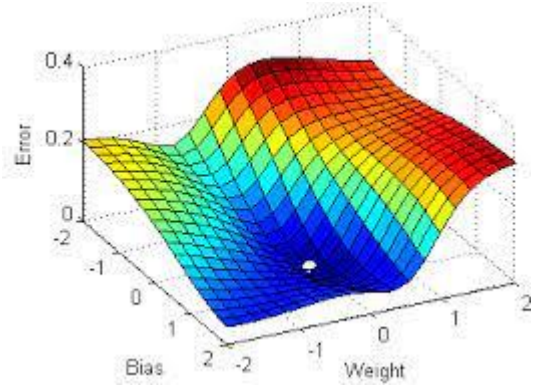




```
svc_clf = LinearSVC()
svc_clf = OneVsRestClassifier(svc_clf)
clf_table,history = run_classifier(svc_clf,'SVC Classifier',X_resampled_train,X_resampled_test,y_resampled_train,
                                   y_resampled_test,False,classifier_columns,clf_table)
```

3. **SGD Classifier:**

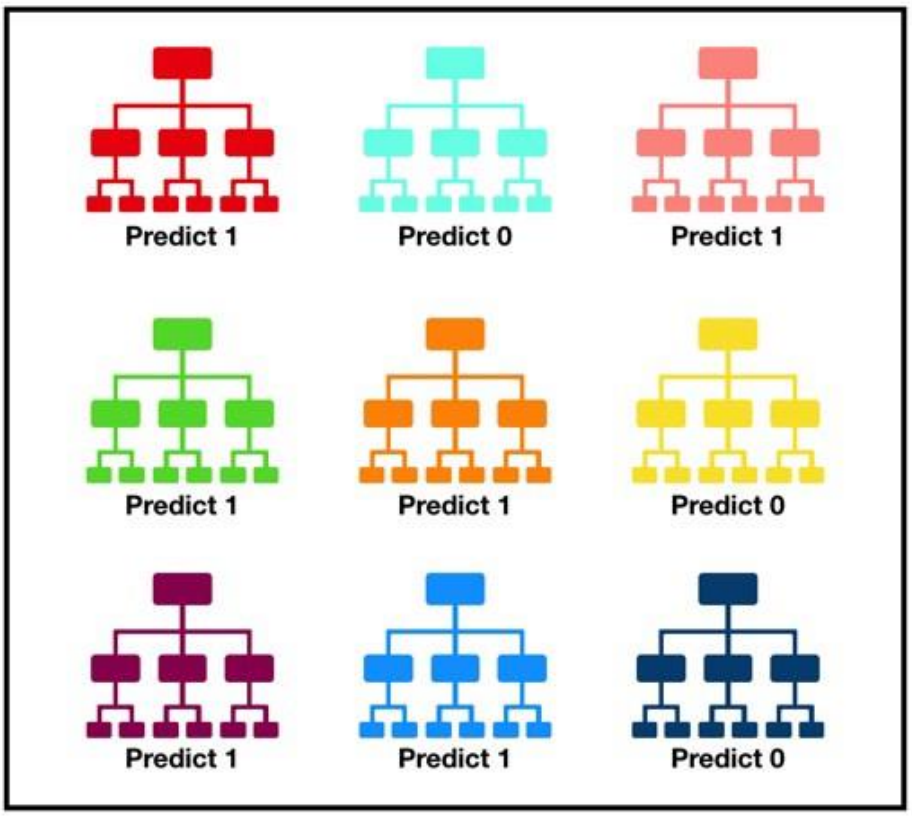
Stochastic Gradient Descent (SGD) is a simple yet efficient optimization algorithm used to find the values of parameters/coefficients of functions that minimize a cost function. In other words, it is used for discriminative learning of linear classifiers under convex loss functions such as SVM and Logistic regression.




```
sgd_model_clf = SGDClassifier(loss='hinge', penalty='l2',alpha=1e-3, random_state=42, max_iter=5, tol=None)
clf_table,history = run_classifier(sgd_model_clf,'SGD Classifier',X_resampled_train,
                                   X_resampled_test,y_resampled_train,y_resampled_test,False,classifier_columns,clf_table)
```

4. RandomForest Classifier:

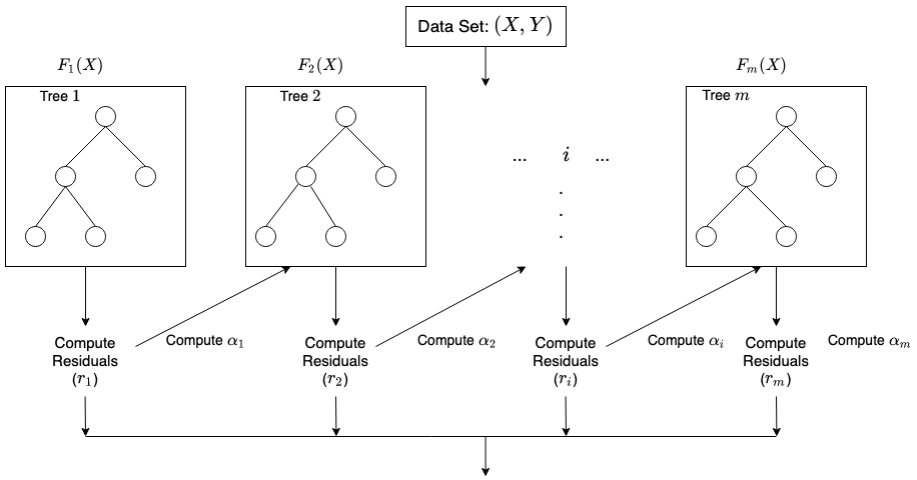
Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes become our model's prediction.



```
random_forest_Clf=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None, criterion='gini',
                                         max_depth=None, max_features='auto', max_leaf_nodes=None,
                                         max_samples=None, min_impurity_decrease=0.0, min_samples_leaf=1,
                                         min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100,
                                         n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)
clf_table,history = run_classifier(random_forest_Clf,'Random Forest Classifier',X_resampled_train,X_resampled_test,
                                   y_resampled_train,y_resampled_test,False,classifier_columns,clf_table)
```

5. XGBoost:

XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. An optimized distributed gradient boosting library designed to be highly efficient, flexible and portable.



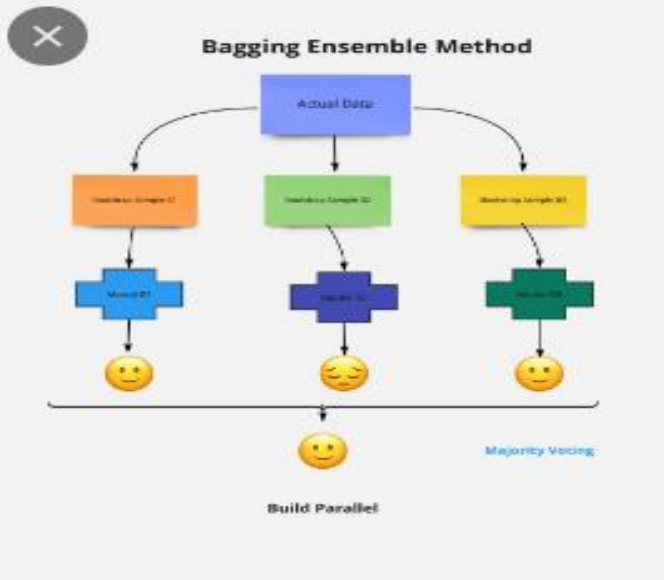
$$F_m(X) = F_{m-1}(X) + \alpha_m h_m(X, r_{m-1}),$$

where α_i, and r_i are the regularization parameters and residuals computed with the i^{th} tree respectfully, and h_i is a function that is trained to predict residuals, r_i using X for the i^{th} tree. To compute α_i we use the residuals computed, r_i and compute the following: $arg \min_{\alpha} = \sum_{i=1}^{m} L(Y_i, F_{i-1}(X_i) + \alpha h_i(X_i, r_{i-1}))$ where $L(Y, F(X))$ is a differentiable loss function.

```
xgb_clf = xgboost.XGBClassifier(max_depth=7, n_estimators=200, colsample_bytree=0.8, subsample=0.8, nthread=10, learning_rate=0.1)
clf_table,history = run_classifier(xgb_clf,'XgBoost Classifier',X_resampled_train,X_resampled_test,
                                   y_resampled_train,y_resampled_test,False,classifier_columns,clf_table)
```

6. **Bagging :**

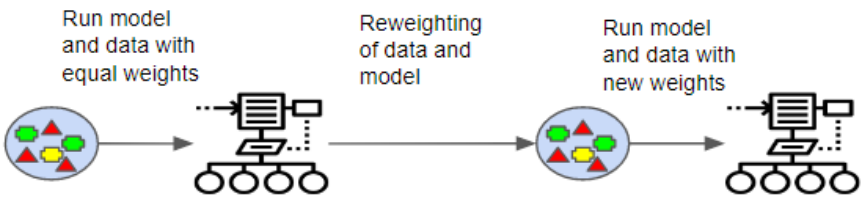
A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction.




```
bagging_clf = BaggingClassifier(tree.DecisionTreeClassifier(random_state=1))
clf_table,history = run_classifier(bagging_clf,'Bagging Classifier',X_resampled_train,X_resampled_test,
                                   y_resampled_train,y_resampled_test,False,classifier_columns,clf_table)
```

7. **GradientBoosting Classifier:**

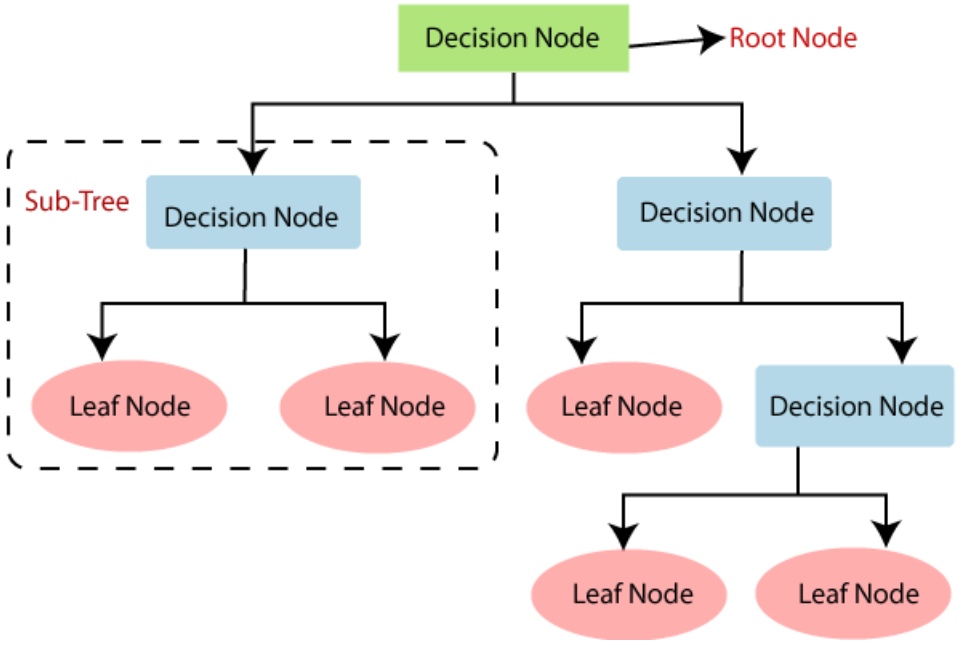
   In Gradient Boosting, each predictor tries to improve on its predecessor by reducing the errors. But the fascinating idea behind Gradient Boosting is that instead of fitting a predictor on the data at each iteration, it actually fits a new predictor to the residual errors made by the previous predictor.




```
boosting_clf = GradientBoostingClassifier(learning_rate=0.01,random_state=1)
clf_table,history = run_classifier(boosting_clf,'Boosting Classifier',X_resampled_train,X_resampled_test,
                                   y_resampled_train,y_resampled_test,False,classifier_columns,clf_table)
```

8. **DecisiontTree Classifier:**

   Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. ... The decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.




```
decisionTree_clf = tree.DecisionTreeClassifier()
clf_table,history = run_classifier(decisionTree_clf,'DecisionTree Classifier',X_resampled_train,X_resampled_test,
                                   y_resampled_train,y_resampled_test,False,classifier_columns,clf_table)
```

Evaluation Metrics for Traditional models: -

Classifier	Accuracy	F1-Score	Precision	Recall	Time taken
MultinomialNB Classifier - Round 2	0.902472	0.902598	0.921811	0.902472	3.713569s
SVC Classifier - Round 2	0.948670	0.946304	0.965018	0.948670	7.533591s
SGD Classifier - Round 2	0.869265	0.879354	0.891305	0.869265	3.873436s
Random Forest Classifier - Round 2	0.957783	0.954622	0.975372	0.957783	42.754390s
XgBoost Classifier - Round 2	0.956631	0.953569	0.974189	0.956631	856.407620s
Bagging Classifier - Round 2	0.952965	0.950734	0.970131	0.952965	54.931186s
Boosting Classifier - Round 2	0.836371	0.826056	0.884197	0.836371	1401.513942s
DecisionTree Classifier - Round 2	0.953384	0.950845	0.970456	0.953384	7.126146s

Evaluation Metrics for Sequential models: -

Classifier	Accuracy	F1-Score	Precision	Recall	Time taken
LSTM	0.920386	0.918289	0.934078	0.920386	3576.689095s
CNN - Round 2	0.935156	0.932964	0.953313	0.935156	238.385721s
GRU	0.943956	0.940658	0.963943	0.943956	5117.891881s

Generating Embeddings & Embeddings-Matrix

We are using Glove 6Billion word embeddings. Used 200d file which has 200 embedding dimensions for each word in the corpus respectively.

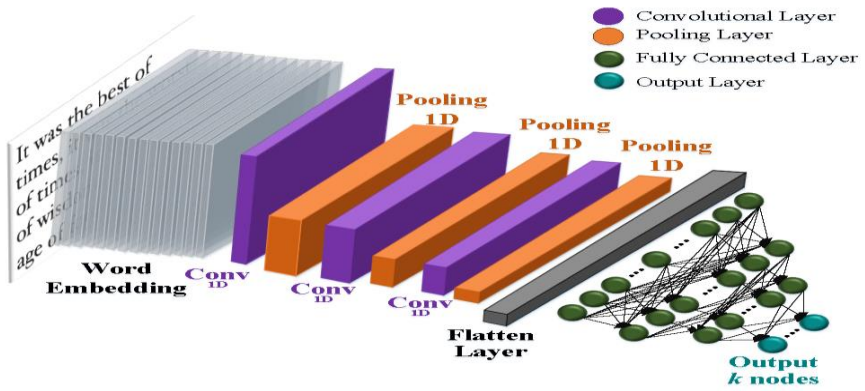
```
gloveFileName = '/content/drive/MyDrive/Capstone/glove.6B.200d.txt'
#Generating the gloved embedded data
X_resampled_train_g, X_resampled_test_g, word_index_g, embeddings_index = generate_embeddings(X_resampled_train,
                                                                                              X_resampled_test,gloveFileName)
```

```
(38184, 1000)
(9546, 1000)
Max no. of words  10535
Total word vectors 400000
```

```
embeddings_matrix = get_embedding_matrix(word_index_g,embedding_dim)
```

```
Embedding Matrix shape (10535, 200)
```

1. **CNN:** Convolutional Neural Networks (CNN) . Although originally built for image processing with architecture similar to the visual cortex, CNNs have also been effectively used for text classification. In a basic CNN for image processing, an image tensor is convolved with a set of kernels of size d by d. These convolution layers are called feature maps and can be stacked to provide multiple filters on the input. To reduce the computational complexity, CNNs use pooling which reduces the size of the output from one layer to the next in the network.



Model: "sequential_1" _____
Layer (type) Output Shape Param #
==========

embedding_1 (Embedding) (None, 1000, 200) 2107000
_____

conv1d (Conv1D) (None, 1000, 128) 76928
_____

max_pooling1d (MaxPooling1D) (None, 333, 128) 0
_____

conv1d_1 (Conv1D) (None, 331, 64) 24640
_____

max_pooling1d_1 (MaxPooling1 (None, 110, 64) 0
_____

dense_2 (Dense) (None, 110, 1024) 66560

_____

dropout_2 (Dropout) (None, 110, 1024) 0

_____

dense_3 (Dense) (None, 110, 256) 262400

_____

dropout_3 (Dropout) (None, 110, 256) 0

_____

dense_4 (Dense) (None, 110, 128) 32896

_____

dropout_4 (Dropout) (None, 110, 128) 0

_____

flatten_1 (Flatten) (None, 14080) 0

_____

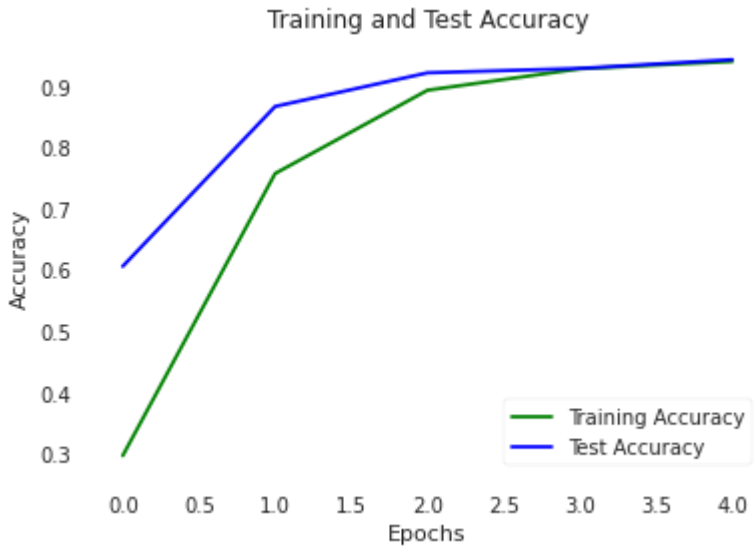
dense_5 (Dense) (None, 74) 1041994

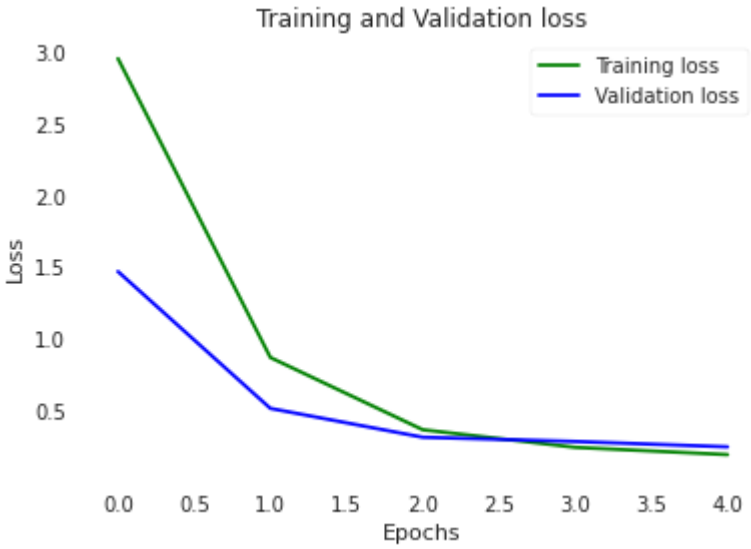
==========

Total params: 3,612,418

Trainable params: 3,612,418

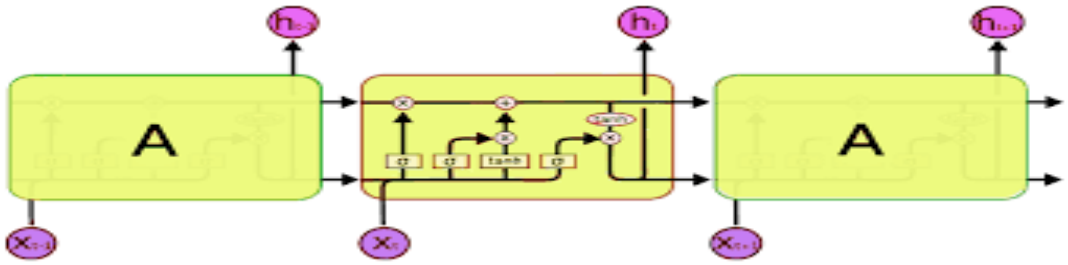
Non-trainable params: 0





2. **LSTM:**

   Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture[1] used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video).



Model: "sequential" _____

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 1000, 200)	2107000
bidirectional (Bidirectional	(None, 1000, 256)	336896
bidirectional_1 (Bidirection	(None, 128)	164352
dropout (Dropout)	(None, 128)	0
dense (Dense)	(None, 100)	12900

dropout_1 (Dropout) (None, 100) 0

_____

flatten (Flatten) (None, 100) 0

_____

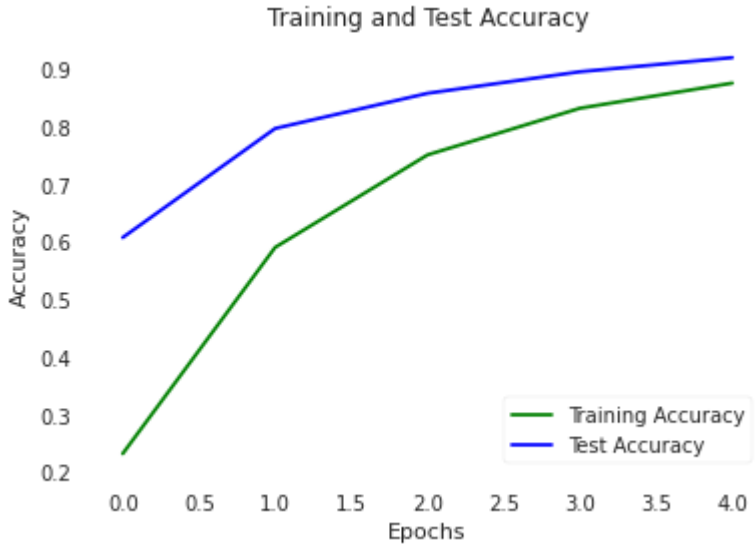
dense_1 (Dense) (None, 74) 74 74

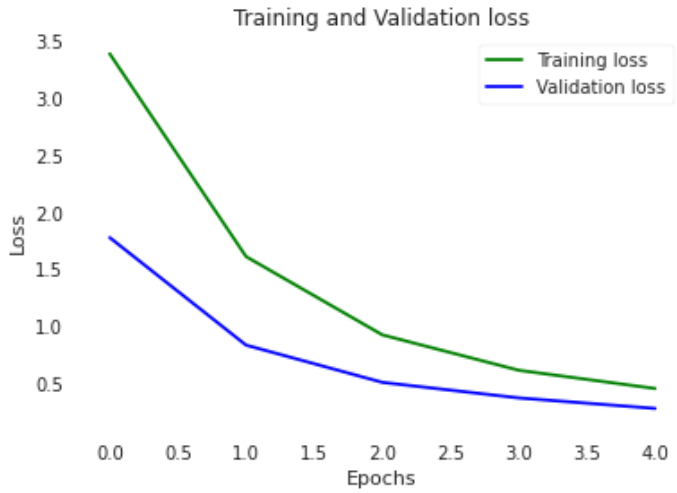
==========

Total params: 2,628,622

Trainable params: 2,628,622

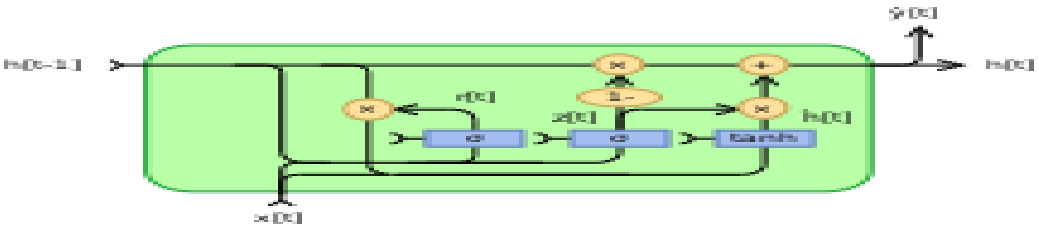
Non-trainable params: 0



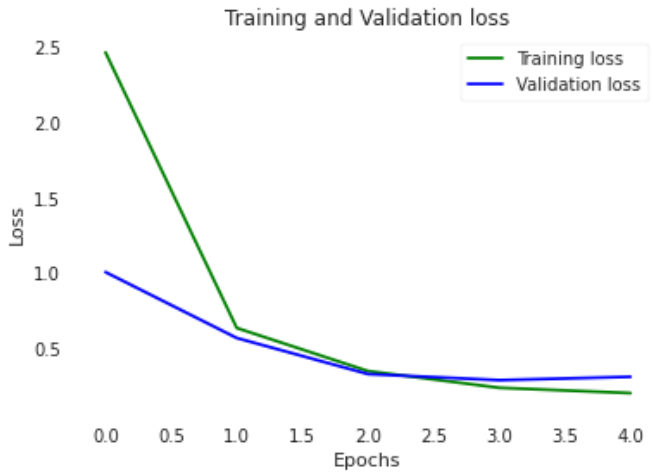


3. GRU (Gated Recurrent Unit):

Gated Recurrent Unit (GRU) is a gating mechanism for RNN.GRU is a simplified variant of the LSTM architecture, but there are differences as follows: GRU contains two gates and does not possess any internal memory.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 1000, 200)         2107000
_________________________________________________________________
gru (GRU)                    (None, 1000, 128)         126720
_________________________________________________________________
dense (Dense)                (None, 1000, 100)         12900
_________________________________________________________________
dropout (Dropout)            (None, 1000, 100)         0
_________________________________________________________________
flatten (Flatten)            (None, 100000)            0
_________________________________________________________________
dense_1 (Dense)              (None, 74)                7400074
=================================================================
Total params: 9,646,694
Trainable params: 9,646,694
Non-trainable params: 0
```



Round –2 Hyperparameter Tuning for Traditional & Sequential Models

For Hyperparameter Tuning, both GridSearchCV and RandomizedSearchCV options were tried. With GridSearchCV taking more time approximately 50 mins for each model given the size of the dataset and limited resources (RAM limitation colab), RandomizedSearchCV was the best suited option.

Generic function to run Traditional Models with RandomizedSearchCV

```
def hyperparameter_tune(base_model, parameters, n_iter, kfold, X_train, y_train,isNotTraditional,preTasks=True):
  k = StratifiedKFold(n_splits=kfold, shuffle=False)
  if preTasks:
    classifer = Pipeline([('tfidf',TfidfVectorizer(min_df=5 ,use_idf=True,analyzer='word', token_pattern=r'\w{1,}', max_features=50
                  ('classifer', base_model)])
    #print(classifer.get_params().keys())
  optimal_model = RandomizedSearchCV(classifer,
                           param_distributions=parameters,
                           n_iter=n_iter,
                           cv=k,
                           n_jobs=-1,
                           random_state=42)
  if isNotTraditional :
    print("NA")
  else:
    optimal_model.fit(X_train, y_train)
    scores = cross_val_score(optimal_model, X_train, y_train, cv=k, scoring="accuracy")
    print("Cross Val Mean: {:.3f}, Cross Val Stdev: {:.3f}".format(scores.mean(), scores.std()))
    print("Best Score: {:.3f}".format(optimal_model.best_score_))
    print("Best Parameters: {}".format(optimal_model.best_params_))
    return optimal_model.best_params_, optimal_model.best_score_
```

1. Multinomial Naïve Bayes

```
#Multinomial NaiveBayes
timer = Timer()
mnb_clf = MultinomialNB()
parameters = {'classifer__alpha': [0.2,0.25,0.3],'classifer__fit_prior': [True,False]}
scores = []
folds = range(2, 20)
for i in folds:
    print("\ncv = ", i)
    timer.start()
    best_params, best_score = hyperparameter_tune(mnb_clf, parameters, 2, i, X_resampled_train, y_resampled_train,False)
    scores.append(best_score)
print(timer.stop(), 'time taken to run the model with hyperparameter tuning')
```

Best Score: 0.911

Best Parameters: {'classifer__fit_prior': False, 'classifer__alpha': 0.2}

591.425673s time taken to run the model with hyperparameter tuning

2. RandomForest Classifier

```
#RandomForestClassifier
forest_clf = RandomForestClassifier()
parameters = {"classifer__max_depth": [3, 5, 10, None],"classifer__n_estimators": [100, 200, 300, 400, 500],
              "classifer__max_features": randint(1, 3),"classifer__criterion": ["gini", "entropy"],
              "classifer__bootstrap": [True, False],"classifer__min_samples_leaf": randint(1, 4)}
scores = []
folds = range(2, 11)
for i in folds:
    print("\ncv = ", i)
    best_params, best_score = hyperparameter_tune(forest_clf, parameters, 2, i, X_resampled_train, y_resampled_train,False)
    scores.append(best_score)
```

Best Score: 0.640

Best Parameters: {'classifer__bootstrap': True, 'classifer__criterion': 'gini', 'classifer__max_depth': 5, 'classifer__max_features': 1, 'classifer__min_samples_leaf': 3, 'classifer__n_estimators': 300}

3600s time taken to run the model with hyperparameter tuning

3. DecisionTree Classifier

```
#DecisionTreeClassifier
timer = Timer()
tree_clf = DecisionTreeClassifier()
parameters = {"classifer__max_depth": [3, None],"classifer__max_features": randint(1, 9),
              "classifer__min_samples_leaf": randint(1, 9),"classifer__criterion": ["gini", "entropy"]}
scores = []
folds = range(2, 30)
for i in folds:
    print("\ncv = ", i)
    timer.start()
    best_params, best_score = hyperparameter_tune(tree_clf, parameters, 2, i, X_resampled_train, y_resampled_train,False)
    scores.append(best_score)
print(timer.stop(), 'time taken to run the model with hyperparameter tuning')
```

Best Score: 0.421

Best Parameters: {'classifer__criterion': 'gini', 'classifer__max_depth': None, 'classifer__max_features': 5, 'classifer__min_samples_leaf': 5}

1194.578216s time taken to run the model with hyperparameter tuning

4. SVM Classifier

```
#SVM Classifier
timer = Timer()
svm_clf = LinearSVC()
parameters = {'classifer__C': [0.1, 1, 10, 100, 1000],
              'classifer__loss': ['hinge','squared_hinge'],
              'classifer__penalty': ['l1','l2'], 'classifer__multi_class': ['ovr','crammer_singer'],
              'classifer__fit_intercept': [True,False],'classifer__class_weight': ['balanced']}
scores = []
folds = range(2, 20)
for i in folds:
    print("\ncv = ", i)
    timer.start()
    best_params, best_score = hyperparameter_tune(svm_clf, parameters, 2, i, X_resampled_train, y_resampled_train,False)
    scores.append(best_score)
print(timer.stop(), 'time taken to run the model with hyperparameter tuning')
```

Best Score: 0.941

Best Parameters: {'classifer__penalty': 'l1', 'classifer__multi_class': 'crammer_singer', 'classifer__loss': 'squared_hinge', 'classifer__fit_intercept': False, 'classifer__class_weight': 'balanced', 'classifer__C': 1}

7200s time taken to run the model with hyperparameter tuning.

5. SGD Classifier

```
#SGD Classifier
timer = Timer()
sgd = SGDClassifier()
parameters = {"classifer__loss": ['hinge', 'log', 'modified_huber', 'squared_hinge','perceptron'],
              "classifer__penalty": ['l1', 'l2', 'elasticnet'],"classifer__alpha": [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],
              "classifer__learning_rate": ['constant', 'optimal', 'invscaling', 'adaptive'],
              "classifer__class_weight": [{1:0.5, 0:0.5}, {1:0.4, 0:0.6}, {1:0.6, 0:0.4}, {1:0.7, 0:0.3}],
              "classifer__eta0" : [1, 10, 100] }
scores = []
folds = range(2, 20)
for i in folds:
    print("\ncv = ", i)
    timer.start()
    best_params, best_score = hyperparameter_tune(sgd, parameters, 2, i, X_resampled_train, y_resampled_train,False)
    scores.append(best_score)
print(timer.stop(), 'time taken to run the model with hyperparameter tuning')
```

Best Score: 0.647

Best Parameters: {'classifer__penalty': 'elasticnet', 'classifer__loss': 'log', 'classifer__learning_rate': 'optimal', 'classifer__eta0': 100, 'classifer__class_weight': {1: 0.5, 0: 0.5}, 'classifer__alpha': 0.001}

Approximately 3600s for 5 iterations

6. Boosting Classifier

```
#Boosting Classifier
timer = Timer()
boost_clf = GradientBoostingClassifier()
parameters = {"classifer__loss": ['deviance'], "classifer__learning_rate": [0.01, 0.1, 0.15, 0.2],
              "classifer__max_depth":[3,5,8], "classifer__max_features":['log2','sqrt','auto'],
              "classifer__criterion": ['friedman_mse','mse'],"classifer__n_estimators":[50,100]}
scores = []
folds = range(2, 20)
for i in folds:
    print("\ncv = ", i)
    timer.start()
    best_params, best_score = hyperparameter_tune(boost_clf, parameters, 2, i, X_resampled_train, y_resampled_train,False)
    scores.append(best_score)
print(timer.stop(), 'time taken to run the model with hyperparameter tuning')
```

Best Score: 0.874

Best Parameters: {'classifer__n_estimators': 100, 'classifer__max_features': 'log2', 'classifer__max_depth': 3, 'classifer__loss': 'deviance', 'classifer__learning_rate': 0.1, 'classifer__criterion': 'friedman_mse'}

Limitation: - RAM got crashed and couldn't run for more iterations.

Sequential Models

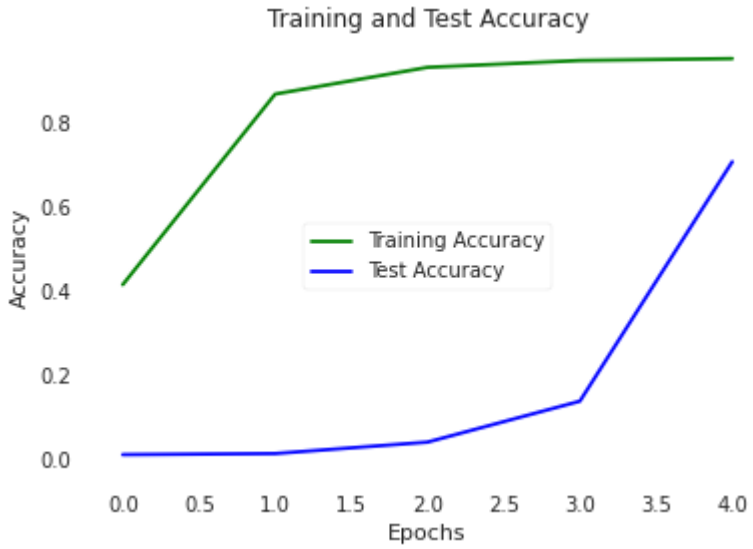
Approach: -

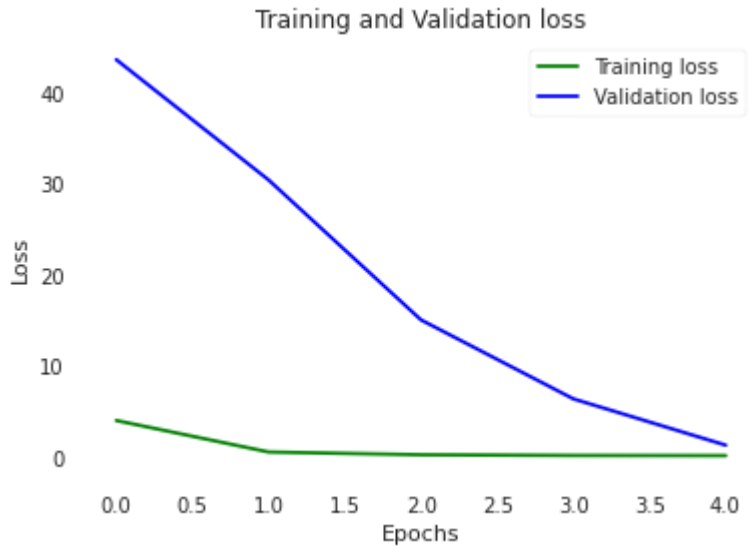
Changed the activation function from ReLU to Tanh and introduced BatchNormalization layer

1. CNN

```python
def build_CNN2(word_index,embeddings_matrix):
  model = Sequential(Embedding(len(word_index) + 1, embedding_dim, input_length=max_length, weights=[embeddings_matrix]))
  model.add(Conv1D(128, 3, padding='same', activation='tanh'))
  model.add(BatchNormalization())
  model.add(MaxPooling1D(3))
  model.add(Conv1D(64, 3, activation='tanh'))
  model.add(MaxPooling1D(3))
  model.add(Dropout(0.2))
  model.add(Flatten())
  model.add(Dense(74, activation='softmax'))
  optimizer=Adam(learning_rate=0.01)
  model.compile(loss='sparse_categorical_crossentropy',optimizer=optimizer,metrics=['accuracy'])
  print(model.summary())
  return model
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_7 (Embedding)      (None, 1000, 200)         2107000
_________________________________________________________________
conv1d_12 (Conv1D)           (None, 1000, 128)         76928
_________________________________________________________________
batch_normalization_5 (Batch (None, 1000, 128)         512
_________________________________________________________________
max_pooling1d_12 (MaxPooling (None, 333, 128)          0
_________________________________________________________________
conv1d_13 (Conv1D)           (None, 331, 64)           24640
_________________________________________________________________
max_pooling1d_13 (MaxPooling (None, 110, 64)           0
_________________________________________________________________
dropout_12 (Dropout)         (None, 110, 64)           0
_________________________________________________________________
flatten_7 (Flatten)          (None, 7040)              0
_________________________________________________________________
dense_16 (Dense)             (None, 74)                521034
_________________________________________________________________
Total params: 2,730,114
Trainable params: 2,729,858
Non-trainable params: 256
_________________________________________________________________
```



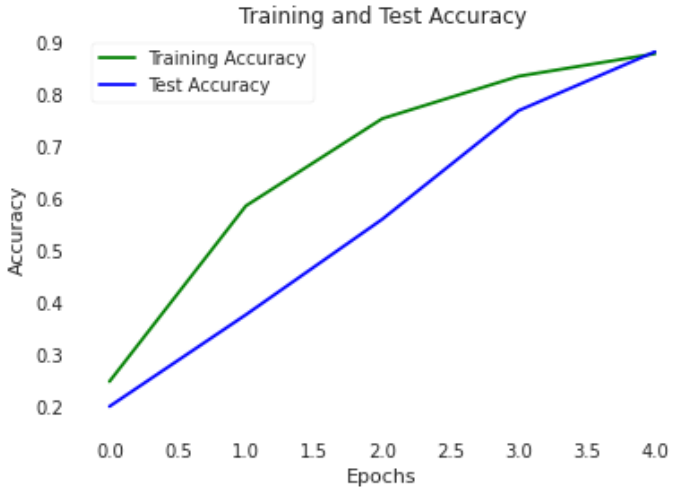


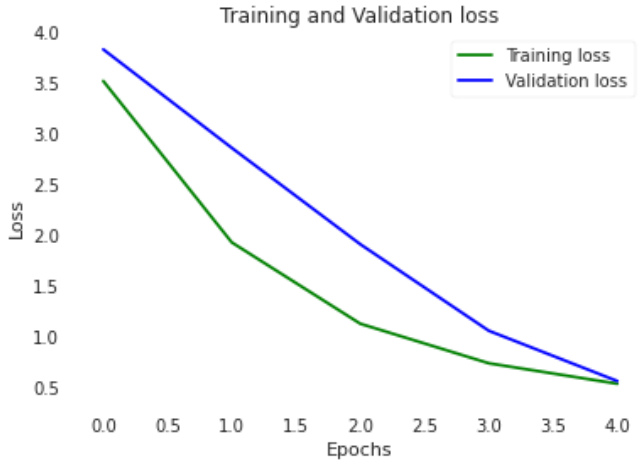
2. LSTM

```
def build_LSTM2(word_index,embeddings_matrix):
  model = Sequential(Embedding(len(word_index) + 1, embedding_dim, input_length=max_length, weights=[embeddings_matrix]))
  model.add(Bidirectional(LSTM(128,return_sequences=True)))
  model.add(BatchNormalization())
  model.add(Bidirectional(LSTM(64, recurrent_dropout=0.1)))
  model.add(Dropout(0.3))
  model.add(Dense(100,activation='tanh'))
  model.add(Dropout(0.4))
  model.add(Flatten())
  model.add(Dense(74,activation='softmax'))
  model.compile(loss='sparse_categorical_crossentropy',optimizer="adam",metrics=['accuracy'])
  print(model.summary())
  return model
```

Layer (type)	Output Shape	Param #
embedding_10 (Embedding)	(None, 1000, 200)	2107000
bidirectional_4 (Bidirection	(None, 1000, 256)	336896
batch_normalization_8 (Batch	(None, 1000, 256)	1024
bidirectional_5 (Bidirection	(None, 128)	164352
dropout_15 (Dropout)	(None, 128)	0
dense_21 (Dense)	(None, 100)	12900
dropout_16 (Dropout)	(None, 100)	0
flatten_10 (Flatten)	(None, 100)	0
dense_22 (Dense)	(None, 74)	7474

Total params: 2,629,646
Trainable params: 2,629,134
Non-trainable params: 512

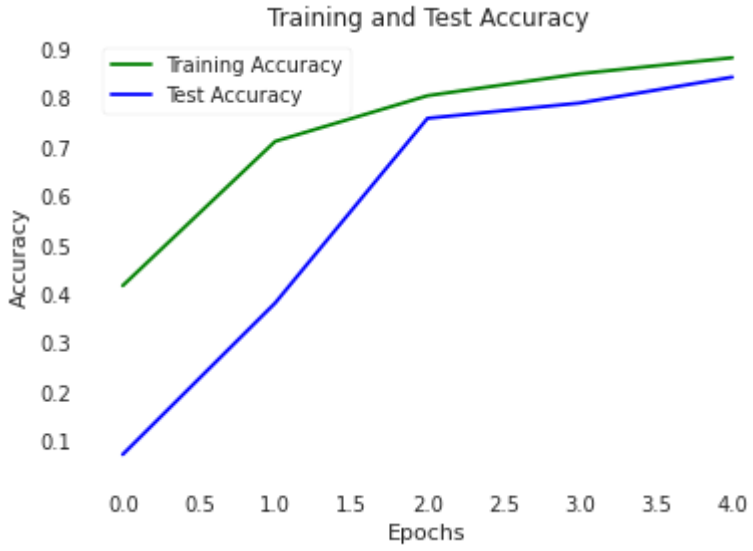


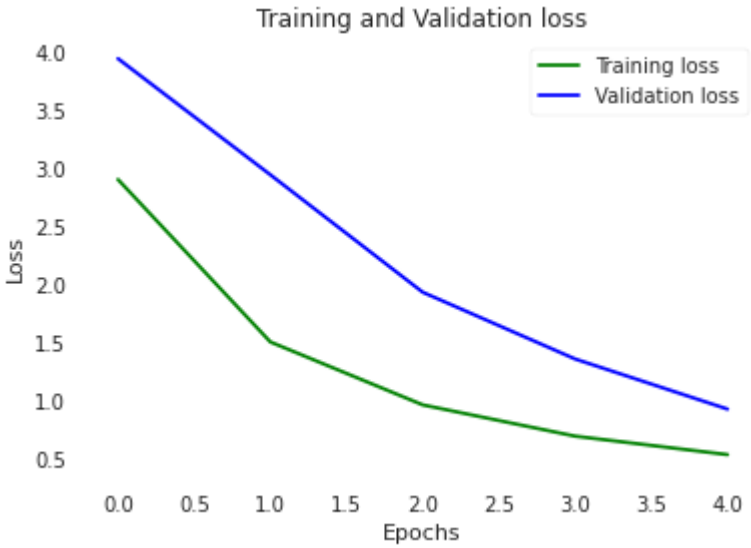


3. GRU

```
def build_GRU2(word_index,embeddings_matrix):
  model = Sequential(Embedding(len(word_index) + 1, embedding_dim, input_length=max_length, weights=[embeddings_matrix]))
  model.add(GRU(128,return_sequences=True))
  model.add(BatchNormalization())
  model.add(Dense(100,activation='tanh'))
  model.add(Dropout(0.4))
  model.add(Flatten())
  model.add(Dense(74,activation='softmax'))
  optimizer=Adam(learning_rate=0.0001)
  model.compile(loss='sparse_categorical_crossentropy',optimizer=optimizer,metrics=['accuracy'])
  print(model.summary())
  return model
```

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 1000, 200)         2107000
_________________________________________________________________
gru_1 (GRU)                  (None, 1000, 128)         126720
_________________________________________________________________
batch_normalization_1 (Batch (None, 1000, 128)         512
_________________________________________________________________
dense_2 (Dense)              (None, 1000, 100)         12900
_________________________________________________________________
dropout_1 (Dropout)          (None, 1000, 100)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 100000)            0
_________________________________________________________________
dense_3 (Dense)              (None, 74)                7400074
=================================================================
Total params: 9,647,206
Trainable params: 9,646,950
Non-trainable params: 256
```





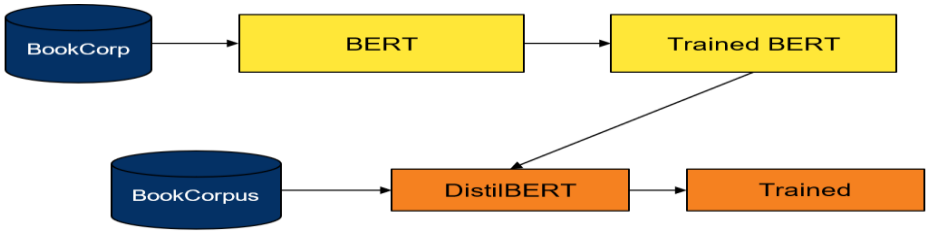
Evaluation metrics with Hyperparameter tuning:

	Classifier	Accuracy	F1-Score	Precision	Recall	Time taken
0	LSTM	0.920386	0.918289	0.934078	0.920386	3576.689095s
0	CNN	0.944270	0.941576	0.963528	0.944270	190.300719s
0	CNN2	0.339514	0.352679	0.610011	0.339514	166.363117s
0	CNN2	0.676828	0.682457	0.840091	0.676828	149.895712s
0	CNN2	0.584224	0.574763	0.886787	0.584224	150.283612s
0	CNN2	0.706474	0.693231	0.915346	0.706474	148.529204s
0	LSTM2	0.444689	0.440100	0.755324	0.444689	417.792489s
0	LSTM2	0.359941	0.379396	0.578241	0.359941	422.684039s
0	LSTM2	0.882464	0.881545	0.905843	0.882464	3559.554775s

	Classifier	Accuracy	F1-Score	Precision	Recall	Time taken
0	GRU2	0.843914	0.850955	0.882953	0.843914	222.286689s

Transfer Learning

We have used DistilBERT as a pretrained model. It is a distilled version of BERT: smaller, faster, cheaper and lighter



```
model_class, tokenizer_class, pretrained_weights = (ppb.DistilBertModel, ppb.DistilBertTokenizer, 'distilbert-base-uncased')
```

We are preparing features and labels based on DistilBERT and employed SVM as model on top to predict the assignment group

DistilBERT, used to create the features and labels using pretrained model gives an accuracy of **92.82%.**

This can be further improved by fine tuning the model based on our dataset and try again.

Cohen Kappa Benchmark Comparison

Cohen Kappa Benchmark

Kappa	Agreement	Interpretation
<0	Less than chance agreement	Poor
0.01–0.20	Slight agreement	Slight
0.21–0.40	Fair agreement	Fair
0.41–0.60	Moderate agreement	Moderate
0.61–0.80	Substantial agreement	Substantial
0.81–1.00	Almost perfect agreement	Almost perfect

Classifier	Accuracy	F1-Score	Precision	Recall	Cohen Kappa Score
Boosting Classifier - Round 2.pkl	0.836371	0.826056	0.884197	0.836371	0.834129
Bagging Classifier - Round 2.pkl	0.952965	0.950734	0.970131	0.952965	0.952317
DecisionTree Classifier - Round 2.pkl	0.953384	0.950845	0.970456	0.953384	0.952742
MultinomialNB Classifier - Round 2.pkl	0.902472	0.902598	0.921811	0.902472	0.901129
Random Forest Classifier - Round 2.pkl	0.957783	0.954622	0.975372	0.957783	0.957202
SGD Classifier - Round 2.pkl	0.869265	0.879354	0.891305	0.869265	0.867459
SVC Classifier - Round 2.pkl	0.948670	0.946304	0.965018	0.948670	0.947963
XgBoost Classifier - Round 2.pkl	0.956631	0.953569	0.974189	0.956631	0.956034
DISTILBERT_LinerSVC.pkl	0.928266	0.927417	0.947322	0.928266	0.927278

From above we see that the above classifiers based on the Cohen Kappa benchmark has an ***agreement*** of ***almost perfect*** and The ***interpretation*** is also ***almost perfect***.

Improvements and Limitations

Improvements:

1. We worked on balancing the classes with Random Over sampler and under sampler, SMOTE, Random Over sampler as Resampling strategies. We were unable to use SMOTE due to less number of resources available. In case we have high number of resources we can utilize smote to generate new synthetic records to treat class imbalance
2. We could use grid search, random search and do hyper parameter tuning and use more EPOCS etc.
3. With hyper parameter tuning, using Randomised search CV gave better results for traditional models in terms of the time consumed and performance compared to GridsearchCV, which was mote times consuming for execution.

4. Plotting the accuracy graphs has been performed
5. DistilBERT can be further improved by fine tuning the model based on our dataset and try again.

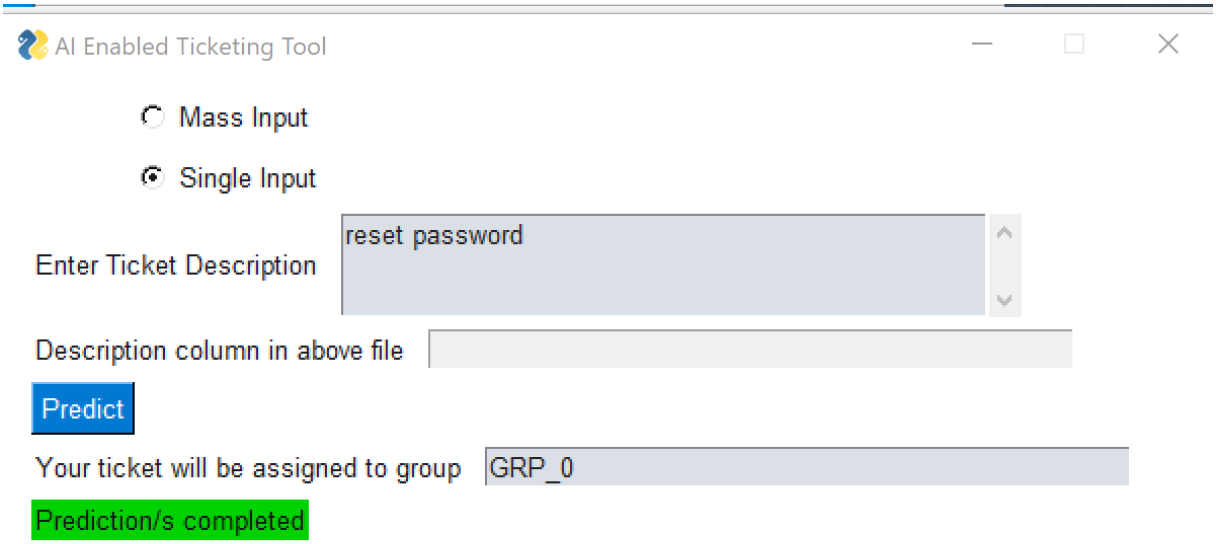
Limitations:

1. We also tried embedding implementations with focal loss as a loss function to handle the class imbalance problem, which helps in giving more weightage to groups with less samples, but the results were not satisfactory
2. So, while building models, we decided not to drop any groups instead use the resampling from utils package to oversample the groups which are less in numbers. We tried SMOTE, but due to high numbers of features and less resources we were unable to use it.
3. In our dataset, 'texts' are domain specific, and texts are quite rough in nature
4. Further dissecting, Resample was used and hence for classes with 2 tickets, they were balanced with duplicated records. There is contribution of 1.35 per group and there are 645 records per class.
5. During hyperparameter tuning there was an interruption due to cache or we had to interrupt process as the server got busy and was not responding. This was major hardware limitation throughout.

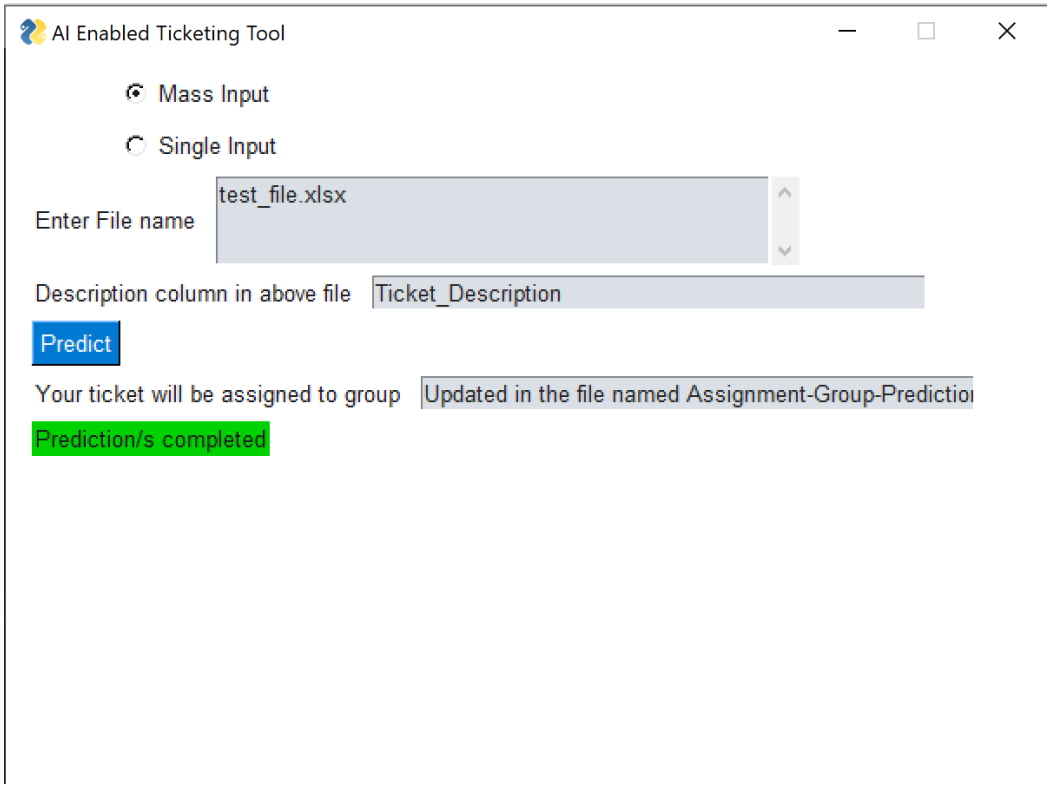
Graphical User Interface

This tool comes with the graphical user interface with below capabilities:

1. Single input – With single input, we can enter the ticket description and click on predict button, the UI will show the assignment group the ticket should be assigned to



2. Mass Input - With mass input, we can provide the file name and the column name this should consider to determine the assignment group. As this contains lot of rows a file will be pickled in the same directory with the corresponding groups for each ticket



The file has below format and can be loaded with below code:

```
with open('Assignment-Group-Predictions.pkl', 'rb') as f:
    predicted_group_df = pickle.load(f)
```

```
predicted_group_df
```

	Ticket Descriptions	Assignment Groups
0	received from: monitoring_tool@company.com ...	45
1	received from: monitoring_tool@company.com ...	45
2	reset password	0

Below model is used to determine the assignment group:

Random Forest Classifier - Round 2.p