Table of Contents

Summary of problem statement, data and findings 4

The Real Problem 4

Objective of the Project 4

Data and Findings ..... **Error! Bookmark not defined.**

2. Overview of the final process 6

3. Step-by-step walk through the solution 7

4. Model Evaluation 17

5. Comparison to benchmark ..... **Error! Bookmark not defined.**

6. Visualization 37

7. Implications 41

8. Limitations 41

9. Closing Reflections 42

1: Summary of problem statement, data and findings

The Real Problem

In the support process, incoming incidents are analyzed and assessed by organization's support teams to fulfill the request.

In many organizations, better allocation and effective usage of the valuable support resources will directly result in substantial cost savings.

Currently the incidents are created by various stakeholders (Business Users, IT Users and Monitoring Tools) within IT Service Management Tool and are assigned to Service Desk teams (L1 / L2 teams).

This team will review the incidents for right ticket categorization, priorities and then carry out initial diagnosis to see if they can resolve. Around ~54% of the incidents are resolved by L1 / L2 teams. In case L1 / L2 is unable to resolve, they will then escalate / assign the tickets to Functional teams from Applications and Infrastructure (L3 teams).

Some portions of incidents are directly assigned to L3 teams by either Monitoring tools or Callers / Requestors. L3 teams will carry out detailed diagnosis and resolve the incidents. Around ~56% of incidents are resolved by Functional / L3 teams. In case if vendor support is needed, they will reach out for their support towards incident closure. L1 / L2 needs to spend time reviewing Standard Operating Procedures (SOPs) before assigning to Functional teams (Minimum ~25-30% of incidents needs to be reviewed for SOPs before ticket assignment).

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

Objective of the Project

In this capstone project, the goal is to build a classifier that can classify the tickets by analyzing text, Guided by powerful AI techniques that can classify incidents to right functional groups can help organizations to reduce the resolving time of the issue and can focus on more productive tasks.

Data Findings:

```
In [ ]: input_data.shape
```

```
Out[ ]: (8500, 4)
```

```
In [ ]: input_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8500 entries, 0 to 8499
Data columns (total 4 columns):
Short description    8492 non-null object
Description          8499 non-null object
Caller               8500 non-null object
Assignment group     8500 non-null object
dtypes: object(4)
memory usage: 265.8+ KB
```

The input Excel File contains a total 8500 records consisting of the Short Description, Description, Caller Name and the group assigned as column values.

2. Overview of the final process

Observations from Target Class:

● The Target class distribution is extremely skewed

● A large no of entries for GRP_0 (mounting to 3976) which account for ~50% of the data

● There are groups with 1 entry also. We could merge all groups with small entries to a group to reduce the imbalance in the target. This may reduce the imbalance to some extent.

Data Pre-processing:

Below steps have been performed for initial pre-processing and clean-up of data.

● Dropped the caller field as the data was not found to be useful for analysis
● Replaced Null values in Short description & description with space.
● Merged Short Description & Description fields for analysis
● Contraction words found in the merged Description are removed for ease of word modelling
● Changed the case sensitivity of words to the common one
● Removed Hashtags and kept the words, Hyperlinks, URLs, HTML

Algorithms used:

We have tried below pre-modelling and modelling techniques

1. Built Bi-gram and Tri-gram models from the bag of words
2. Glove Embedding
3. Bidirectional LSTM Models using both embedding and compared
4. LSTM model
5. Attention model
6. Random Forest classifier
7. SVM Classifier model
8. Decision Tree

3. Step-by-step walk through the solution

Data pre-processing steps:

1. **Handling the Null values in Short Description and Description columns:**

```
input_data.isnull().sum()
```

```
Short description    8
Description          1
Caller               0
Assignment group     0
dtype: int64
```

```
input_data[pd.isnull(input_data).any(axis=1)]
```

	Short description	Description	Caller	Assignment group
2604	NaN	\r\n\r\nreceived from: ohdrnswl.rezuibdt@gmail...	ohdrnswl rezuibdt	GRP_34
3383	NaN	\r\n-connected to the user system using teamvi...	qftpazns fxpnytmk	GRP_0
3906	NaN	-user unable tologin to vpn.\r\n-connected to...	awpcmsey ctdiuqwe	GRP_0
3910	NaN	-user unable tologin to vpn.\r\n-connected to...	rhwsmefo tvphyura	GRP_0
3915	NaN	-user unable tologin to vpn.\r\n-connected to...	hxripljo efzounig	GRP_0
3921	NaN	-user unable tologin to vpn.\r\n-connected to...	cziadygo veiosxby	GRP_0
3924	NaN	name:wvqgbdhm fwchqjor\nlanguage:\nbrowser:mic...	wvqgbdhm fwchqjor	GRP_0
4341	NaN	\r\n\r\nreceived from: eqmuniov.ehxkcbgj@gmail...	eqmuniov ehxkcbgj	GRP_0
4395	i am locked out of skype	NaN	viyglzfo ajtfzpkb	GRP_0

```
## Relacing Null 'Short description' values with 'Description'
input_data['Short description'] = input_data['Short description'].fillna(input_data['Description'])
```

2. Joining the 'Short description' and 'Description' as one column 'FullDescription':

```
# Total number of words in the corpus
tickets['FullDescription'] = tickets['Short description'].str.cat(tickets['Description'], sep=' ', na_rep=' ')
tickets.head()
```

	Short description	Description	Caller	Assignment group	FullDescription
0	login issue	-verified user details.(employee# & manager na...	spxjnwir pjlcoqds	GRP_0	login issue -verified user details.(employee# ...
1	outlook	\r\n\r\nreceived from: hmjdrvpb.komuaywn@gmail...	hmjdrvpb komuaywn	GRP_0	outlook \r\n\r\nreceived from: hmjdrvpb.komuay...
2	cant log in to vpn	\r\n\r\nreceived from: eylqgodm.ybqkwiam@gmail...	eylqgodm ybqkwiam	GRP_0	cant log in to vpn \r\n\r\nreceived from: eylq...
3	unable to access hr_tool page	unable to access hr_tool page	xbkucsvz gcpydteq	GRP_0	unable to access hr_tool page unable to access...
4	skype error	skype error	owlgqjme qhcozdfx	GRP_0	skype error skype error

3. **Handling Encoded Texts:**

```
def is_valid_unicode_str(text):
    if not sequence_weirdness(text):
        # nothing weird, should be okay
        return True
    try:
        text.encode('sloppy-windows-1252')
    except UnicodeEncodeError:
        # Not CP-1252 encodable, probably fine
        return True
    else:
        # Encodable as CP-1252, Mojibake alert level high
        return False
```

```
tickets['is_valid'] = tickets.iloc[:,-1:].applymap(is_valid_unicode_str)
```

```
tickets.query('is_valid == False').iloc[:,-2:]
```

	FullDescription	is_valid
98	password expiry tomorrow \n\nreceived from: ec...	False
115	server issues \r\n\r\nreceived from: bgqpotek....	False
123	mobile device activation from: tvcdfqgp nrbcqw...	False
163	æ'¤å›ž: ticket_no1564867 -- comments added \n\...	False
169	[urgent!!] delivery note creation request!! \...	False
...	...	...
8387	please review your recent ticketing_tool ticke...	False
8388	ç"µè„'å¼€æœºå¼€ä¸□å‡ºæ□¥ to å°□è´ºï¼Œæ—©ä¸Šç"µ...	False
8397	customer group enhanced field \r\n\r\nreceived...	False
8415	machine nÃ£o estÃ¡ funcionando i am unable to ...	False
8416	an mehreren pc`s lassen sich verschiedene prgr...	False

825 rows × 2 columns

- Total of 825 rows contain encoded values

```
tickets['FullDescription'] = tickets['FullDescription'].apply(fix_text)
```

```
tickets['FullDescription'][8388]
```

'电脑开机开不出来 to 小贺,早上电脑开机开不出来'

- The encoding issue is fixed and below we can see the text has successfully changed to Chinese language

4. Detecting language of each description in the dataset:

```
lang = []
for i in range(len(tickets['FullDescription'])):
    try:
        lang.append(detect(tickets['FullDescription'][i]))
    except:
        print("This row throws error:", i)
        lang.append('no')
tickets['lang'] = lang
```

- The language detected is stored in a new column named "lang"

```
len(tickets[~tickets["lang"].isin(['no','en'])])
```

```
1348
```

- A total of 1348 data is found in language other than English

5. Translation all the descriptions which are not in English

```
translator = Translator()
cd_trans = []
for desc in tickets.index:
    if (tickets['lang'][desc] == (['no','en'])):
        cd_trans.append(tickets['FullDescription'][desc])
    else :
        cd_trans.append(translator.translate(tickets['FullDescription'][desc], dest='en', src='auto').text)
tickets['TranslatedDescription'] = cd_trans
```

```
tickets.head()
```

	Caller	Assignment group	FullDescription	is_valid	lang	TranslatedDescription
0	spxjnwir pjlcoqds	GRP_0	login issue -verified user details.(employee# ...	True	[no, en]	login issue -verified user details.(employee# ...
1	hmjdrvpb komuaywn	GRP_0	outlook \n\nreceived from: hmjdrvpb.komuaywn@g...	True	en	outlook \n\nreceived from: hmjdrvpb.komuaywn@g...
2	eylqgodm ybqkwiam	GRP_0	cant log in to vpn \n\nreceived from: eylqgodm...	True	en	cant log in to vpn \n\nreceived from: eylqgodm...
3	xbkucsvz gcpydteq	GRP_0	unable to access hr_tool page unable to access...	True	en	unable to access hr_tool page unable to access...
4	owlgqjme qhcozdfx	GRP_0	skype error skype error	True	no	skype error skype error

```
tickets.iloc[[3710,3097,3875,4467,5714],[2,5,7,8]]
```

	FullDescription	TranslatedDescription
3710	电话故障 铸棒车间电话故障,39523850	Telephone breakdown: Telephone breakdown of ca...
3097	电脑硬盘故障,请求维修。 电脑硬盘故障,请求维修。	The computer hard disk is faulty, request repa...
3875	电脑无法连接公共盘,请帮我转给小贺 电脑无法连接公共盘,请帮我转给小贺	The computer cannot connect to the public disk...
4467	铸造车间电脑故障 铸造车间记录生产数据的电脑不能开机	Computer failure in the foundry shop The compu...
5714	主机不能开启 主机不能开启,电源灯正常,主机面板1、3指示灯常亮	The host cannot be turned on The host cannot b...

- The translated data is stored in new column named '**TranslatedDescription**'.

6. Removal of unwanted data from the 'TranslatedDescription' column and store the clean-text in 'CleanDescription' column:

```
def getRegexList():
    '''
    Adding regex list as per the given data set to flush off the unnecessary text

    '''
    regexList = []
    regexList += ['From:(.*)\r\n']  # from line
    regexList += ['Sent:(.*)\r\n']  # sent to line
    regexList += ['received from:(.*)\r\n']  # received data line
    regexList += ['received']  # received data line
    regexList += ['To:(.*)\r\n']  # to line
    regexList += ['CC:(.*)\r\n']  # cc line
    regexList += ['(.*)infection']  # footer
    regexList += ['\[cid:(.*)]']  # images cid
    regexList += ['https?:[^\]\n\r]+']  # https & http
    regexList += ['Subject:']
    regexList += ['[\w\d\-\_\.]+@[\w\d\-\_\.]+']  # emails are not required
    regexList += ['[0-9][\-0-90-9 ]+']  # phones are not required
    regexList += ['[0-9]+[\r\n]+" "']  # numbers, next line and space not needed
    regexList += ['[0-9]+[a-zA-Z]']  # single letters appened to numbers not needed
    regexList += ['[0-9]']  # numbers not needed
    regexList += ['[^a-zA-z 0-9]+']  # anything that is not a letter
    regexList += ['[\r\n]']  # \r\n
    regexList += [' [a-zA-Z] ']  # single letters makes no sense
    regexList += [' [a-zA-Z][a-zA-Z] ']  # two-letter words makes no sense
    regexList += ["  "]  # double spaces

    regexList += ['^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*(\.[a-z]{2,4})$']
    regexList += ['[\w\d\-\_\.]+ @ [\w\d\-\_\.]+']
    regexList += ['Subject:']
    regexList += ['[^a-zA-Z]']

    return regexList
```

```
:
def cleanDataset(dataset, columnsToClean, regexList):
    for column in columnsToClean:
        for regex in regexList:
            dataset[column] = dataset[column].str.lower().str.replace(regex.lower(), ' ')
```

7. Further cleaning the text and removing the ‘Caller’ names from description text:

```
callers_list = input_data['Caller'].unique()
callers_list
```

```
array(['spxjnwir pjlcoqds', 'hmjdrvpb komuaywn', 'eylqgodm ybqkwiam', ...,
       'bjitvswa yrmugfnq', 'oybwdsgx oxyhwrfz', 'kqvbrspl jyzoklfx'],
      dtype=object)
```

```
def removeCallerNames(indx, text):
    for callers in callers_list:
          callerlist = [xname for xname in callers.split()]
          for names in callerlist:
            text = text.replace(names,'')
    return text
```

```
for i, row in enumerate(cleaned_tickets.itertuples(), 0):
    cleaned_tickets['CleanDescription'].iloc[i]=removeCallerNames(i, row.CleanDescription)
```

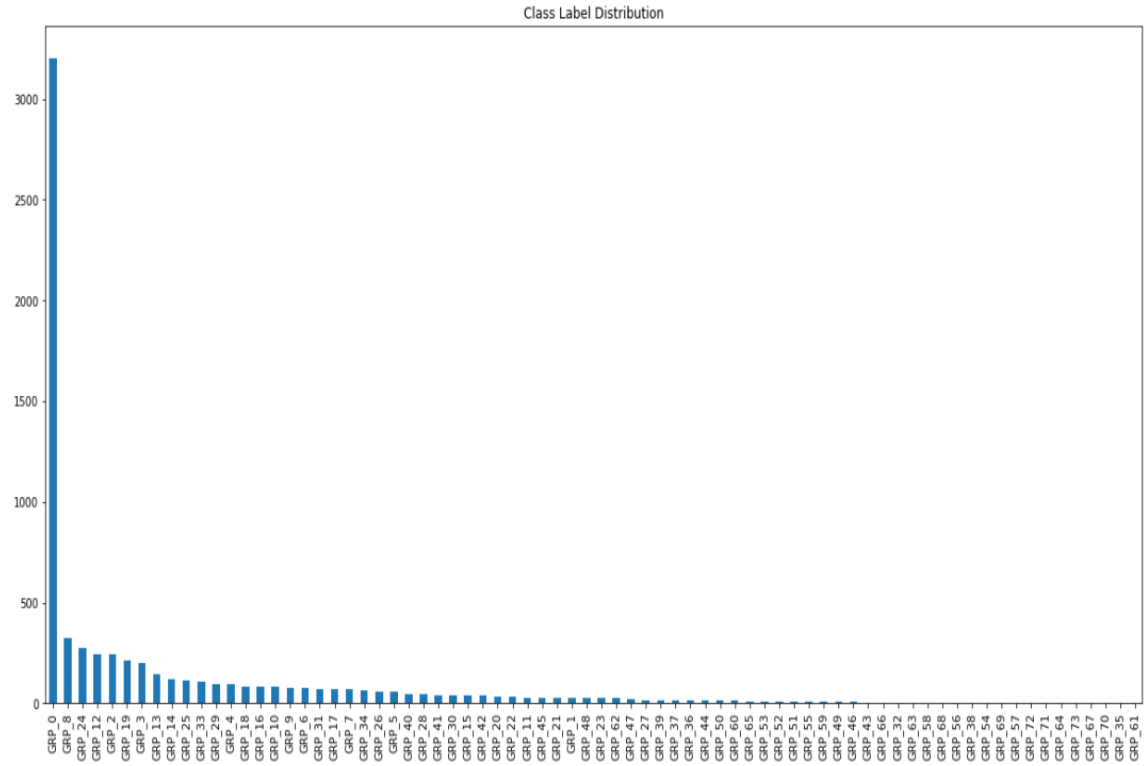
Salient features of your data

8. Visualizations:

- Plotting the 'Assignment group' data in bar chart using matplotlib library:

```
cleaned_tickets['Assignment group'].value_counts().sort_values(ascending=False).plot(kind='bar', figsize=(20,10), title='Class Label Distribution')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3947e7e1d0>
```



- There are 74 Assignment groups which is the Target class.

- The class imbalance is high as data is highly skewed towards ‘GRP_0’ occupying 46.78% of the data

```
group_df = pd.DataFrame.from_dict(group_frequency, orient='index', columns=['frequency'])
group_df['percent representation'] = group_df['frequency'] / group_df['frequency'].sum()
group_df['cumulative percentage'] = group_df['percent representation'].cumsum(axis = 0)
group_df.head(15)
```

	frequency	percent representation	cumulative percentage
GRP_0	3199	0.467895	0.467895
GRP_8	313	0.045780	0.513676
GRP_24	277	0.040515	0.554190
GRP_12	243	0.035542	0.589732
GRP_2	238	0.034811	0.624543
GRP_19	212	0.031008	0.655551
GRP_3	198	0.028960	0.684511
GRP_13	141	0.020623	0.705134
GRP_25	116	0.016967	0.722100
GRP_14	116	0.016967	0.739067
GRP_33	107	0.015650	0.754717
GRP_4	96	0.014041	0.768758
GRP_29	95	0.013895	0.782653
GRP_16	85	0.012432	0.795086
GRP_18	85	0.012432	0.807518

9. Processing on text data using NLTK and Regex library:

- Removing Numbers from the text:

```
def removeNumbers(text):
    """ Removes integers """
    text = ''.join([i for i in text if not i.isdigit()])
    return text
```

- Replacing Contractions with complete words:

```
contraction_patterns = [ (r'won\'t', 'will not'),(r'didn\'t', 'did not'),(r'didnt', 'did not'), (r'can\'t', 'cannot'),(r'cant',
'cannot'), (r'i\'m', 'i am'), (r'ain\'t', 'is not'), (r'(\w+)\'ll', '\g<1> will'), (r'(\w+)n\'t', '\g<1> not'),
                        (r'(\w+)\'ve', '\g<1> have'), (r'(\w+)\'s', '\g<1> is'), (r'(\w+)\'re', '\g<1> are'), (r'(\w+)\'d', '\g
<1> would'), (r'&', 'and'), (r'dammit', 'damn it'), (r'dont', 'do not'), (r'wont', 'will not') ]
def replaceContraction(text):
    patterns = [(re.compile(regex), repl) for (regex, repl) in contraction_patterns]
    for (pattern, repl) in patterns:
        (text, count) = re.subn(pattern, repl, text)
    return text
```

- Identify the words preceded by ‘Not’ and find the antonyms words for the same:

```
#Replace Negations with Antonym
def replace(word, pos=None):

    antonyms = set()
    for syn in wordnet.synsets(word, pos=pos):
        for lemma in syn.lemmas():
            for antonym in lemma.antonyms():
                antonyms.add(antonym.name())
    if len(antonyms) == 1:
        return antonyms.pop()
    else:
        return None
def replaceNegations(text):
    """ Finds "not" and antonym for the next word and if found, replaces not and the next word with the antonym """
    i, l = 0, len(text)
    words = []
    while i < l:
        word = text[i]
        if word == 'not' and i+1 < l:
            ant = replace(text[i+1])
            if ant:
                words.append(ant)
                i += 2
                continue
        words.append(word)
        i += 1
    return words

def antonym(text):
    tokens = nltk.word_tokenize(text)
    tokens = replaceNegations(tokens)
    text = " ".join(tokens)
    return text
```

- Remove the frequently used ‘Stopwords’ to increase the efficiency of the model to identify the keywords required for classification:

```
#Remove Stopwords
stoplist = stopwords.words('english')
stoplist.remove('no')
stoplist.remove('not')
def stp_words(text):
    finalTokens = []
    tokens = nltk.word_tokenize(text)
    for w in tokens:
        if (w not in stoplist):
            finalTokens.append(w)
    text = " ".join(finalTokens)
    return text
```

- Removal of the mail related words which are not significant to classify the incidents to the respective groups:

```
#Remove mail related words
mail_words_list = ['hi','hello','com','gmail','cc','regards','thanks']
def mail_words(text):
    finalTokens = []
    tokens = nltk.word_tokenize(text)
    for w in tokens:
        if (w not in mail_words_list):
            finalTokens.append(w)
    text = " ".join(finalTokens)
    return text
```

- Lemmatization of the text:

```
#Lemmatization
stemmer = PorterStemmer() #set stemmer
lemmatizer = WordNetLemmatizer() # set lemmatizer

def nltk_tag_to_wordnet_tag(nltk_tag):
    if nltk_tag.startswith('J'):
        return wordnet.ADJ
    elif nltk_tag.startswith('V'):
        return wordnet.VERB
    elif nltk_tag.startswith('N'):
        return wordnet.NOUN
    elif nltk_tag.startswith('R'):
        return wordnet.ADV
    else:
        return None
```

```
def lemmatize_sentence(sentence):
    #tokenize the sentence and find the POS tag for each token
    nltk_tagged = nltk.pos_tag(nltk.word_tokenize(sentence))
    #tuple of (token, wordnet_tag)
    wordnet_tagged = map(lambda x: (x[0], nltk_tag_to_wordnet_tag(x[1])), nltk_tagged)
    lemmatized_sentence = []
    for word, tag in wordnet_tagged:
        if tag is None:
            #if there is no available tag, append the token as is
            lemmatized_sentence.append(word)
        else:
            #else use the tag to lemmatize the token
            lemmatized_sentence.append(lemmatizer.lemmatize(word, tag))
    return " ".join(lemmatized_sentence)
```

10. **The final text is stored in 'Clean Description - After':**

```
    #create new column and add updated data
    cleaned_tickets.at[index, 'Clean Description - After']= proc_des
```

```
cleaned_tickets.head()
```

	CleanDescription	Assignment group	Clean Description - After
0	login issue verified user details employee ma...	GRP_0	login issue verify user detail employee manage...
1	outlook from hello team meetings skype meeti...	GRP_0	outlook team meeting skype meeting etc not app...
2	cant log to vpn from cannot log to vpn best	GRP_0	not log vpn not log vpn best
3	unable access tool page unable access tool page	GRP_0	unable access tool page unable access tool page
4	skype error skype error	GRP_0	skype error skype error

4. Model Evaluation

❖ Basic Model building using Unsupervised Learning Models:

Step 1: Tokenisation

```
from tensorflow.keras.preprocessing.text import Tokenizer
MAX_NB_WORDS = 10000
t = Tokenizer(num_words=MAX_NB_WORDS, split=' ', char_level=False, oov_token=None, document_count=0)
# t = Tokenizer(num_words=MAX_NB_WORDS)
t.fit_on_texts(cleaned_tickets['Clean Description - After'])
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(cleaned_tickets['Clean Description - After'])
```

Step 2: Train-Test Split

- Spliting the data set into 75:25 ratio with target as 'Assignment group'

```
train_x, test_x, train_y, test_y = model_selection.train_test_split(X, cleaned_tickets['Assignment group'],
                                                                    test_size=0.25, random_state=42)
```

Step 3: Model Building

```
models=[]
models.append(('LogisticRegression',LogisticRegression(solver = 'liblinear')))
models.append(('knn',KNeighborsClassifier()))
models.append(('SVC',SVC(gamma = 'auto', kernel= 'poly', degree=2)))
models.append(('Naive Bayes',MultinomialNB()))
models.append(("Decision_tree",DecisionTreeClassifier(criterion = 'entropy')))
models.append(("Random_Forest",RandomForestClassifier(criterion = 'entropy')))
models.append(("Ada_Boost",AdaBoostClassifier( n_estimators= 20)))
models.append(("Bagging",BaggingClassifier(n_estimators=74, max_samples= .7, bootstrap=True)))
models.append(("Gradient_Boosting",GradientBoostingClassifier(n_estimators = 10, learning_rate = 0.05)))
```

Step 4: Fit and evaluate the models

- Fitting into the model

```
def fit_n_print(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
    score = model.score(X_test, y_test)
    return score  # return accuracy metrics
```

```
result1 = pd.DataFrame({'Model':[],'Accuracy':[]})
for name,model in models:
    accuracy = fit_n_print(model, train_x, test_x, train_y, test_y )
    result1 = result1.append(pd.Series({'Model':name, 'Accuracy':accuracy}), ignore_index=True)
print(result1)
```

```
                Model  Accuracy
0  LogisticRegression  0.567130
1                 knn  0.534722
2                 SVC  0.467014
3         Naive Bayes  0.505208
4       Decision_tree  0.491319
5       Random_Forest  0.568287
6           Ada_Boost  0.484954
7             Bagging  0.592014
8   Gradient_Boosting  0.524306
```

Observation¶

- There is no much difference in accuracy even after tweaking the hyperparameters
- All the models are having accuracy between 46% to 59%
- As the problem statement is to classify the text given as description, it is recommended to use deep learning (neural network) techniques to build the model and enhance the accuracy of determining the groups effectively.

❖ Enhanced modelling using Deep Learning Techniques:

Language modelling

Language modelling is a special case of classification where the model is trained to predict the next word or character in the sentence. At each time step t, the output vector gives the probability distribution of x_t over all the words/characters in the vocabulary, conditioned on the previous words/characters in the sequence,

The language model can also be used to generate text of arbitrary size by repeatedly sampling characters for the desired number of time steps (for character-level granularity) or until the special end-of-sentence token is selected (for word-level granularity).
For a character-level language model for instance, T can easily exceed 20 or 25.
This greatly amplifies the adverse effects of the well-known vanishing and exploding gradients problem, which prevents long-range dependencies from being learned. Note that this issue can also be experienced with feed-forward neural networks, such as the Multi-Layer Perceptron, but it just gets
worse with RNN due to their inherent tendency to be deep.

Pre-processing and data readiness:

Step 1: Re-naming of the columns and deleting the unused columns

```
df.head()
```

	Assignment group	ProcessedDescription
0	GRP_0	login issue verify user detail employee manage...
1	GRP_0	outlook team meeting skype meeting etc not app...
2	GRP_0	not log vpn not log vpn best
3	GRP_0	unable access tool page unable access tool page
4	GRP_0	skype error skype error

Step 2: Reconfirming that no null value is present

```
df[pd.isnull(df).any(axis=1)]
```

	Assignment group	ProcessedDescription
1556	GRP_0	NaN
2423	GRP_34	NaN

```
df.dropna(subset = ["ProcessedDescription"], inplace=True)
```

```
df.isna().sum()
```

```
Assignment group        0
ProcessedDescription    0
dtype: int64
```

Step 3: Check the required size for padding the text sequences

```
df.ProcessedDescription.apply(lambda x: len(x.split(" "))).mean()
```

```
23.47769489542197
```

Thus, padding size of 25 should be good fit.

Step 4: Tokenisation of the text data

```
words = []
for i in df.ProcessedDescription.values:
    words.append(i.split())
words[:1]
```

```
[['login',
  'issue',
  'verify',
  'user',
  'detail',
  'employee',
  'manager',
  'name',
  'check',
  'user',
  'name',
  'ad',
  'reset',
  'password',
  'advise',
  'user',
  'login',
  'check',
  'caller',
  'confirm',
  'able',
  'login',
  'issue',
  'resolve']]
```

```
tokenizer = text.Tokenizer(num_words=10000)
tokenizer.fit_on_texts(words)
tokenized_train = tokenizer.texts_to_sequences(words)
```

Step 5: Word Vocab and Indexing

```
tokenizer.word_index
```

```
{'not': 1,
 'please': 2,
 'password': 3,
 'yes': 4,
 'erp': 5,
 'tool': 6,
 'user': 7,
 'na': 8,
 'company': 9,
 'access': 10,
 'issue': 11,
 'work': 12,
 'unable': 13,
 'reset': 14,
 'error': 15,
 'email': 16,
 'need': 17,
 'sid': 18,
```

```
vocab_size = len(tokenizer.word_index) + 1
print(vocab_size)
```

```
10162
```

The vocab size id 10162, plus 1 is added as the counting starts from 0.

Step 6: Handling the imbalance of the data

```
groups = df.groupby(['Assignment group'])
misc_group=[]
for grp in df['Assignment group'].unique():
      if(groups.get_group(grp).shape[0]<100):
        misc_group.append(grp)
print('GRP_74 which have under 100 samples are {} '.format(len(misc_group)))
df['Assignment group - After']=df['Assignment group'].apply(lambda x : 'GRP_74' if x in misc_group else x)
```

```
GRP_74 which have under 100 samples are 63
```

```
df['Assignment group - After'].value_counts().sort_values(ascending=False).index
```

```
Index(['GRP_0', 'GRP_74', 'GRP_8', 'GRP_24', 'GRP_12', 'GRP_2', 'GRP_19',
       'GRP_3', 'GRP_13', 'GRP_14', 'GRP_25', 'GRP_33'],
      dtype='object')
```

All the groups which have records less than 100 are clubbed into one group named ‘GRP_74’.

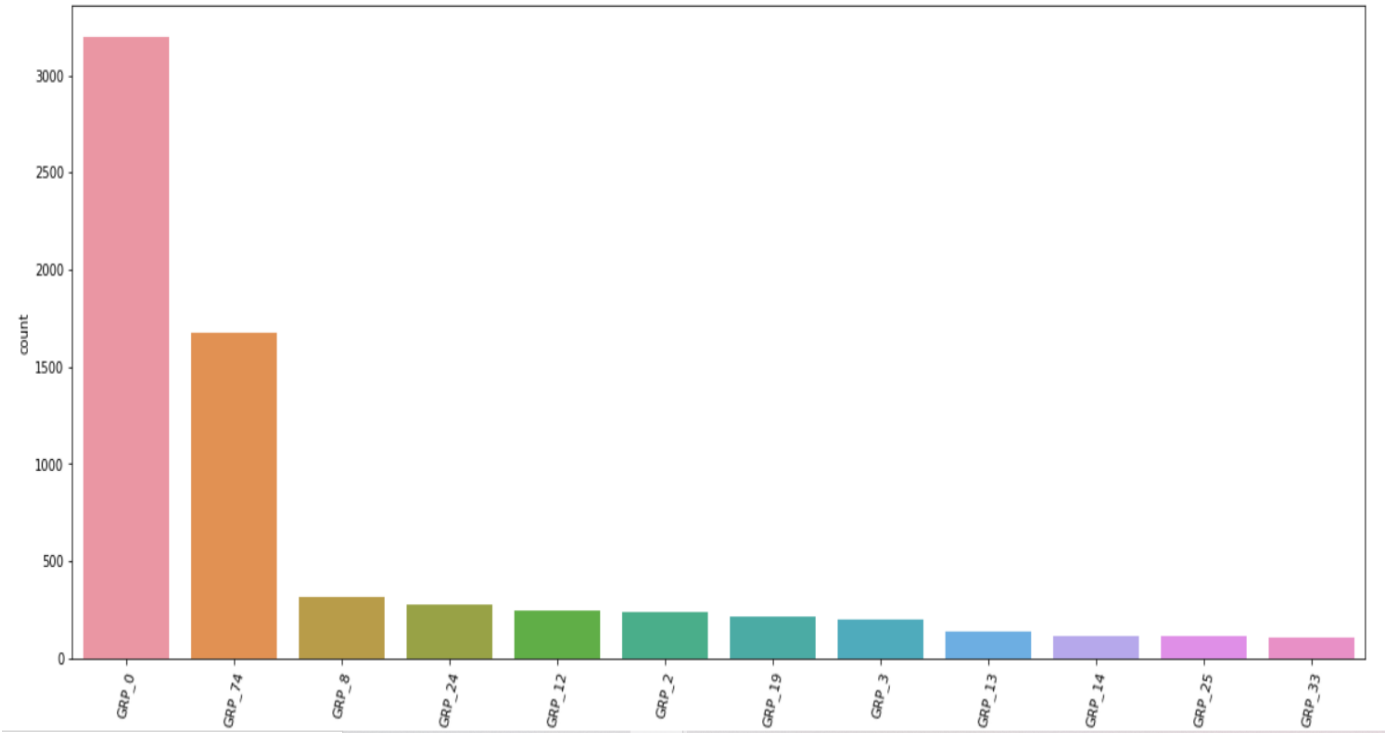
Step 7: Visualisation of the newly formed grouping structure

```
df['Assignment group - After'].value_counts()
```

```
GRP_0     3199
GRP_74    1677
GRP_8      313
GRP_24     277
GRP_12     243
GRP_2      238
GRP_19     212
GRP_3      198
GRP_13     141
GRP_25     116
GRP_14     116
GRP_33     107
Name: Assignment group - After, dtype: int64
```

```
df_desc = df['Assignment group - After'].value_counts().sort_values(ascending=False).index
plt.figure(figsize= (20,8))
plt.xticks(rotation=75)
sns.countplot(x='Assignment group - After', data=df, order=df_desc)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ebdd8c1d0>
```



- Now we have a total of 12 categorical data under ‘Assignment group’

Step 8: Label Encoding of the Targe Variable and converting into categorical data

```
encoder = LabelEncoder()
Y = df['Assignment group - After']
encoder.fit(Y)
encoded_Y = encoder.transform(Y)
# convert integers to dummy variables (i.e. one hot encoded)
dummy_y = to_categorical(encoded_Y)
```

Step 9: Train-validation-Test Split

```
from sklearn import model_selection
X_train_val, X_test, y_train_val, y_test = model_selection.train_test_split(X, dummy_y, test_size=0.25, random_state=42)
```

```
X_train, X_val, y_train, y_val = model_selection.train_test_split(X_train_val, y_train_val, test_size=0.3, random_state=42)
```

```
print("Size of the training dataframe is ",len(X_train))
print("Size of the Validation dataframe is ",len(X_val))
print("Size of the test dataframe is ",len(X_test))
print("{0:0.2f}% data is in training set".format((len(X_train)/len(df.index)) * 100))
print("{0:0.2f}% data is in training set".format((len(X_val)/len(df.index)) * 100))
print("{0:0.2f}% data is in test set".format((len(X_test)/len(df.index)) * 100))
```

```
Size of the training dataframe is  3588
Size of the Validation dataframe is  1539
Size of the test dataframe is  1710
52.48% data is in training set
22.51% data is in training set
25.01% data is in test set
```

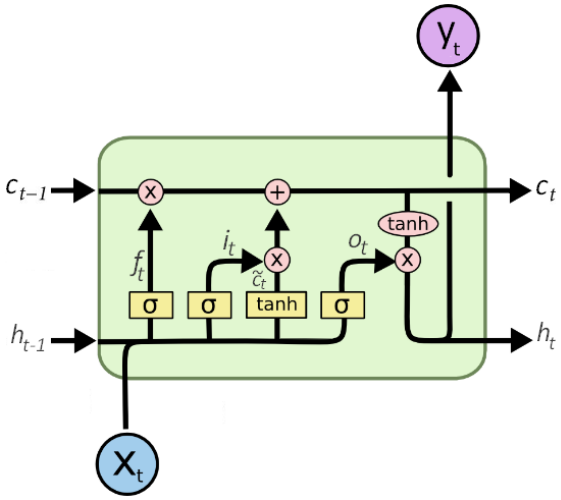
Model Building:

LSTM unit

In practice, whenever people use RNNs, they use the LSTM or the GRU unit (see next subsection), as these cells are engineered in a way that allows them to escape vanishing/exploding gradients and keep track of information over longer time periods.

As shown in Fig. 8, the two things that change in the LSTM unit compared to the basic RNN unit are (1) the presence of a cell state (c_t), which serves as an explicit memory, and (2) how hidden states are computed.

With vanilla RNNs, the hidden state is computed with a single layer as $h_t = \tanh(Ux_t+Wh_{t-1}+b)$ (see eq. 6). With the LSTM unit however, the hidden state is computed by four interacting layers that give the network the ability to remember or forget specific information about the preceding elements in the sequence.



The LSTM unit. Adapted from Chris Colah's blog.

- **Model 1: LSTM with 2 Dense layers**

Step 1: Model Structure

```
lstm_model_4 = Sequential()
lstm_model_4.add(Embedding(vocab_size, output_dim=200, input_length=25, trainable=True))
lstm_model_4.add(Dropout(0.5))
lstm_model_4.add(LSTM(128, recurrent_dropout = 0.5 , dropout = 0.5))
lstm_model_4.add(Dense(64, activity_regularizer=l1(0.01)))
lstm_model_4.add(LeakyReLU(alpha=0.3))
lstm_model_4.add(Dropout(0.5))
lstm_model_4.add(Dense(12, activity_regularizer=l1(0.01),activation='softmax'))
lstm_model_4.compile(loss = 'categorical_crossentropy', optimizer=adam_optimizer,metrics = ['accuracy'])
print(lstm_model_4.summary())
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_14 (Embedding)     (None, 25, 200)           2032400
_________________________________________________________________
dropout_10 (Dropout)         (None, 25, 200)           0
_________________________________________________________________
lstm_11 (LSTM)               (None, 128)               168448
_________________________________________________________________
dense_17 (Dense)             (None, 64)                8256
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 64)                0
_________________________________________________________________
dropout_11 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_18 (Dense)             (None, 12)                780
=================================================================
Total params: 2,209,884
Trainable params: 2,209,884
Non-trainable params: 0
_________________________________________________________________
None
```

Step 2: Define Optimizer

```
adam_optimizer = Adam(lr=1e-2, beta_1=0.09, beta_2=0.999, amsgrad=True)

lstm_model_4.compile(loss='categorical_crossentropy',
              optimizer=adam_optimizer,
              metrics='accuracy')
```

Step 3: Train and Fit the Model

```
BATCH_SIZE = 32
EPOCHS = 20
stop = EarlyStopping(monitor="loss", patience=5, mode="min")
reduce_lr = ReduceLROnPlateau(monitor="loss", factor=0.2, patience=5, min_lr=1e-6, verbose=1, mode="min")

history = lstm_model_4.fit(X_train_val, y_train_val, validation_split=0.5, epochs=EPOCHS, batch_size=BATCH_SIZE, verbose=1, shuffle=True, callbacks=[stop, reduce_lr])
```

```
Epoch 1/20
81/81 [==============================] - 9s 109ms/step - loss: 1.7935 - accuracy: 0.5205 - val_loss: 4.9285 - val_accuracy: 0.1026
Epoch 2/20
81/81 [==============================] - 9s 106ms/step - loss: 1.4216 - accuracy: 0.6243 - val_loss: 1.4928 - val_accuracy: 0.6112
Epoch 3/20
81/81 [==============================] - 8s 105ms/step - loss: 1.0856 - accuracy: 0.7401 - val_loss: 1.5457 - val_accuracy: 0.6127
Epoch 4/20
81/81 [==============================] - 9s 105ms/step - loss: 0.9192 - accuracy: 0.7827 - val_loss: 1.5986 - val_accuracy: 0.6022
Epoch 5/20
81/81 [==============================] - 9s 105ms/step - loss: 0.7937 - accuracy: 0.8357 - val_loss: 2.1976 - val_accuracy: 0.5885
Epoch 6/20
81/81 [==============================] - 9s 106ms/step - loss: 0.7836 - accuracy: 0.8365 - val_loss: 1.8982 - val_accuracy: 0.6022
Epoch 7/20
81/81 [==============================] - 9s 106ms/step - loss: 0.7291 - accuracy: 0.8451 - val_loss: 2.1652 - val_accuracy: 0.5819
Epoch 8/20
81/81 [==============================] - 9s 105ms/step - loss: 0.7471 - accuracy: 0.8584 - val_loss: 2.0851 - val_accuracy: 0.6108
Epoch 9/20
81/81 [==============================] - 9s 106ms/step - loss: 0.7408 - accuracy: 0.8564 - val_loss: 2.2000 - val_accuracy: 0.5952
Epoch 10/20
81/81 [==============================] - 9s 105ms/step - loss: 0.7546 - accuracy: 0.8529 - val_loss: 2.2990 - val_accuracy: 0.5924
Epoch 11/20
81/81 [==============================] - 9s 106ms/step - loss: 0.7396 - accuracy: 0.8541 - val_loss: 2.2466 - val_accuracy: 0.6057
Epoch 12/20
81/81 [==============================] - ETA: 0s - loss: 0.7478 - accuracy: 0.8541
Epoch 00012: ReduceLROnPlateau reducing learning rate to 0.0019999999552965165.
81/81 [==============================] - 9s 106ms/step - loss: 0.7478 - accuracy: 0.8541 - val_loss: 2.4501 - val_accuracy: 0.6104
```

Step 4: Evaluate the Model on Test data

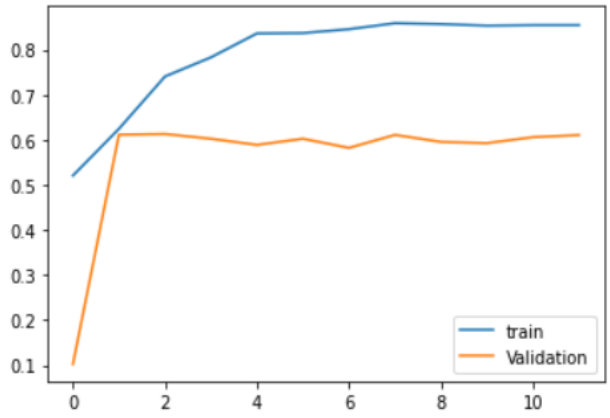
```
loss, accuracy = lstm_model_4.evaluate(X_test, y_test)
print('Accuracy: %f' % (accuracy*100))
print('loss: %f' % (loss))
```

```
54/54 [==============================] - 0s 8ms/step - loss: 2.3069 - accuracy: 0.6333
Accuracy: 63.333333
loss: 2.306902
```

Step 6: Plot the accuracy of train and validation data

```
from matplotlib import pyplot

pyplot.plot(history.history['accuracy'], label='train')
pyplot.plot(history.history['val_accuracy'], label='Validation')
pyplot.legend()
pyplot.show()
```



Observation:

- The training loss has gone down to 74.78% but validation loss is still as high as 245.01%
- Although the training set was able to learn and the accuracy has reached 85.41% but the validation failed to reach that level and accuracy reached was 61.04%
- Compared to validation data, performance of test data was better with loss as 230% and accuracy as 63.33%

- **Model 2: LSTM with pre-defined Embedding weights of dimension 200**

Embedding Layer

As all our Machine Learning and Deep learning algorithms are incapable of processing strings or plain text in their raw form, word embeddings are used to convert the texts into numbers. There may be different numerical representations of the same text. It tries to map a word using a dictionary to a vector.

We have experimented below 2 types of embedding in our models with the dimension as 100.

1. Word2Vector Embedding:

Word2Vec models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space.

2. GloVe (Global Vectors) Embedding:

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

Step 1: Load the embedding weights

```
EMBEDDING_FILE_200 = './glove.6B.200d.txt'
```

```
def define_embedding(EMBEDDING_FILE, size):
  embeddings = {}
  for o in open(EMBEDDING_FILE):
      word = o.split(" ")[0]
      # print(word)
      embd = o.split(" ")[1:]
      embd = np.asarray(embd, dtype='float32')
      # print(embd)
      embeddings[word] = embd

  # create a weight matrix for words in training docs
  embedding_matrix = np.zeros((vocab_size, size))

  for word, i in tokenizer.word_index.items():
    embedding_vector = embeddings.get(word)
    if embedding_vector is not None:
      embedding_matrix[i] = embedding_vector
    return embedding_matrix
```

```
embedding_matrix = define_embedding(EMBEDDING_FILE_200, 200)
```

Step 2: Build the Model

```
embedding_size = 200
lstm_out = 196

lstm_model_1 = Sequential()
lstm_model_1.add(Embedding(vocab_size, output_dim=embedding_size, weights=[embedding_matrix], input_length=200, trainable=True))
lstm_model_1.add(SpatialDropout1D(0.4))
lstm_model_1.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
lstm_model_1.add(Dense(12,activation='softmax'))
lstm_model_1.compile(loss = 'categorical_crossentropy', optimizer=keras.optimizers.Adam(lr = 0.01),metrics = ['accuracy'])
print(lstm_model_1.summary())
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 200, 200)          2032400
_________________________________________________________________
spatial_dropout1d_2 (Spatial (None, 200, 200)          0
_________________________________________________________________
lstm_1 (LSTM)                (None, 196)               311248
_________________________________________________________________
dense_2 (Dense)              (None, 12)                2364
=================================================================
Total params: 2,346,012
Trainable params: 2,346,012
Non-trainable params: 0
_________________________________________________________________
None
```

Step 3: Train and Evaluate the Model

```
# fit the model
lstm_model_1.fit(X_train, y_train, validation_data = (X_val,y_val), epochs=10, batch_size=32)
# evaluate the model
lstm_loss_1, lstm_accuracy_1 = lstm_model_1.evaluate(X_test, y_test, verbose=0)
print('Accuracy: %f' % (lstm_accuracy_1*100))
print('loss: %f' % (lstm_loss_1*100))
```

```
Epoch 1/10
WARNING:tensorflow:Model was constructed with shape (None, 200) for input Tensor("embedding_53_input:0", shape=(None, 200), dtype=floa
WARNING:tensorflow:Model was constructed with shape (None, 200) for input Tensor("embedding_53_input:0", shape=(None, 200), dtype=floa
113/113 [==============================] - ETA: 0s - loss: 1.4743 - accuracy: 0.5465WARNING:tensorflow:Model was constructed with shap
113/113 [==============================] - 17s 148ms/step - loss: 1.4743 - accuracy: 0.5465 - val_loss: 1.2739 - val_accuracy: 0.6225
Epoch 2/10
113/113 [==============================] - 16s 145ms/step - loss: 0.9766 - accuracy: 0.7001 - val_loss: 1.2155 - val_accuracy: 0.6303
Epoch 3/10
113/113 [==============================] - 16s 146ms/step - loss: 0.5979 - accuracy: 0.8091 - val_loss: 1.2787 - val_accuracy: 0.6374
Epoch 4/10
113/113 [==============================] - 16s 146ms/step - loss: 0.3760 - accuracy: 0.8829 - val_loss: 1.4046 - val_accuracy: 0.6400
Epoch 5/10
113/113 [==============================] - 16s 145ms/step - loss: 0.2656 - accuracy: 0.9119 - val_loss: 1.5892 - val_accuracy: 0.6212
Epoch 6/10
113/113 [==============================] - 16s 146ms/step - loss: 0.2040 - accuracy: 0.9362 - val_loss: 1.6757 - val_accuracy: 0.6361
Epoch 7/10
113/113 [==============================] - 16s 146ms/step - loss: 0.1813 - accuracy: 0.9415 - val_loss: 1.7817 - val_accuracy: 0.6452
Epoch 8/10
113/113 [==============================] - 17s 146ms/step - loss: 0.1457 - accuracy: 0.9515 - val_loss: 1.9239 - val_accuracy: 0.6413
Epoch 9/10
113/113 [==============================] - 17s 147ms/step - loss: 0.1210 - accuracy: 0.9643 - val_loss: 2.0365 - val_accuracy: 0.6452
Epoch 10/10
113/113 [==============================] - 16s 146ms/step - loss: 0.1109 - accuracy: 0.9624 - val_loss: 2.0508 - val_accuracy: 0.6420
Accuracy: 61.695904
loss: 220.526338
```

Observation:

- The test accuracy has increased by just 1% which is 61.69% than the simple LSTM model and loss is still as high as 220%
- Here also the model has over-fitted as we can see training data accuracy has consistently increased up to 96.24% but validation accuracy is stuck to 64.20%
- Regularization needs to be added to the dense layers to overcome over-fitting

Model 3: LSTM Model by excluding ‘GRP-74’ which consisted of incidents from various group

Step 1: Remove the records where target data was ‘GRP_74’

```
df_filtered = df[df['Assignment group - After'] != 'GRP_74']
```

```
df_filtered.shape
```

```
(5160, 3)
```

```
df_filtered['Assignment group - After'].value_counts()
```

```
GRP_0     3199
GRP_8      313
GRP_24     277
GRP_12     243
GRP_2      238
GRP_19     212
GRP_3      198
GRP_13     141
GRP_14     116
GRP_25     116
GRP_33     107
Name: Assignment group - After, dtype: int64
```

Step 2: Build the Model

```
lstm_model_4 = Sequential()
lstm_model_4.add(Embedding(vocab_size, output_dim=200, input_length=25, trainable=True))
lstm_model_4.add(Dropout(0.5))
lstm_model_4.add(LSTM(128, recurrent_dropout = 0.5 , dropout = 0.5))
lstm_model_4.add(Dense(64, activity_regularizer=l1(0.01)))
lstm_model_4.add(LeakyReLU(alpha=0.3))
lstm_model_4.add(Dropout(0.5))
lstm_model_4.add(Dense(11, activity_regularizer=l1(0.01),activation='softmax'))
lstm_model_4.compile(loss = 'categorical_crossentropy', optimizer=adam_optimizer,metrics = ['accuracy'])
print(lstm_model_4.summary())
```

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_13 (Embedding)     (None, 25, 200)           2032400
_________________________________________________________________
dropout_8 (Dropout)          (None, 25, 200)           0
_________________________________________________________________
lstm_10 (LSTM)               (None, 128)               168448
_________________________________________________________________
dense_15 (Dense)             (None, 64)                8256
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 64)                0
_________________________________________________________________
dropout_9 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_16 (Dense)             (None, 11)                715
=================================================================
Total params: 2,209,819
Trainable params: 2,209,819
Non-trainable params: 0
```

Step 3: Train and Fit the Model

```
BATCH_SIZE = 32
EPOCHS = 20
history = lstm_model_4.fit(X_attd_train_val, y_attd_train_val, validation_split=0.2, epochs=EPOCHS, batch_size=BATCH_SIZE, verbose=1, shuffle=True, callbacks=[stop, reduce_lr])
```

```
Epoch 1/20
91/91 [==============================] - 8s 91ms/step - loss: 0.9439 - accuracy: 0.8418 - val_loss: 1.3182 - val_accuracy: 0.7635
Epoch 2/20
91/91 [==============================] - 8s 91ms/step - loss: 0.8061 - accuracy: 0.8581 - val_loss: 1.2515 - val_accuracy: 0.7649
Epoch 3/20
91/91 [==============================] - 8s 91ms/step - loss: 0.7365 - accuracy: 0.8667 - val_loss: 1.2006 - val_accuracy: 0.7663
Epoch 4/20
91/91 [==============================] - 8s 91ms/step - loss: 0.6801 - accuracy: 0.8726 - val_loss: 1.2877 - val_accuracy: 0.7663
Epoch 5/20
91/91 [==============================] - 8s 91ms/step - loss: 0.6443 - accuracy: 0.8837 - val_loss: 1.1598 - val_accuracy: 0.7676
Epoch 6/20
91/91 [==============================] - 8s 92ms/step - loss: 0.6192 - accuracy: 0.8889 - val_loss: 1.1425 - val_accuracy: 0.7732
Epoch 7/20
91/91 [==============================] - 8s 91ms/step - loss: 0.6067 - accuracy: 0.8896 - val_loss: 1.2295 - val_accuracy: 0.7718
Epoch 8/20
91/91 [==============================] - 8s 91ms/step - loss: 0.5702 - accuracy: 0.8979 - val_loss: 1.2027 - val_accuracy: 0.7801
Epoch 9/20
91/91 [==============================] - 8s 91ms/step - loss: 0.5645 - accuracy: 0.9034 - val_loss: 1.1859 - val_accuracy: 0.7773
Epoch 10/20
91/91 [==============================] - 8s 92ms/step - loss: 0.5222 - accuracy: 0.9072 - val_loss: 1.1434 - val_accuracy: 0.7732
Epoch 11/20
91/91 [==============================] - 8s 92ms/step - loss: 0.5382 - accuracy: 0.9052 - val_loss: 1.1618 - val_accuracy: 0.7746
Epoch 12/20
91/91 [==============================] - 8s 92ms/step - loss: 0.5164 - accuracy: 0.9128 - val_loss: 1.2002 - val_accuracy: 0.7676
Epoch 13/20
91/91 [==============================] - 8s 91ms/step - loss: 0.4756 - accuracy: 0.9173 - val_loss: 1.2175 - val_accuracy: 0.7704
Epoch 14/20
91/91 [==============================] - 8s 92ms/step - loss: 0.4547 - accuracy: 0.9259 - val_loss: 1.2243 - val_accuracy: 0.7787
Epoch 15/20
91/91 [==============================] - 8s 92ms/step - loss: 0.4413 - accuracy: 0.9294 - val_loss: 1.2778 - val_accuracy: 0.7787
```

Step 4: Evaluate the Model on Test data

```
loss, accuracy = lstm_model_4.evaluate(X_attd_test, y_attd_test)
print('Accuracy: %f' % (accuracy*100))
print('loss: %f' % (loss))
```

```
49/49 [==============================] - 1s 11ms/step - loss: 1.4020 - accuracy: 0.7578
Accuracy: 75.775194
loss: 1.402029
```

Observation:

- Although the training data has overfit, but still the validation and test accuracy has increased considerably
- The validation accuracy has gone down to 78.01% and loss has increased to 124%
- The test accuracy has increased to 75.78% as compared to attention layer model
- On the other hand, the loss has reduced to 140.20%, which is high as compared to attention models

Model 4: Attention Model by excluding 'GRP-74' which consisted of incidents from various group

Attention Model

The attention mechanism [1] was developed in the context of encoder-decoder architectures for Neural Machine Translation (NMT) [2, 24], and rapidly applied to naturally related tasks such as image captioning (translating an image to a sentence) [25], and summarization (translating to a more compact language) [21]. From a high-level, by allowing the decoder to shop for what it needs over multiple vectors, attention relieves the encoder from the burden of having to embed the input into a single fixed-length vector, and thus allows to keep much more information [1].

Today, attention is ubiquitous in deep learning models, and is not used only in encoder decoder contexts. Notably, attention devices have been proposed for encoders only, to solve tasks such as document classification [27] or representation learning [5]. Such mechanisms are qualified as self or inner attention.

In what follows, we will start by presenting attention in the original context of encoder decoder for NMT, using the general framework introduced by [20], and then introduce self attention.

Step 1: Build the Attention layer

```
class Attention(tf.keras.Model):
    def __init__(self, units):
        super(Attention, self).__init__()
        self.W1 = tf.keras.layers.Dense(units)
        self.W2 = tf.keras.layers.Dense(units)
        self.V = tf.keras.layers.Dense(1)

    def call(self, features, hidden):
        # hidden shape == (batch_size, hidden size)
        # hidden_with_time_axis shape == (batch_size, 1, hidden size)
        # we are doing this to perform addition to calculate the score
        hidden_with_time_axis = tf.expand_dims(hidden, 1)

        # score shape == (batch_size, max_length, 1)
        # we get 1 at the last axis because we are applying score to self.V
        # the shape of the tensor before applying self.V is (batch_size, max_length, units)
        score = tf.nn.tanh(
            self.W1(features) + self.W2(hidden_with_time_axis))
        # attention_weights shape == (batch_size, max_length, 1)
        attention_weights = tf.nn.softmax(self.V(score), axis=1)

        # context_vector shape after sum == (batch_size, hidden_size)
        context_vector = attention_weights * features
        context_vector = tf.reduce_sum(context_vector, axis=1)
        return context_vector, attention_weights
```

Step 2: Build the Model

```
sequence_input = Input(shape=(MAX_LEN,), dtype="int32")
embedded_sequences = Embedding(MAX_FEATURES, EMBED_SIZE)(sequence_input)

lstm = LSTM(RNN_CELL_SIZE, return_sequences = True)(embedded_sequences)

# Getting our LSTM outputs
(lstm, state_h, state_c) = LSTM(RNN_CELL_SIZE, return_sequences=True, return_state=True)(lstm)

context_vector, attention_weights = Attention(10)(lstm, state_h)
dense1 = Dense(1024, activity_regularizer=l1(0.01))(context_vector)
activation1 = LeakyReLU(alpha=0.2)(dense1)
dropout1 = Dropout(0.5)(activation1)
dense2 = Dense(512, activity_regularizer=l1(0.1))(dropout1)
activation2 = LeakyReLU(alpha=0.2)(dense2)
dropout2 = Dropout(0.5)(activation2)
output = Dense(11, activation="softmax")(dropout2)

model_label = keras.Model(inputs=sequence_input, outputs=output)
```

```
adam_optimizer = Adam(lr=1e-3, beta_1=0.009, beta_2=0.999, amsgrad=True)

model_label.compile(loss='categorical_crossentropy',
              optimizer=adam_optimizer,
              metrics='accuracy')
```

Step 3: Train and Fit the Model

```
BATCH_SIZE = 32
EPOCHS = 30
for i in range(EPOCHS):
  model_label.fit(X_attd_train_val, y_attd_train_val, validation_split=0.2, epochs=1, batch_size=BATCH_SIZE, verbose=1, shuffle=True, callbacks=[stop, reduce_lr])
  model_label.reset_states()
```

```
91/91 [==============================] - 3s 35ms/step - loss: 1.0313 - accuracy: 0.7913 - val_loss: 1.0595 - val_accuracy: 0.7925
91/91 [==============================] - 3s 35ms/step - loss: 0.8703 - accuracy: 0.8231 - val_loss: 0.9105 - val_accuracy: 0.8008
91/91 [==============================] - 3s 37ms/step - loss: 0.7950 - accuracy: 0.8432 - val_loss: 0.8976 - val_accuracy: 0.8285
91/91 [==============================] - 3s 37ms/step - loss: 0.7382 - accuracy: 0.8588 - val_loss: 0.9151 - val_accuracy: 0.8188
91/91 [==============================] - 3s 35ms/step - loss: 0.7107 - accuracy: 0.8823 - val_loss: 0.9217 - val_accuracy: 0.8368
91/91 [==============================] - 3s 35ms/step - loss: 0.6845 - accuracy: 0.9052 - val_loss: 0.8723 - val_accuracy: 0.8520
91/91 [==============================] - 3s 35ms/step - loss: 0.6462 - accuracy: 0.9218 - val_loss: 0.8924 - val_accuracy: 0.8382
91/91 [==============================] - 3s 35ms/step - loss: 0.6240 - accuracy: 0.9259 - val_loss: 0.8842 - val_accuracy: 0.8437
91/91 [==============================] - 3s 35ms/step - loss: 0.6009 - accuracy: 0.9308 - val_loss: 0.8476 - val_accuracy: 0.8479
91/91 [==============================] - 3s 35ms/step - loss: 0.5758 - accuracy: 0.9367 - val_loss: 0.9423 - val_accuracy: 0.8437
91/91 [==============================] - 3s 35ms/step - loss: 0.5537 - accuracy: 0.9436 - val_loss: 0.9581 - val_accuracy: 0.8562
91/91 [==============================] - 3s 35ms/step - loss: 0.5403 - accuracy: 0.9640 - val_loss: 0.8804 - val_accuracy: 0.8617
91/91 [==============================] - 3s 35ms/step - loss: 0.5293 - accuracy: 0.9699 - val_loss: 0.8534 - val_accuracy: 0.8617
91/91 [==============================] - 3s 35ms/step - loss: 0.5160 - accuracy: 0.9768 - val_loss: 0.8337 - val_accuracy: 0.8617
91/91 [==============================] - 3s 35ms/step - loss: 0.4839 - accuracy: 0.9841 - val_loss: 1.6602 - val_accuracy: 0.7026
91/91 [==============================] - 3s 35ms/step - loss: 0.4978 - accuracy: 0.9817 - val_loss: 0.9272 - val_accuracy: 0.8492
91/91 [==============================] - 3s 35ms/step - loss: 0.4569 - accuracy: 0.9900 - val_loss: 0.8662 - val_accuracy: 0.8520
91/91 [==============================] - 3s 35ms/step - loss: 0.4345 - accuracy: 0.9931 - val_loss: 0.8344 - val_accuracy: 0.8562
91/91 [==============================] - 3s 35ms/step - loss: 0.4234 - accuracy: 0.9917 - val_loss: 0.9151 - val_accuracy: 0.8382
```

Step 4: Evaluate the Model on Test data

```
loss, accuracy = model_label.evaluate(X_attd_test, y_attd_test)
print('Accuracy: %f' % (accuracy*100))
print('loss: %f' % (loss))
```

```
49/49 [==============================] - 0s 8ms/step - loss: 1.2642 - accuracy: 0.7487
Accuracy: 74.870801
loss: 1.264174
```

Observation:

- Using the attention layer in the network and removing the sparse data from th e dataset has kicked the accuracy to a large extent
- The training data has over-
  fit to 99.27% but for the first time the validation accuracy has gone upto 84.92 %
- The test accuracy has also increased to 74.87%
- The validation and test loss are the minimum as compared to all the model, 8 2.28% and 1.26% respectively

Prediction:

```
incident = ['job printer printer issue']
#['login issue verify user detail employee']
#vectorizing the tweet by the pre-fitted tokenizer instance
inc = tokenizer.texts_to_sequences(incident)
#padding the tweet to have exactly the same shape as `embedding_2` input
#twt = pad_sequences(inc, maxlen=28, dtype='int32', value=0)
inc = sequence.pad_sequences(inc, maxlen=50)
print(inc)
sentiment = lstm_model_4.predict(inc,batch_size=1,verbose = 2)
print(sentiment)
```

```
[[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 56 38
  38 11]]
1/1 - 0s
[[0.97194606 0.00216034 0.00175424 0.00179068 0.00553611 0.00377704
  0.0024497  0.00251492 0.00322086 0.00269284 0.00215725]]
```

- The above gives the probability percentage of the data that can be present for each group.

6. Visualization

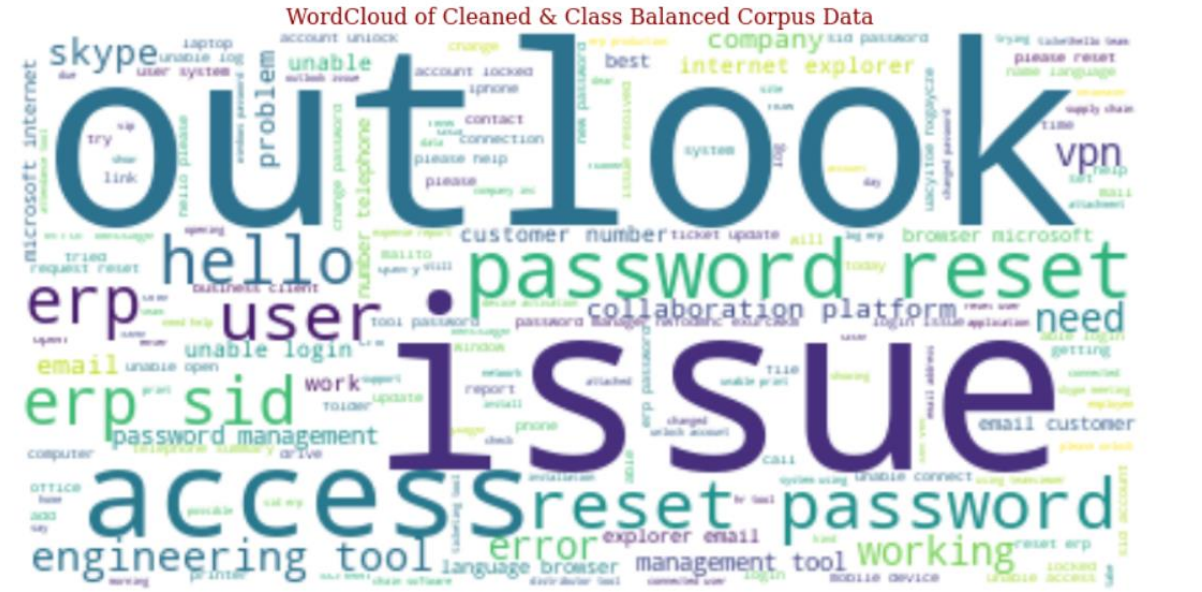
❖ Word Cloud for Top 5 groups assigned:

```
font = {'family': 'serif',
        'color':  'darkred',
        'weight': 'normal',
        'size': 16,
        }
```

```
def visualize_wordCloud(data) :
    All_words = ""
    All_words += " ".join(data)
    wordcloud = WordCloud(background_color='white').generate(All_words) # width and hei
ork
    plt.figure(figsize=(15,15))
    plt.title("WordCloud of Cleaned & Class Balanced Corpus Data", fontdict=font)
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis("off")
    plt.show()
```

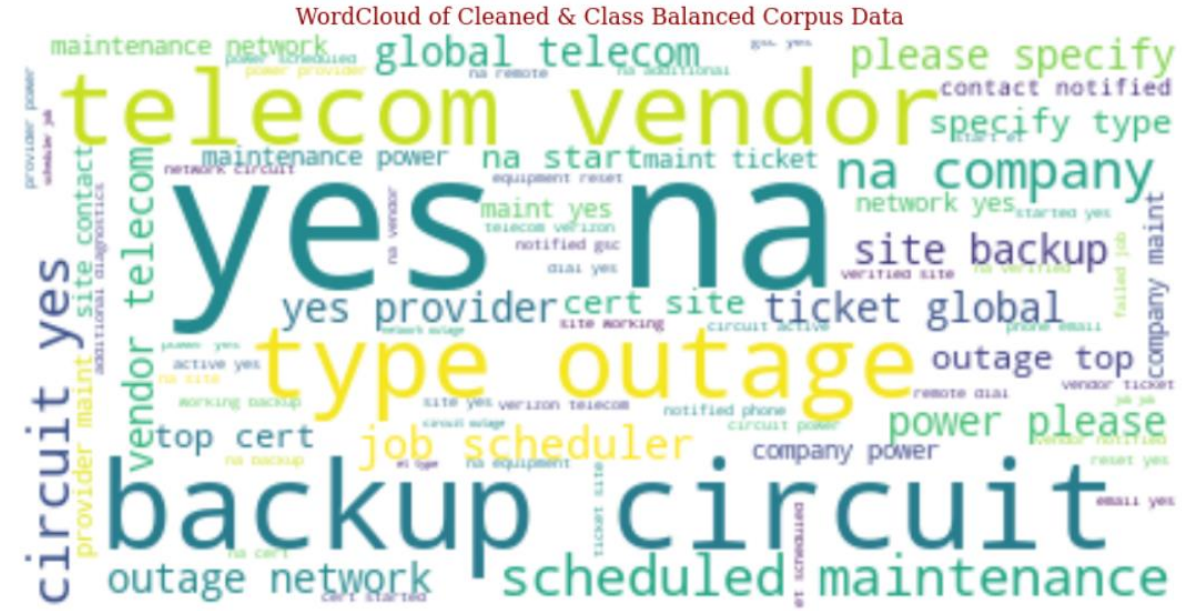
1) Group 0

```
visualize_wordCloud(cleaned_tickets[cleaned_tickets["Assignment group"].isin(['GRP_0'])].CleanDescription)
```



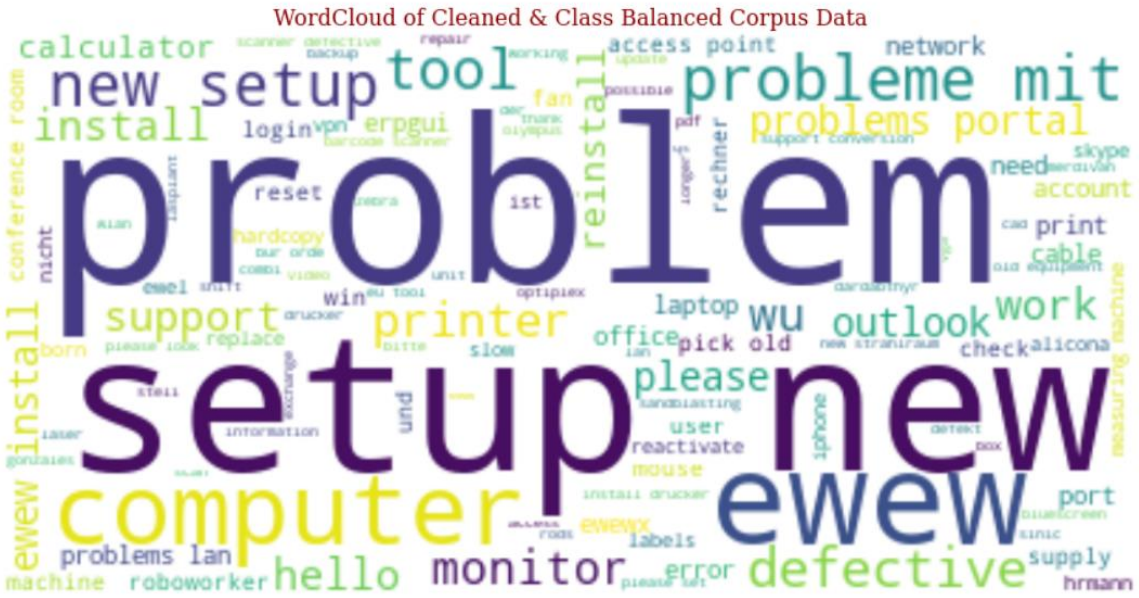
2) Group 8

```
visualize_wordCloud(cleaned_tickets[cleaned_tickets["Assignment group"].isin(['GRP_8'])].CleanDescription)
```



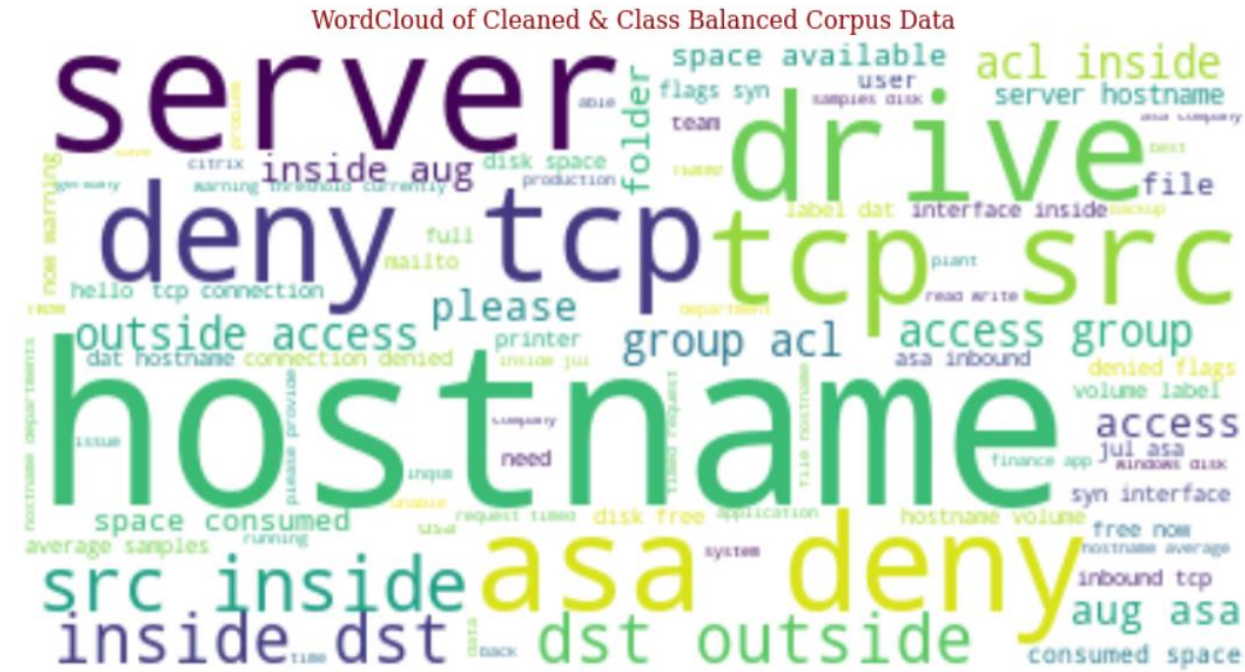
3) Group 24

```
visualize_wordCloud(cleaned_tickets[cleaned_tickets["Assignment group"].isin(['GRP_24'])].CleanDescription)
```



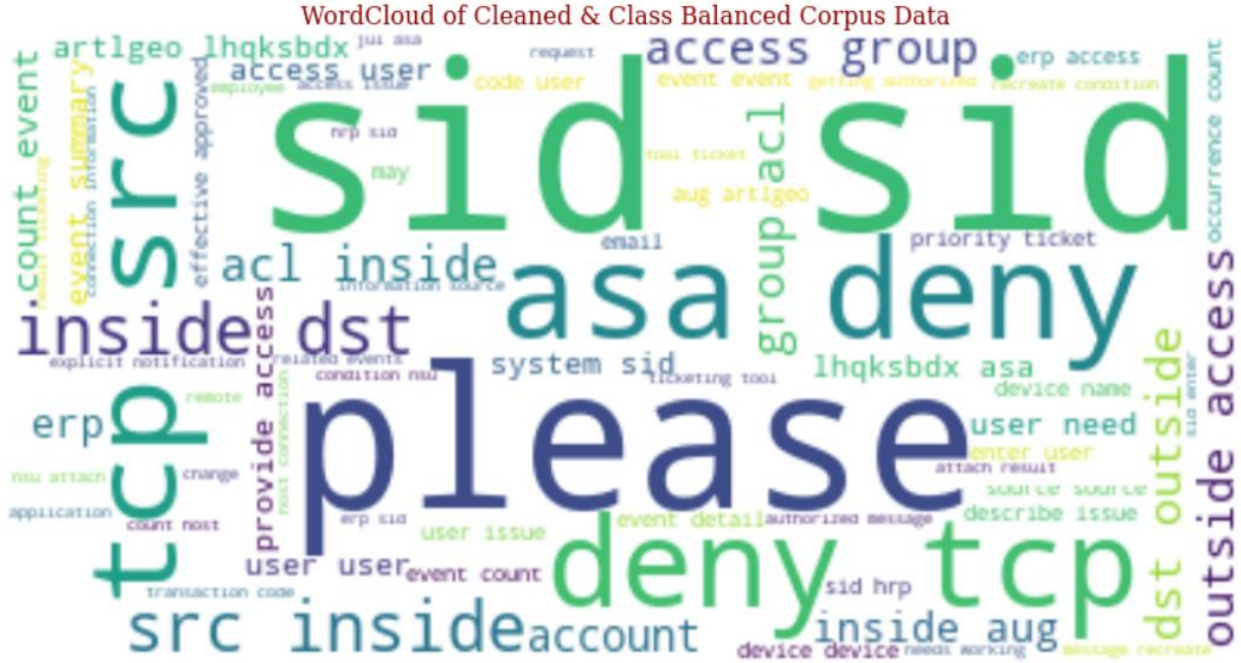
4) Group 12

```
visualize_wordCloud(cleaned_tickets[cleaned_tickets["Assignment group"].isin(['GRP_12'])].CleanDescription)
```



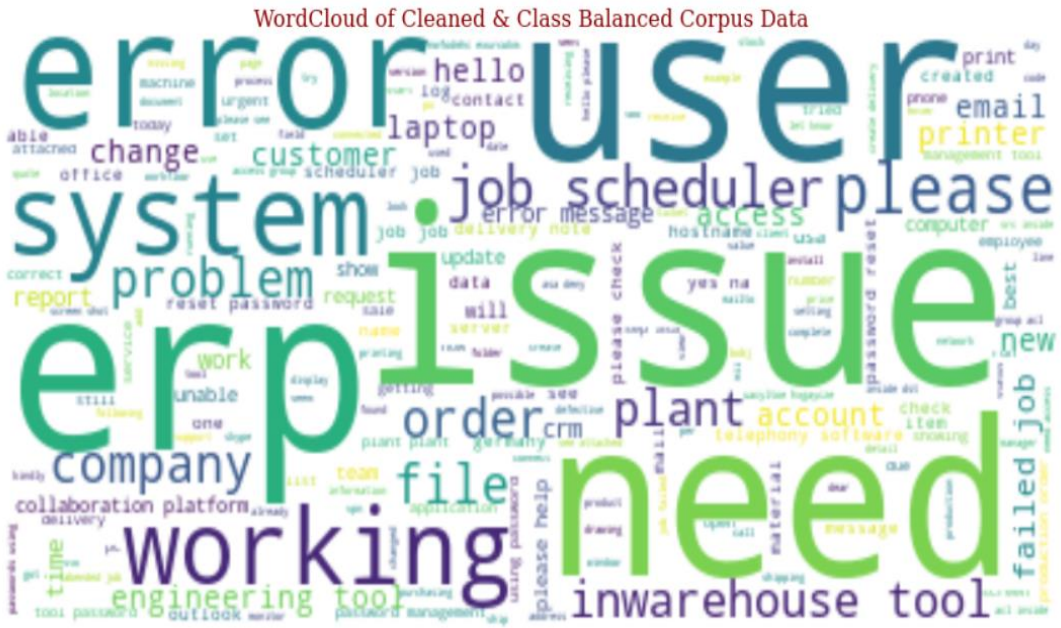
5) Group 2

```
visualize_wordCloud(cleaned_tickets[cleaned_tickets["Assignment group"].isin(['GRP_2'])].CleanDescription)
```



6) Other Groups other than the top 5

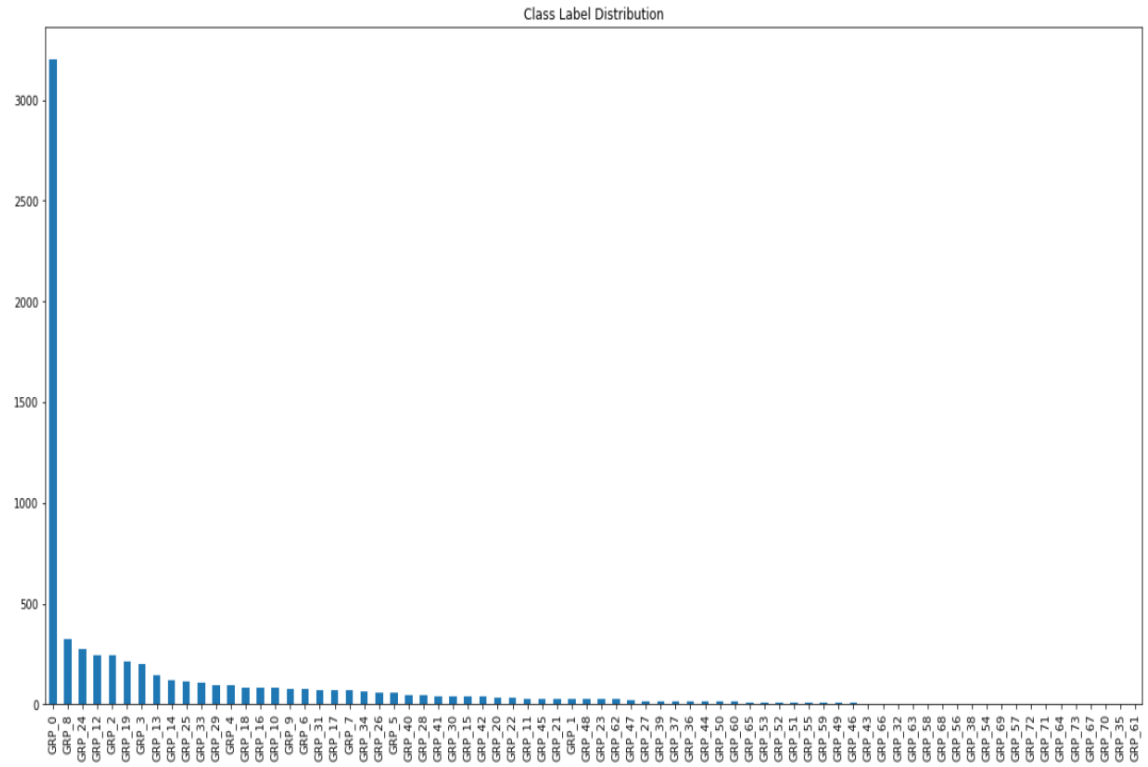
```
visualize_wordCloud(cleaned_tickets[~cleaned_tickets["Assignment group"].isin(['GRP_0','GRP_8','GRP_24','GRP_12','GRP_2'])].CleanDescription)
```



7) Plotting the 'Assignment group' data in bar chart using matplotlib library:

```
cleaned_tickets['Assignment group'].value_counts().sort_values(ascending=False).plot(kind='bar', figsize=(20,10), title='Class Label Distribution')
```

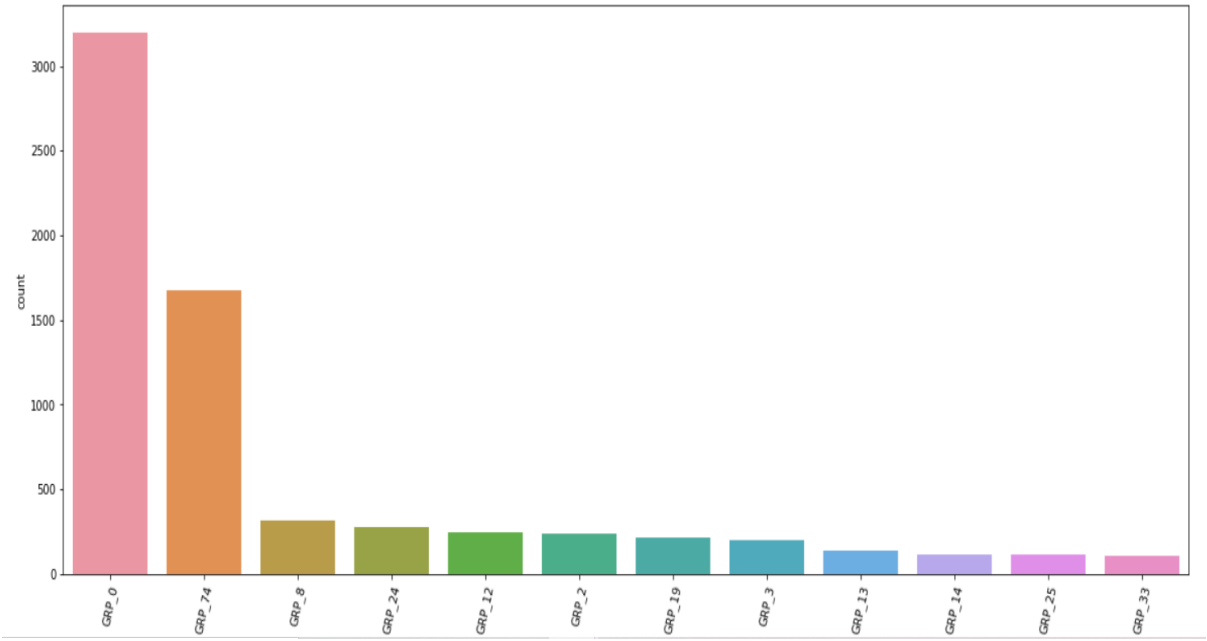
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3947e7e1d0>
```



8) After handling data imbalance:

```
df_desc = df['Assignment group - After'].value_counts().sort_values(ascending=False).index
plt.figure(figsize= (20,8))
plt.xticks(rotation=75)
sns.countplot(x='Assignment group - After', data=df, order=df_desc)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ebdd8c1d0>
```



7. Implications

Although this model can classify the IT tickets with 91.24% accuracy, to achieve better accuracy in the real world it would be good if the business can collect additional data around 300 records for each group.

8. Limitations

As part of Data pre-processing, we had grouped all assignment groups with less than 10 entries as one group (GRP_74) which had reduced the Target class from 74 to 12 groups. While applying this model in the real world there could be additional intervention required to classify the tickets if it has been classified as GRP_74 by our model. Since the number of elements reported under GRP_74 are less, we expect this intervention to be done less often.

9. Closing Reflections

We found the data was present in multiple languages and in various formats such as emails, chat, etc bringing in a lot of variability in the data to be analysed. The Business can improve the process of raising tickets via a common unified IT Ticket Service Portal which reduces the above-mentioned variability. By doing this, the model can perform better which can help businesses to identify the problem area for relevant clusters of topics.