Project Description

One of the key activities of any IT function is to “Keep the lights on” to ensure there is no impact to the Business operations. IT leverages Incident Management process to achieve the above Objective. An incident is something that is unplanned interruption to an IT service or reduction in the quality of an IT service that affects the Users and the Business. The main goal of Incident Management process is to provide a quick fix / workarounds or solutions that resolves the interruption and restores the service to its full capacity to ensure no business impact.

In most of the organizations, incidents are created by various Business and IT Users, End Users/ Vendors if they have access to ticketing systems, and from the integrated monitoring systems and tools. Assigning the incidents to the appropriate person or unit in the support team has critical importance to provide improved user satisfaction while ensuring better allocation of support resources. The assignment of incidents to appropriate IT groups is still a manual process in many of the IT organizations. Manual assignment of incidents is time consuming and requires human efforts. There may be mistakes due to human errors and resource consumption is carried out ineffectively because of the misaddressing. On the other hand, manual assignment increases the response and resolution times which result in user satisfaction deterioration / poor customer service.

Business Domain Value

In the support process, incoming incidents are analysed and assessed by organization’s support teams to fulfil 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. 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. 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.

Tools and technologies used:

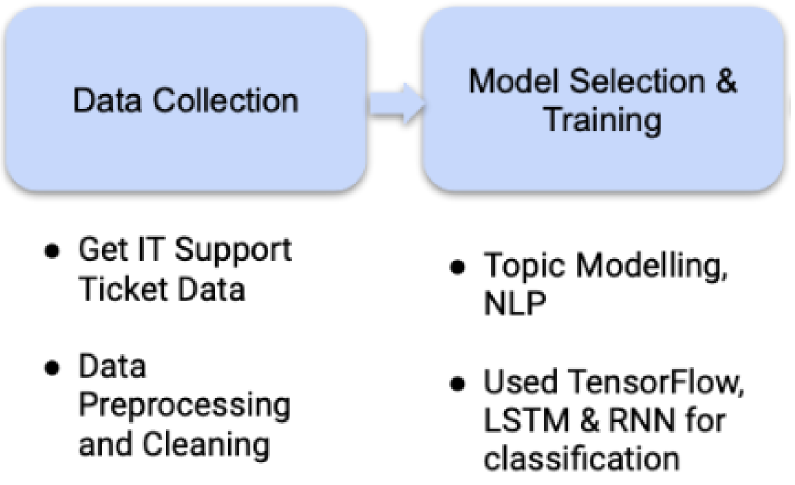
* Python

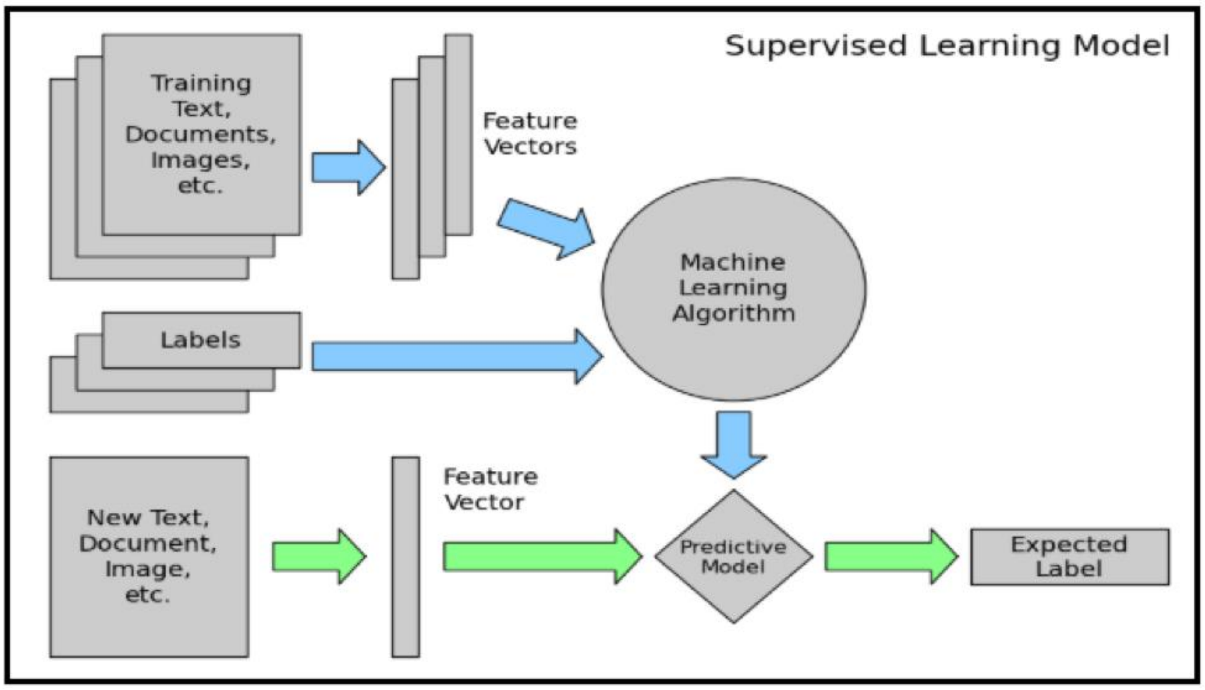
* Neural Network models

* 50000+ sample tickets from open sources

Process Overview:

The overall workflow of the process can be divided into various sub parts. A very high-level overview is shown in the figure below.


Data Collection
Model Selection & Training
Get IT Support Ticket Data
Data Preprocessing and Cleaning
Topic Modelling, NLP
Used TensorFlow, LSTM & RNN for classification


Supervised Learning Model
Training Text, Documents, Images, etc.
Feature Vectors
Labels
Machine Learning Algorithm
New Text, Document, Image, etc.
Feature Vector
Predictive Model
Expected Label

An end-to-end text classification pipeline is composed of following components:

1. Training text: It is the input text through which our supervised learning model can learn and predict the required class.

2. Feature Vector: A feature vector is a vector that contains information describing the characteristics of the input data.

3. Labels: These are the predefined categories/classes that our model will predict

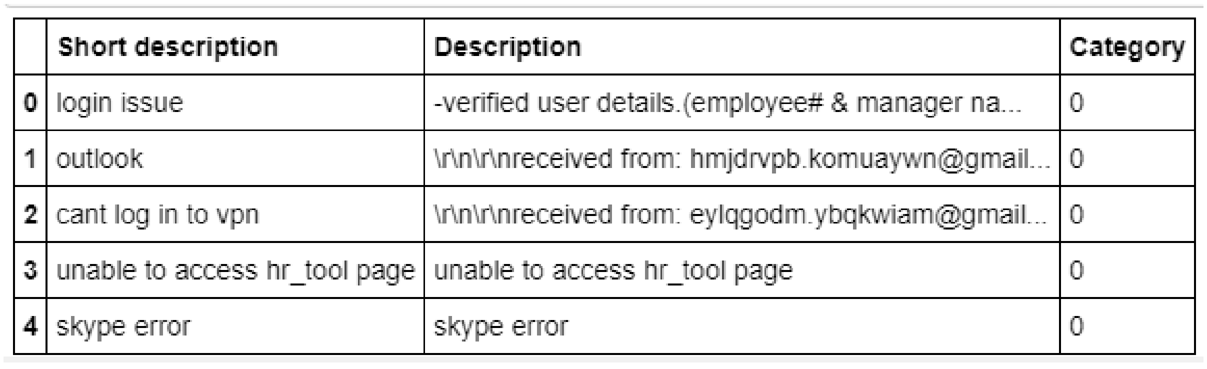
4. ML Algo: It is the algorithm through which our model can deal with text classification (In our case: CNN, RNN, HAN)

5. Predictive Model: A model which is trained on the historical dataset which can perform label predictions.

Dataset:

We pulled the dataset directly from open source and tried to classify them using ML techniques.

Each column (or attributes) contains information of incident records. A sample of incident records is shown in below.



	Short description	Description	Category
0	login issue	-verified user details.(employee# & manager na...	0
1	outlook	\r\n\r\nreceived from: hmjdrvpb.komuaywn@gmail...	0
2	cant log in to vpn	\r\n\r\nreceived from: eylqgodm.ybqkwiam@gmail...	0
3	unable to access hr_tool page	unable to access hr_tool page	0
4	skype error	skype error	0

- Short description – These descriptions gives an overview of the issue raised by user

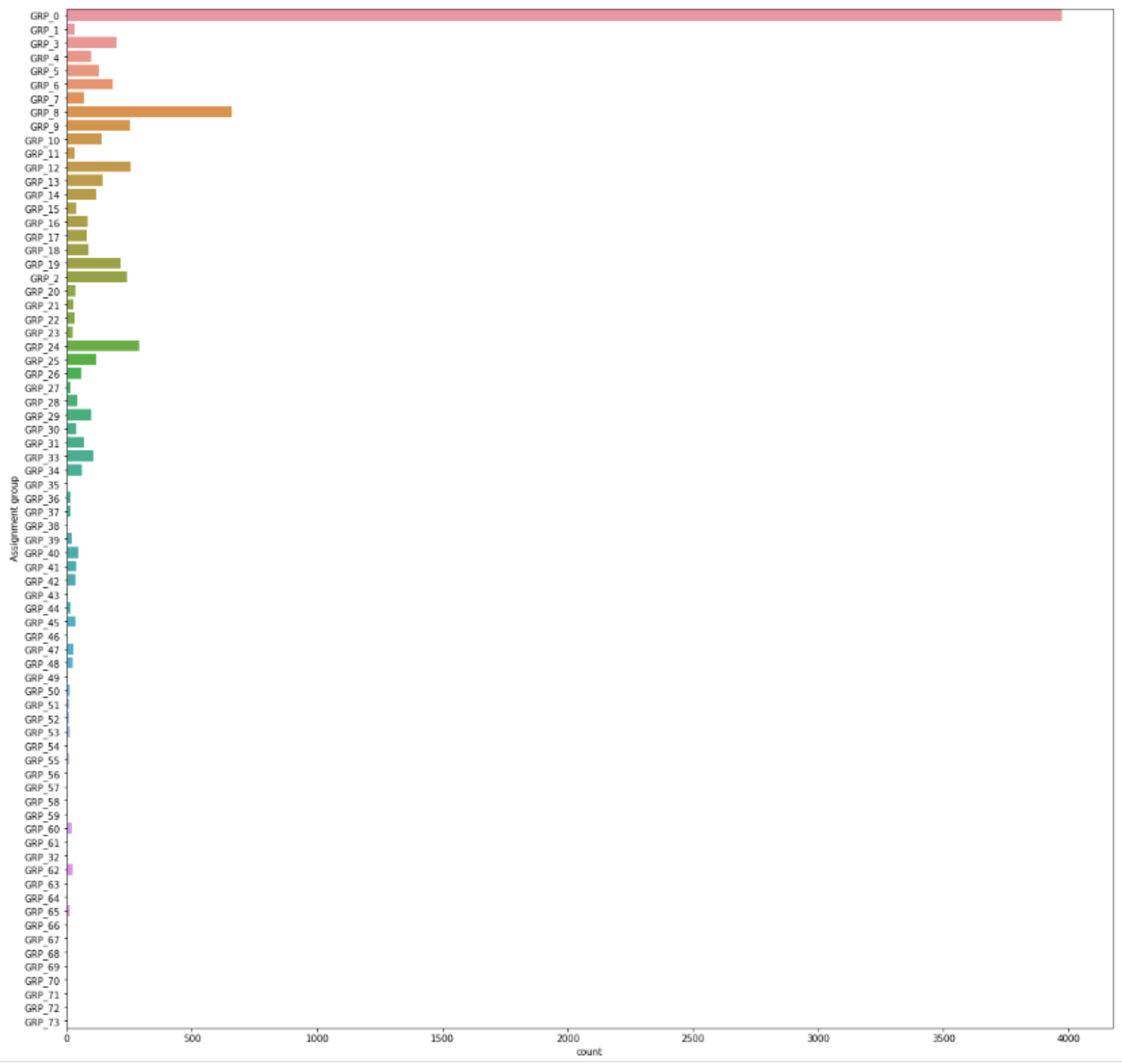
- Description – Explanation of the issue faced by user

- Caller – User details

- Assignment group – Group assigned to different category of issues raised by user.

Findings:

- The dataset provided is highly imbalanced.
- There are Null values present in the dataset.
- There are 74 unique Assignment groups in the dataset.



Python Notebook for the Capstone Project

We went with below approaches: -

a) **With Sampling -** **AIML Capstone_upsampled.ipynb**
b) **Without Sampling -** **AIML Capstone_v2.ipynb**

EDA & PRE-PROCESSING

Data Preparation and Method Discovery

Data preparation processes include data recognition, parsing, filtering, data cleansing, and transformation. In this case, we add data grouping by keywords. Thus, the data preparation processes are as follows:

(a) Data Recognition: identify the incident records collected as the sample of raw structured data in spreadsheet format.

(b) Data parsing: the purpose of data parsing is to resolve a sentence into its component parts of speech. We created another keyword dictionary and modified the program to execute both of dictionaries.

(c) Data filtering: involves selecting rows and columns of data for further document collection or text corpus. Thus, the text corpus includes several columns, including system failures, component failures, incident descriptions, and assigned resolver group.

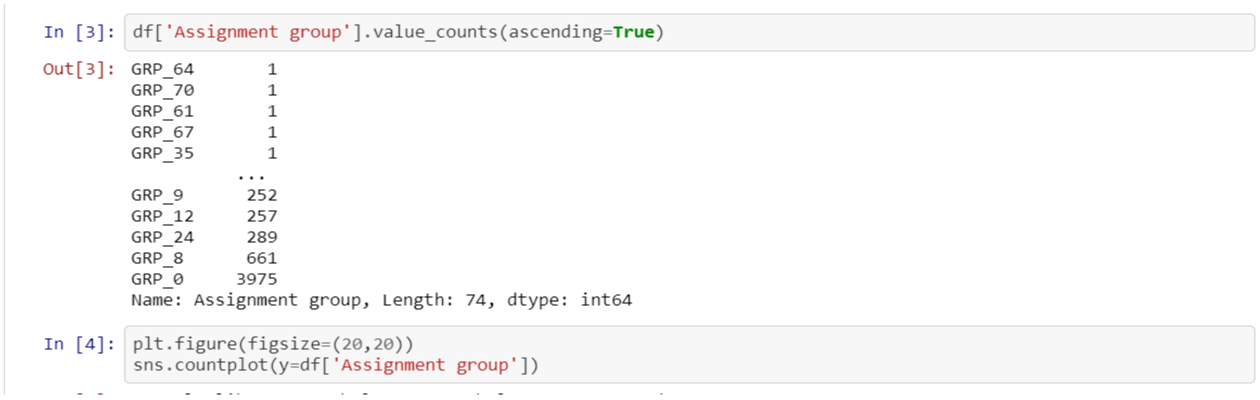
(d) Data cleaning: correct inconsistent data, checking to see the data conform across its columns and filling in missing values in particular for the component failures and assigned resolver groups.

(e) Data grouping: from the word extraction given a lot of words and then grouped them into the words of component and system-type failures.

(f) Data transformation: we transformed data prior to data analysis. Several steps need data transformation such as word extraction, text measurement, text mining discovery methods, comparing several decision trees to find out the most suitable method.

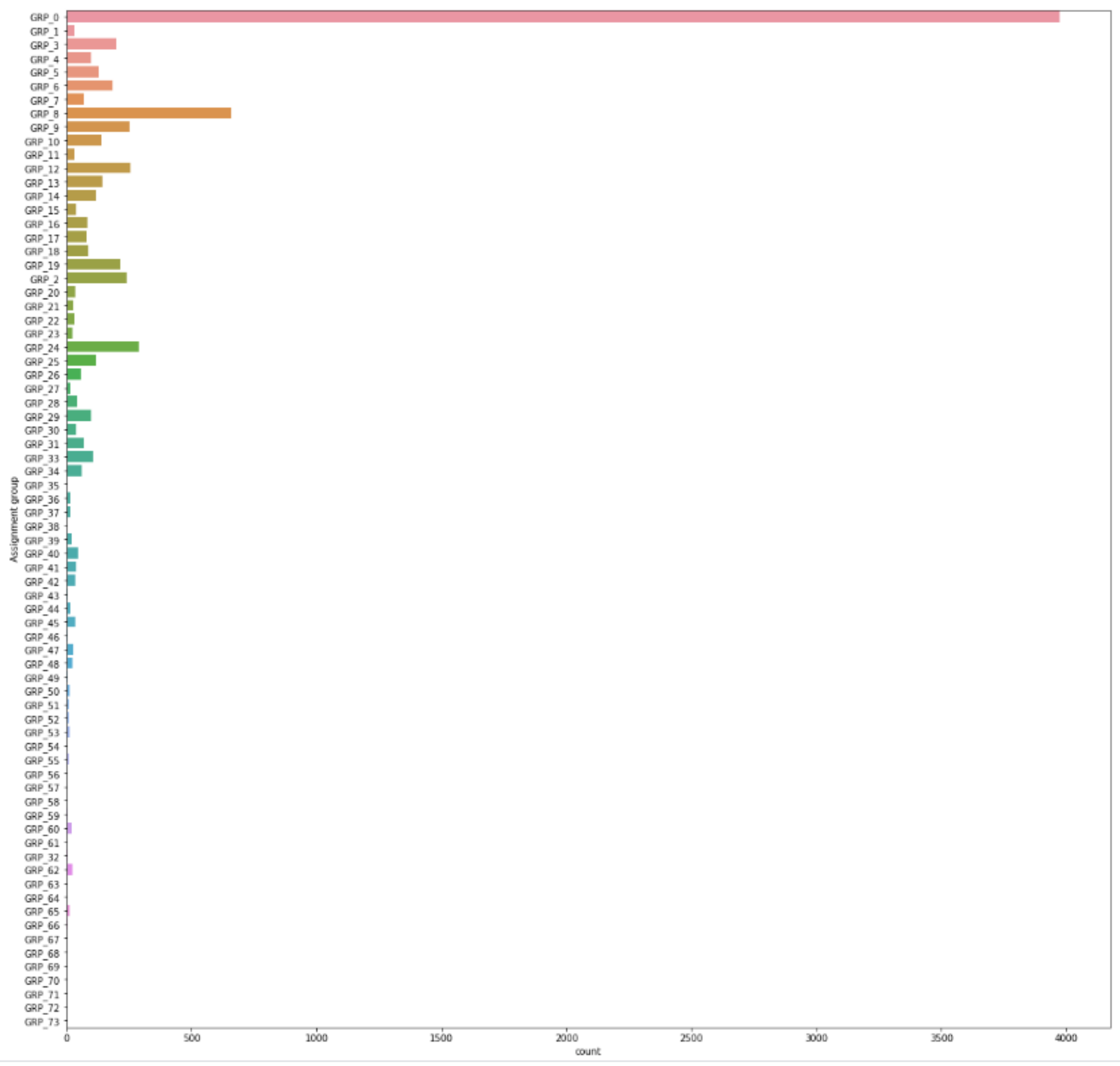
Before proceeding in building the model, we will perform Exploratory Data Analysis (EDA) and Pre-processing on our dataset to cleanse it and also make some observations of the data that will help us to build a better model.

1. In the dataset, we will see how many categories of tickets we have and plot them count wise. We also observe that there is a huge imbalance in the dataset with GROUP_0 having the most records and many other categories having just 1 record.



```
In [3]: df['Assignment group'].value_counts(ascending=True)

Out[3]: GRP_64       1
        GRP_70       1
        GRP_61       1
        GRP_67       1
        GRP_35       1
                  ...
        GRP_9      252
        GRP_12     257
        GRP_24     289
        GRP_8      661
        GRP_0     3975
        Name: Assignment group, Length: 74, dtype: int64

In [4]: plt.figure(figsize=(20,20))
        sns.countplot(y=df['Assignment group'])
```

Here is the countplot showing the class imbalance in the dataset and the number of records we have for each category.



2. On looking at the dataset , we also observe that the column ‘caller’ is irrelevant and can be dropped. Also we will check the other columns for empty values and replace them with some word, in our case , we have replaced ‘Short description’ with the word ‘the’.

3. The column ‘Assignment group’ is also categorical and we have converted it into a corresponding integer. We have also assigned this to a new column ‘category’ and dropped the previous column.

```
In [3]: # Drop irrelevant columns

        df = df.drop(['Caller'], axis=1)
```

```
In [4]: # Replace NAN with any stop word

        df['Short description'].fillna('the', inplace=True)
```

```
In [5]: # Convert categories into unique codes

        df['Category'] = df['Assignment group'].astype('category').cat.codes
```

```
In [6]: df = df.drop(['Assignment group'], axis=1)
```

```
In [7]: df.head()
```

Out[7]:

	Short description	Description	Category
0	login issue	-verified user details.(employee# & manager na...	0
1	outlook	\r\n\r\nreceived from: hmjdrvpb.komuaywn@gmail...	0
2	cant log in to vpn	\r\n\r\nreceived from: eylqgodm.ybqkwiam@gmail...	0
3	unable to access hr_tool page	unable to access hr_tool page	0
4	skype error	skype error	0

4. On looking at the ‘description’ column , we also see various emailids, numbers, and punctations and escape characters which we need to get rid of. We also remove some unnecessary words like hi, hello etc which are not meaningful.

```
In [8]: # Remove email id from description

        import re
        df['Description'] = df['Description'].apply((lambda x: re.sub('received from:.\S+@\S+','',x)))
```

```
In [9]: # Remove words that are not relevant

        df['Description'] = df['Description'].apply((lambda x: re.sub('hi|hello|team|thanks','',x)))
```

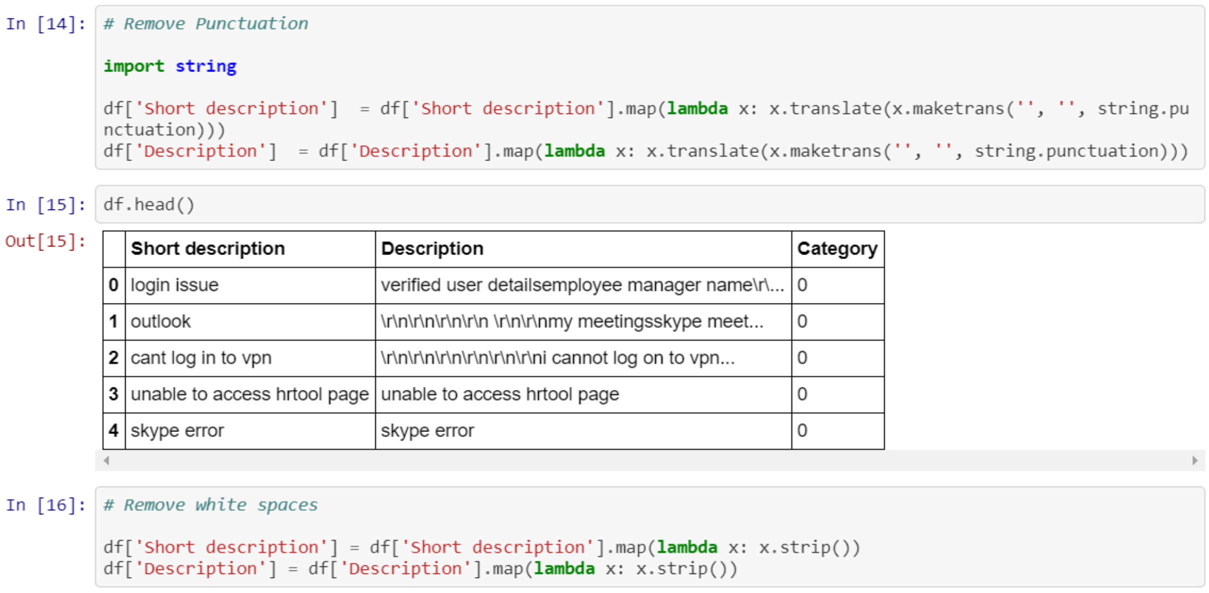
```
In [10]: # Change to lowercase
         df['Short description'] = df['Short description'].map(lambda x: x.lower())
         df['Description'] = df['Description'].map(lambda x: x.lower())
```

```
In [11]: df.head()
```

Out[11]:

	Short description	Description	Category
0	login issue	-verified user details.(employee# & manager na...	0
1	outlook	\r\n\r\n\r\n\r\n ,\r\n\r\nmy meetings/skype me...	0
2	cant log in to vpn	\r\n\r\n\r\n\r\n\r\n\r\ni cannot log on to vpn...	0
3	unable to access hr_tool page	unable to access hr_tool page	0
4	skype error	skype error	0

```
In [12]: # Remove numbers

         import re

         df['Short description'] = df['Short description'].map(lambda x: re.sub(r'\d+', '', x))
         df['Description'] = df['Description'].map(lambda x: re.sub(r'\d+', '', x))
```

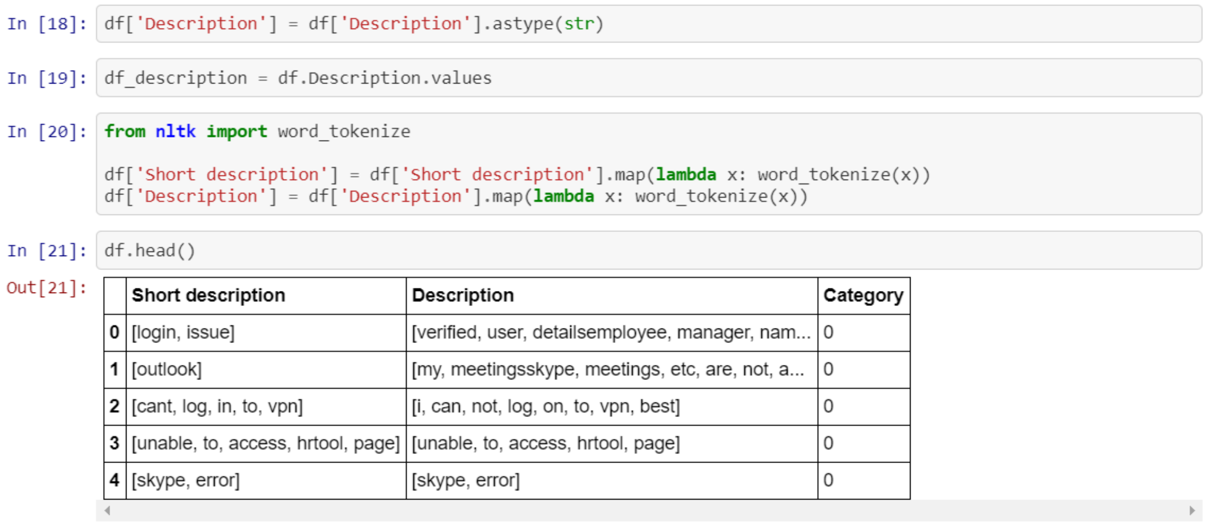


```
In [14]: # Remove Punctuation

         import string

         df['Short description']  = df['Short description'].map(lambda x: x.translate(x.maketrans('', '', string.pu
         nctuation)))
         df['Description']  = df['Description'].map(lambda x: x.translate(x.maketrans('', '', string.punctuation)))

In [15]: df.head()

Out[15]:
```

	Short description	Description	Category
0	login issue	verified user detailsemployee manager name\r\...	0
1	outlook	\r\n\r\n\r\n\r\n \r\n\r\nmy meetingsskype meet...	0
2	cant log in to vpn	\r\n\r\n\r\n\r\n\r\n\r\ni cannot log on to vpn...	0
3	unable to access hrtool page	unable to access hrtool page	0
4	skype error	skype error	0

```
In [16]: # Remove white spaces

         df['Short description'] = df['Short description'].map(lambda x: x.strip())
         df['Description'] = df['Description'].map(lambda x: x.strip())
```

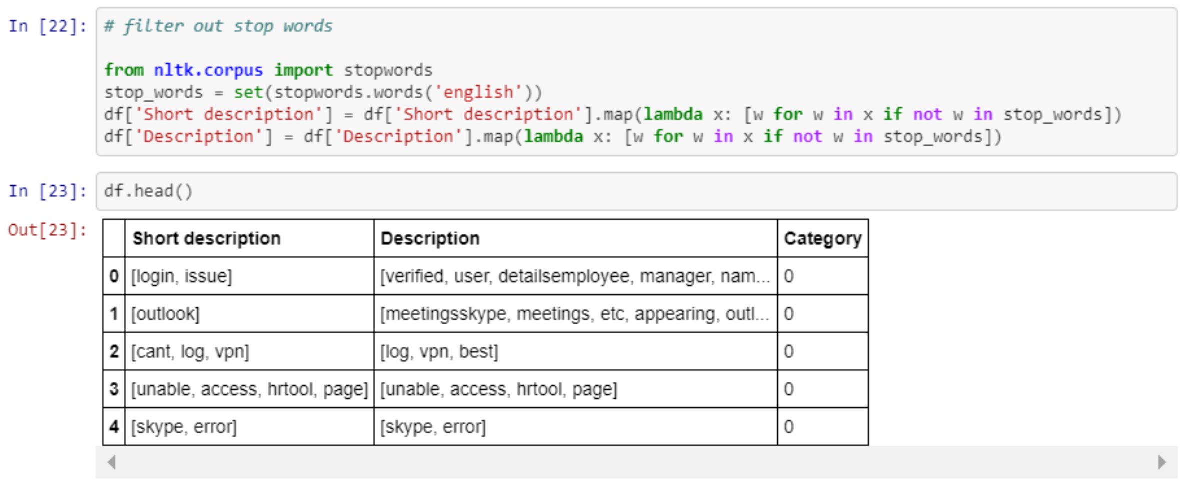
5. For our NLP model, we will now tokenize the data from our string into word tokens. Eg- login issue becomes [login, issue] – 2 tokens.



```
In [18]: df['Description'] = df['Description'].astype(str)

In [19]: df_description = df.Description.values

In [20]: from nltk import word_tokenize

         df['Short description'] = df['Short description'].map(lambda x: word_tokenize(x))
         df['Description'] = df['Description'].map(lambda x: word_tokenize(x))

In [21]: df.head()

Out[21]:
```

	Short description	Description	Category
0	[login, issue]	[verified, user, detailsemployee, manager, nam...	0
1	[outlook]	[my, meetingsskype, meetings, etc, are, not, a...	0
2	[cant, log, in, to, vpn]	[i, can, not, log, on, to, vpn, best]	0
3	[unable, to, access, hrtool, page]	[unable, to, access, hrtool, page]	0
4	[skype, error]	[skype, error]	0

6. Filter out the stop words from the Short Description & Description -



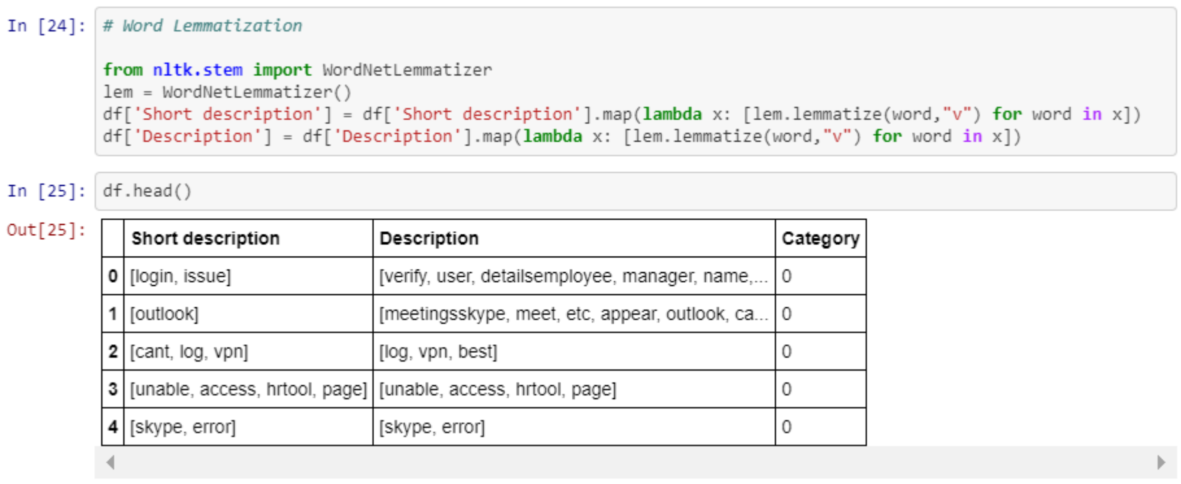
```
In [22]: # filter out stop words

         from nltk.corpus import stopwords
         stop_words = set(stopwords.words('english'))
         df['Short description'] = df['Short description'].map(lambda x: [w for w in x if not w in stop_words])
         df['Description'] = df['Description'].map(lambda x: [w for w in x if not w in stop_words])
```

```
In [23]: df.head()
```

Out[23]:

	Short description	Description	Category
0	[login, issue]	[verified, user, detailsemployee, manager, nam...	0
1	[outlook]	[meetingsskype, meetings, etc, appearing, outl...	0
2	[cant, log, vpn]	[log, vpn, best]	0
3	[unable, access, hrtool, page]	[unable, access, hrtool, page]	0
4	[skype, error]	[skype, error]	0

7. For text pre-processing, grouping together the different inflected forms of a word so they can be analysed as a single item. So that it brings context to the words. Basically, it links words with similar meaning to one word.



```
In [24]: # Word Lemmatization

         from nltk.stem import WordNetLemmatizer
         lem = WordNetLemmatizer()
         df['Short description'] = df['Short description'].map(lambda x: [lem.lemmatize(word,"v") for word in x])
         df['Description'] = df['Description'].map(lambda x: [lem.lemmatize(word,"v") for word in x])
```

```
In [25]: df.head()
```

Out[25]:

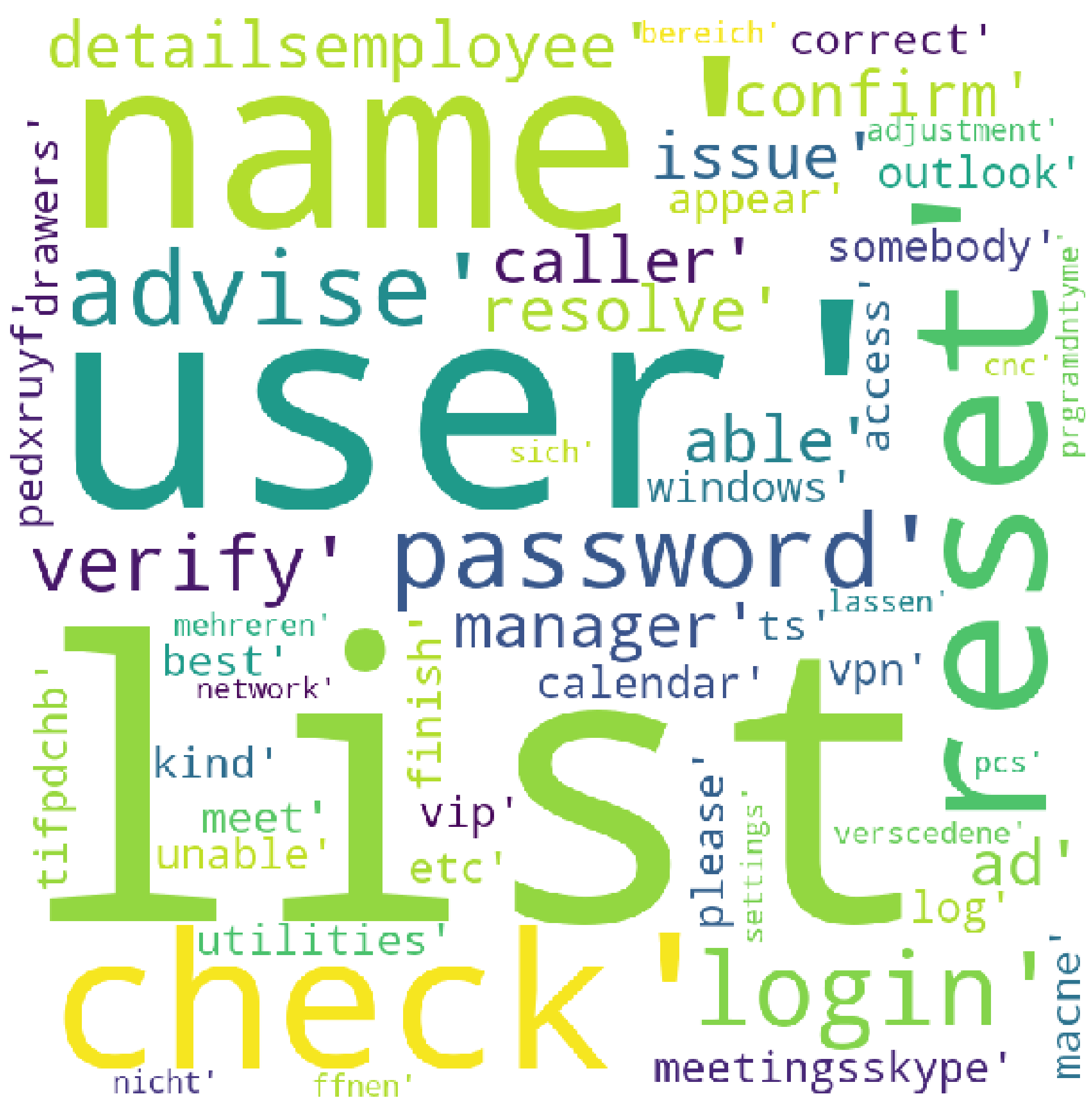
	Short description	Description	Category
0	[login, issue]	[verify, user, detailsemployee, manager, name,...	0
1	[outlook]	[meetingsskype, meet, etc, appear, outlook, ca...	0
2	[cant, log, vpn]	[log, vpn, best]	0
3	[unable, access, hrtool, page]	[unable, access, hrtool, page]	0
4	[skype, error]	[skype, error]	0

8. Using Word Cloud as data visualization technique for representing text data in which the size of each word indicates its frequency or importance.

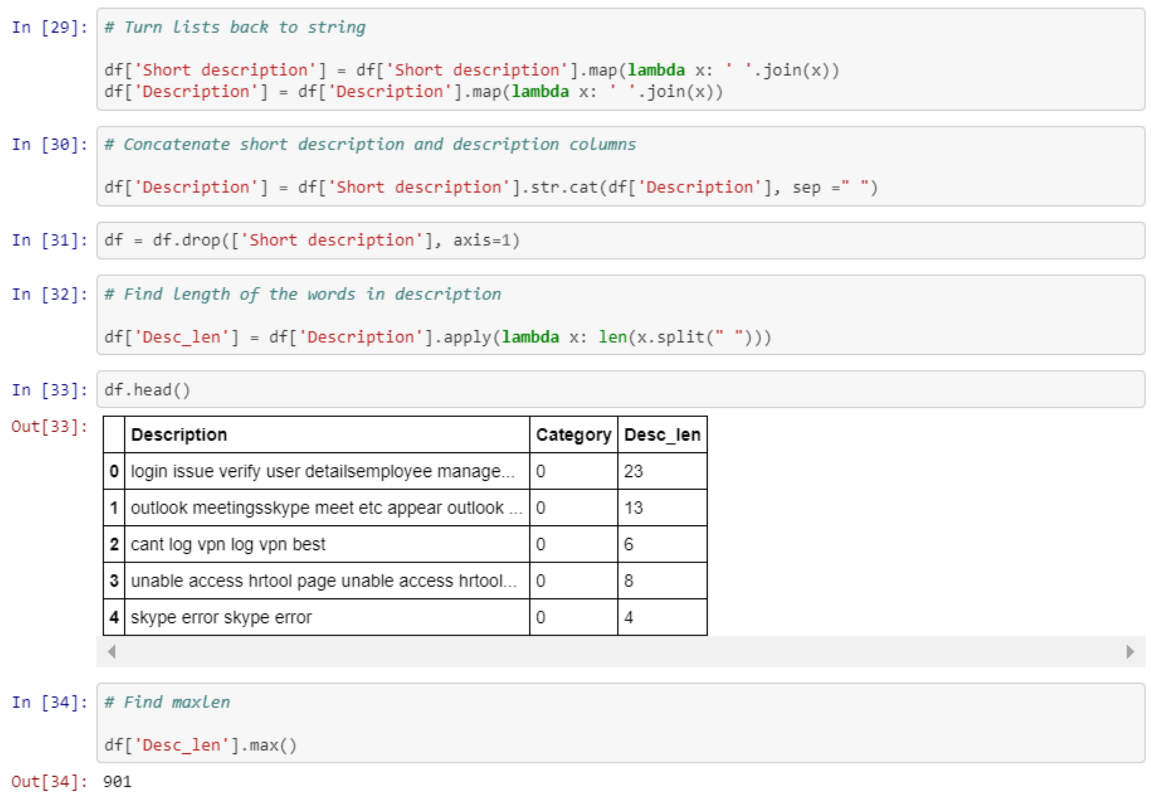
```
In [26]: from wordcloud import WordCloud

         val = str(df_description)

         wordcloud = WordCloud(width = 800, height = 800,
                         background_color ='white',
                         stopwords = stop_words,
                         min_font_size = 10).generate(val)
```

```
In [27]: # plot the WordCloud image

         plt.figure(figsize = (8, 8), facecolor = None)
         plt.imshow(wordcloud)
         plt.axis("off")
         plt.tight_layout(pad = 0)

         plt.show()
```

Output: -



9. Applying concatenation and getting length of Description in separate column.

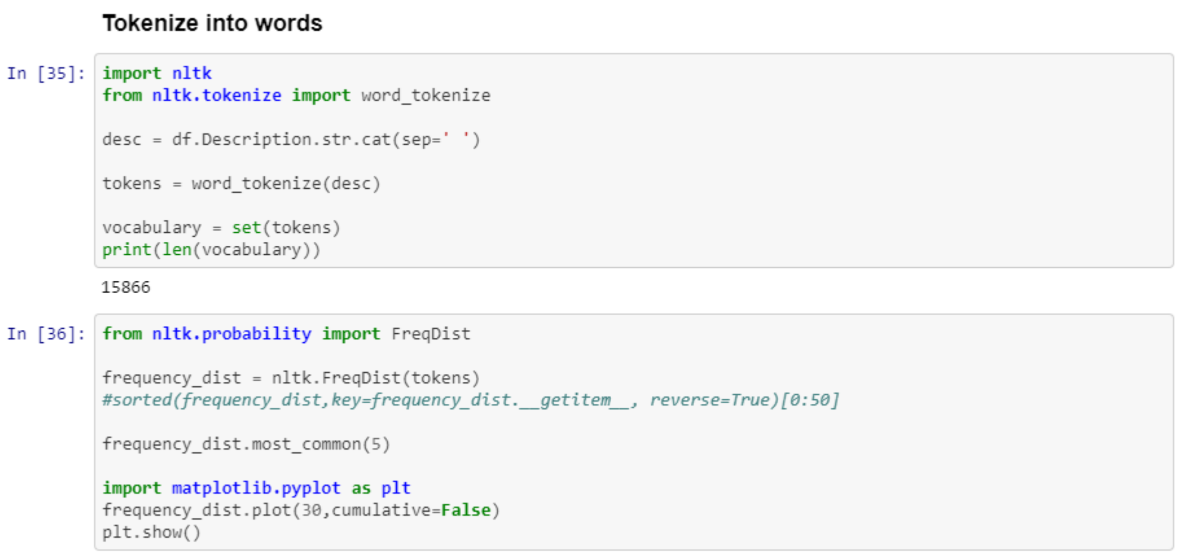


```
In [29]: # Turn lists back to string

         df['Short description'] = df['Short description'].map(lambda x: ' '.join(x))
         df['Description'] = df['Description'].map(lambda x: ' '.join(x))

In [30]: # Concatenate short description and description columns

         df['Description'] = df['Short description'].str.cat(df['Description'], sep =" ")

In [31]: df = df.drop(['Short description'], axis=1)

In [32]: # Find length of the words in description

         df['Desc_len'] = df['Description'].apply(lambda x: len(x.split(" ")))

In [33]: df.head()

Out[33]:
```

	Description	Category	Desc_len
0	login issue verify user detailsemployee manage...	0	23
1	outlook meetingsskype meet etc appear outlook ...	0	13
2	cant log vpn log vpn best	0	6
3	unable access hrtool page unable access hrtool...	0	8
4	skype error skype error	0	4

```
In [34]: # Find maxlen

         df['Desc_len'].max()

Out[34]: 901
```

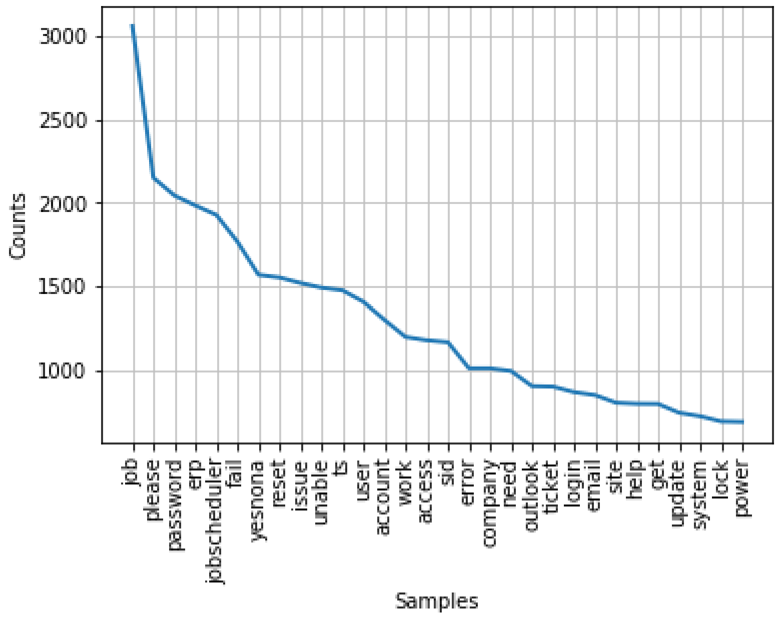
10. Each word needs to be captured and subjected to further analysis using Word Tokenization (used in NLP for splitting a large sample of text into words)



Tokenize into words

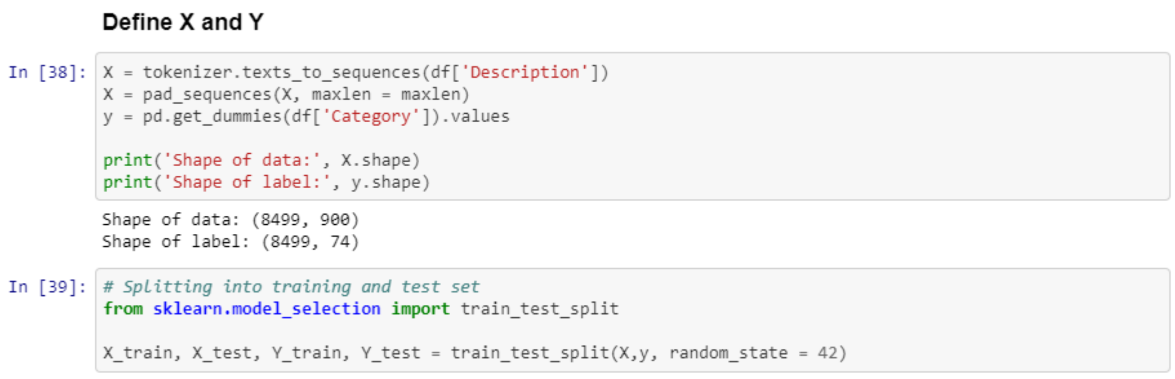
```
In [35]: import nltk
         from nltk.tokenize import word_tokenize

         desc = df.Description.str.cat(sep=' ')

         tokens = word_tokenize(desc)

         vocabulary = set(tokens)
         print(len(vocabulary))

         15866

In [36]: from nltk.probability import FreqDist

         frequency_dist = nltk.FreqDist(tokens)
         #sorted(frequency_dist,key=frequency_dist.__getitem__, reverse=True)[0:50]

         frequency_dist.most_common(5)

         import matplotlib.pyplot as plt
         frequency_dist.plot(30,cumulative=False)
         plt.show()
```

Output: -



11. Defining the model and using pad_sequences to ensure that all sequences in a list have the same length.

    Splitting the data into training and testing set.



Define X and Y

```
In [38]: X = tokenizer.texts_to_sequences(df['Description'])
         X = pad_sequences(X, maxlen = maxlen)
         y = pd.get_dummies(df['Category']).values

         print('Shape of data:', X.shape)
         print('Shape of label:', y.shape)

         Shape of data: (8499, 900)
         Shape of label: (8499, 74)
```

```
In [39]: # Splitting into training and test set
         from sklearn.model_selection import train_test_split

         X_train, X_test, Y_train, Y_test = train_test_split(X,y, random_state = 42)
```

CHOOSING MODEL:

Natural Language Processing (NLP) sits at the nexus of computer science and linguistics, defining the solutions for how machine and human languages can interact with one another. Functionally, NLP consumes human language by analysing and manipulating data (often in the form of text) to derive meaning. To do this, we need to convert data that is passed between humans into a numeric format that is machine readable. This process of encoding text is called Vectorization, and catalyses computational processes like applying mathematical rules and performing matrix operations that can produce valuable insights.

Now that we understand how computers consume text data, we can experiment using different models!

For the model approach, we use the following six steps:

1) Data preparation with text documents of incident records.

2) Document collection or Text corpus.

3) Data divided for training documents and test documents.

4) Text measurement.

5) Method selection based on the training documents.

(6) Model validation based on the test documents.

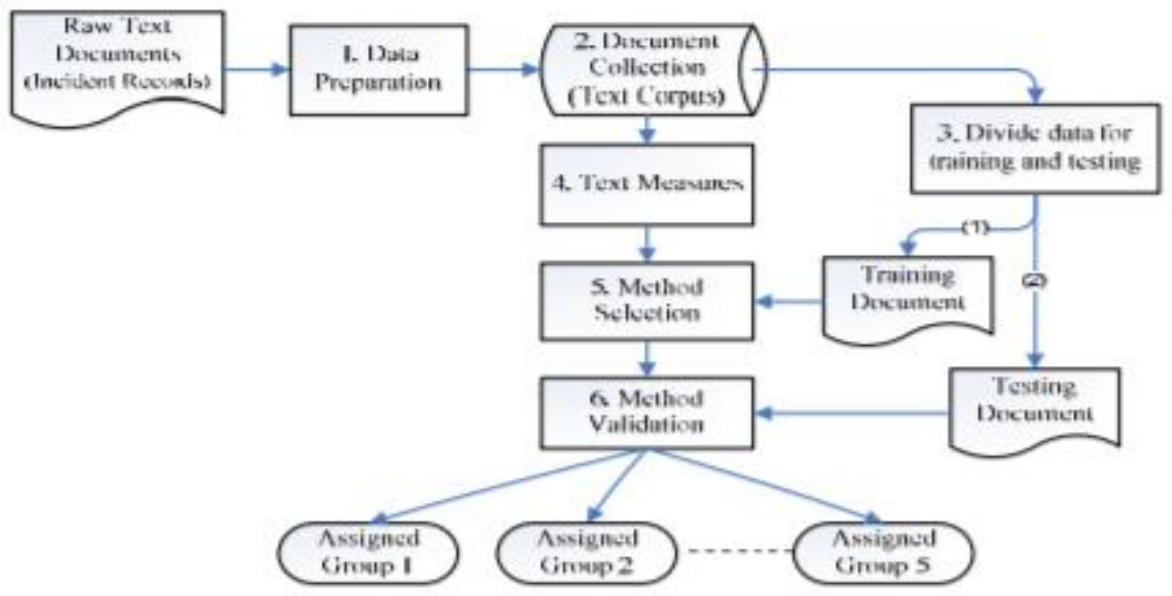


Figure 2. Data preparation procedure for text mining discovery algorithms

Model Selection:

Since we are dealing with sequential data, Recurrent Neural Networks (RNN) of which LSTM which is very good at holding long term memories can be used for this problem.

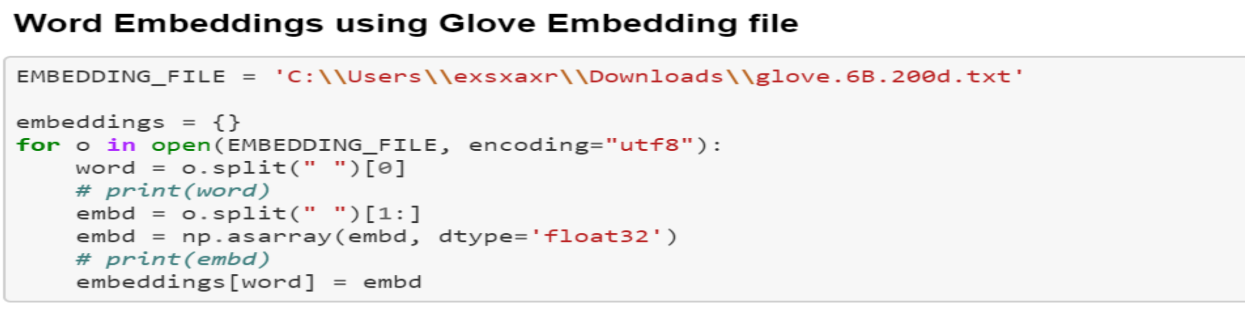
LSTM is a good choice for such sequences which have long term dependencies in it.

Model Building

Long Short-Term Memory networks — “LSTMs” — are a special kind of RNN, capable of learning long-term dependencies.

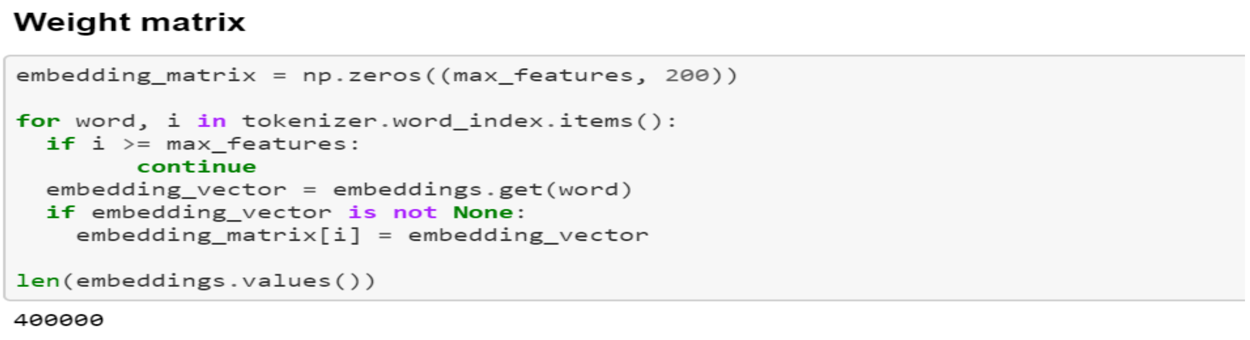
We tried to tackle the problem by using recurrent neural network and LSTM encoder and used LSTM layer in Keras to address the issue of long-term dependencies.

Using the knowledge from an external embedding can enhance the precision of your RNN because it integrates new information (lexical and semantic) about the words, an information that has been trained and distilled on a very large corpus of data. The pre-trained embedding we used is GloVe.



Word Embeddings using Glove Embedding file

```
EMBEDDING_FILE = 'C:\\Users\\exsxaxr\\Downloads\\glove.6B.200d.txt'

embeddings = {}
for o in open(EMBEDDING_FILE, encoding="utf8"):
    word = o.split(" ")[0]
    # print(word)
    embd = o.split(" ")[1:]
    embd = np.asarray(embd, dtype='float32')
    # print(embd)
    embeddings[word] = embd
```

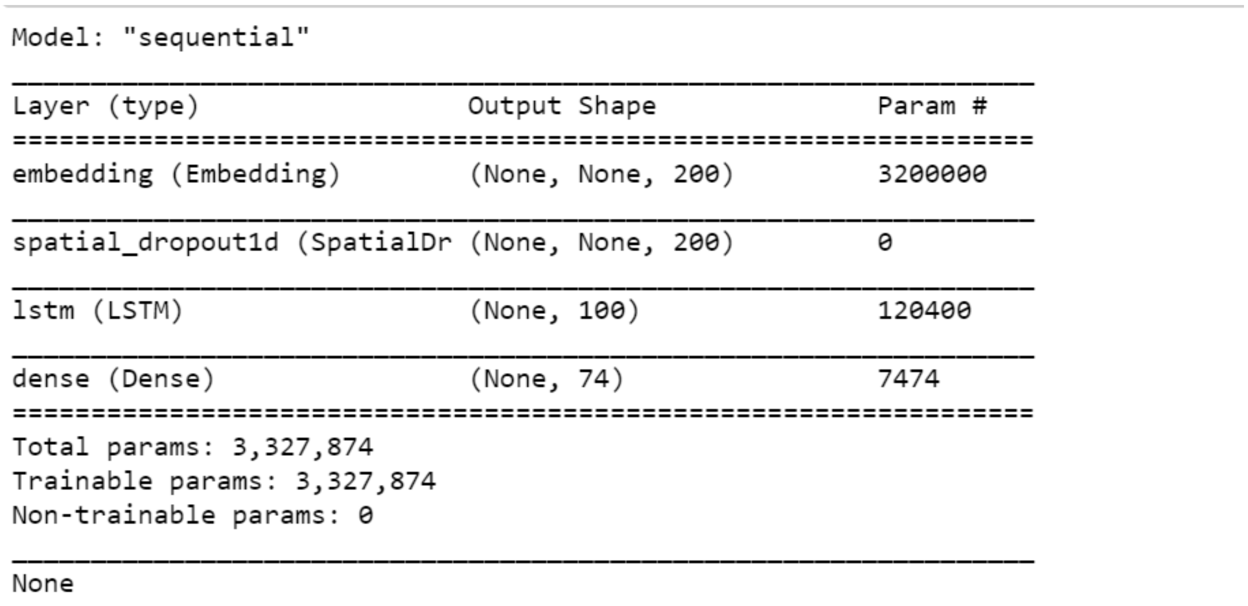


Weight matrix

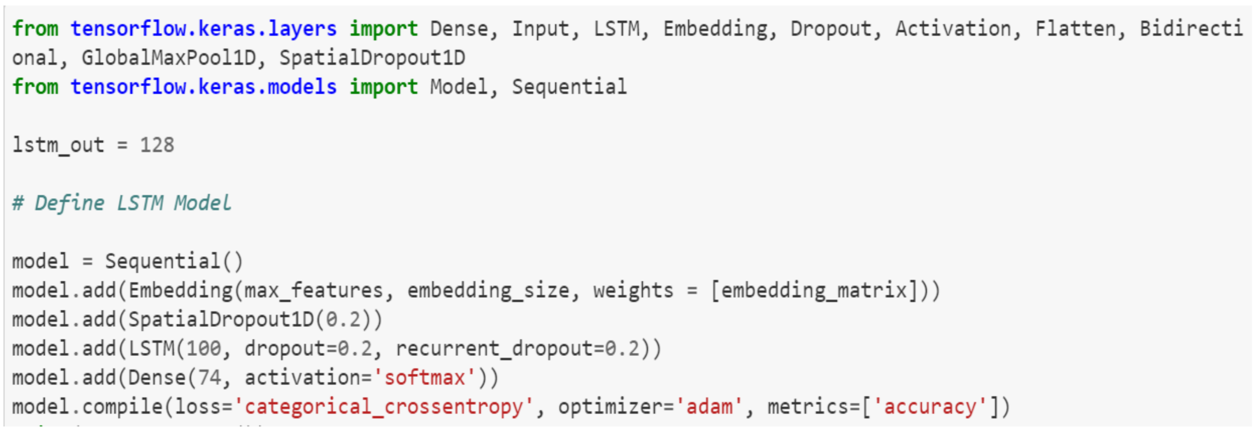
```
embedding_matrix = np.zeros((max_features, 200))

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

len(embeddings.values())
```

```
400000
```

Architecture:



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 200)         3200000
_________________________________________________________________
spatial_dropout1d (SpatialDr (None, None, 200)         0
_________________________________________________________________
lstm (LSTM)                  (None, 100)               120400
_________________________________________________________________
dense (Dense)                (None, 74)                7474
=================================================================
Total params: 3,327,874
Trainable params: 3,327,874
Non-trainable params: 0
_________________________________________________________________
None
```



```
from tensorflow.keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activation, Flatten, Bidirecti
onal, GlobalMaxPool1D, SpatialDropout1D
from tensorflow.keras.models import Model, Sequential

lstm_out = 128

# Define LSTM Model

model = Sequential()
model.add(Embedding(max_features, embedding_size, weights = [embedding_matrix]))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(74, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Model Performance:

Our model scoring and selection is based on the standard evaluation metrics Accuracy, Precision, and F1 score. We got an accuracy of 77% in the training set and 65% in the validation set.

The metrics are calculated as below:

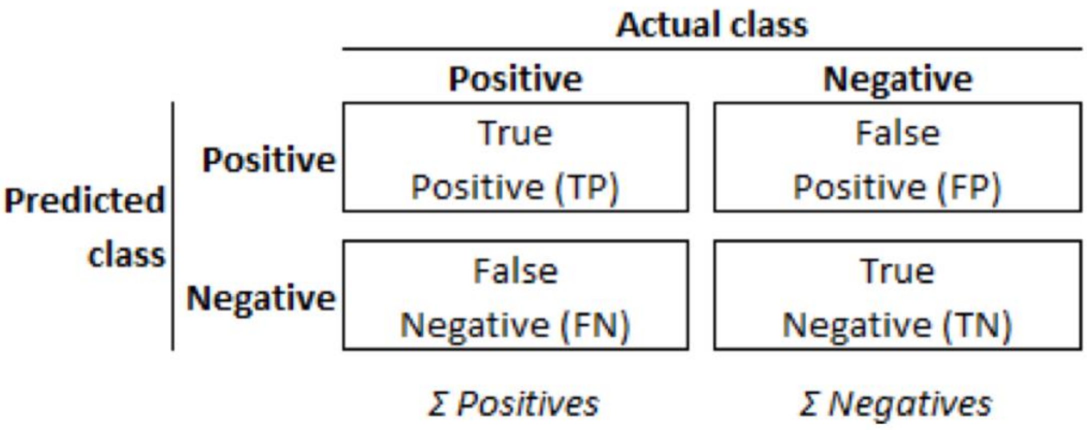
$$Accuracy = \frac{TP + TN}{all\ records}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 * TP}{2 * TP + FN + FP}$$

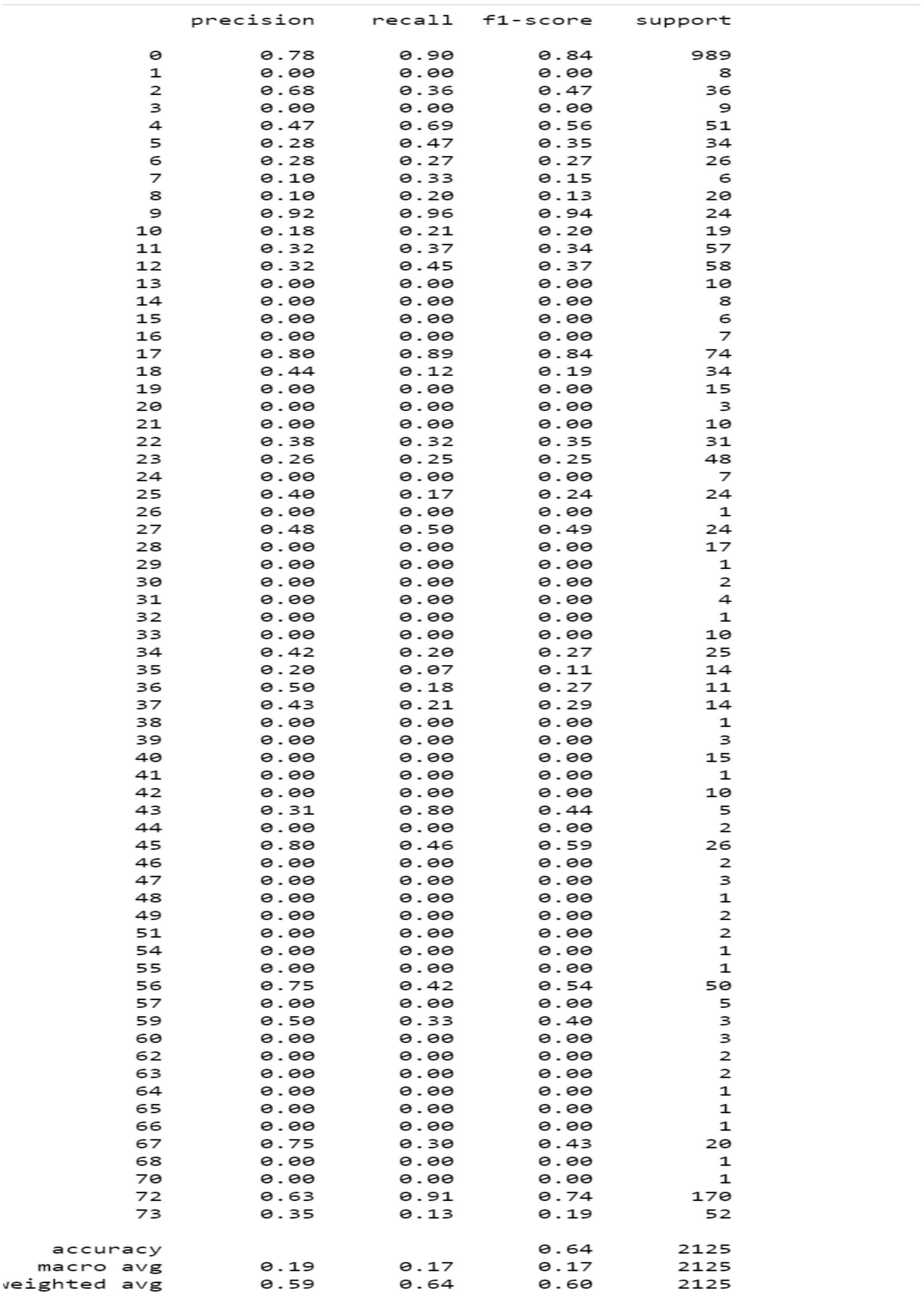
Confusion matrix of predicted vs. actual categorizations



		Actual class	
		Positive	**Negative**
Predicted class	**Positive**	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)
		Σ Positives	*Σ Negatives*

Classification report

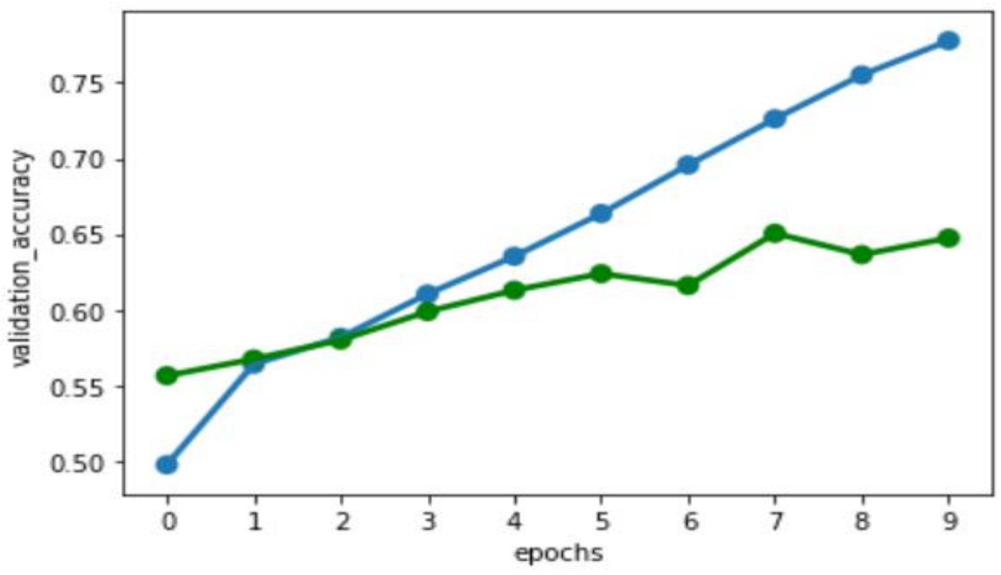
The classification report shows the class wise report of precision, recall and f1-score metrics.



```
              precision    recall  f1-score   support

           0       0.78      0.90      0.84       989
           1       0.00      0.00      0.00         8
           2       0.68      0.36      0.47        36
           3       0.00      0.00      0.00         9
           4       0.47      0.69      0.56        51
           5       0.28      0.47      0.35        34
           6       0.28      0.27      0.27        26
           7       0.10      0.33      0.15         6
           8       0.10      0.20      0.13        20
           9       0.92      0.96      0.94        24
          10       0.18      0.21      0.20        19
          11       0.32      0.37      0.34        57
          12       0.32      0.45      0.37        58
          13       0.00      0.00      0.00        10
          14       0.00      0.00      0.00         8
          15       0.00      0.00      0.00         6
          16       0.00      0.00      0.00         7
          17       0.80      0.89      0.84        74
          18       0.44      0.12      0.19        34
          19       0.00      0.00      0.00        15
          20       0.00      0.00      0.00         3
          21       0.00      0.00      0.00        10
          22       0.38      0.32      0.35        31
          23       0.26      0.25      0.25        48
          24       0.00      0.00      0.00         7
          25       0.40      0.17      0.24        24
          26       0.00      0.00      0.00         1
          27       0.48      0.50      0.49        24
          28       0.00      0.00      0.00        17
          29       0.00      0.00      0.00         1
          30       0.00      0.00      0.00         2
          31       0.00      0.00      0.00         4
          32       0.00      0.00      0.00         1
          33       0.00      0.00      0.00        10
          34       0.42      0.20      0.27        25
          35       0.20      0.07      0.11        14
          36       0.50      0.18      0.27        11
          37       0.43      0.21      0.29        14
          38       0.00      0.00      0.00         1
          39       0.00      0.00      0.00         3
          40       0.00      0.00      0.00        15
          41       0.00      0.00      0.00         1
          42       0.00      0.00      0.00        10
          43       0.31      0.80      0.44         5
          44       0.00      0.00      0.00         2
          45       0.80      0.46      0.59        26
          46       0.00      0.00      0.00         2
          47       0.00      0.00      0.00         3
          48       0.00      0.00      0.00         1
          49       0.00      0.00      0.00         2
          51       0.00      0.00      0.00         2
          54       0.00      0.00      0.00         1
          55       0.00      0.00      0.00         1
          56       0.75      0.42      0.54        50
          57       0.00      0.00      0.00         5
          59       0.50      0.33      0.40         3
          60       0.00      0.00      0.00         3
          62       0.00      0.00      0.00         2
          63       0.00      0.00      0.00         2
          64       0.00      0.00      0.00         1
          65       0.00      0.00      0.00         1
          66       0.00      0.00      0.00         1
          67       0.75      0.30      0.43        20
          68       0.00      0.00      0.00         1
          70       0.00      0.00      0.00         1
          72       0.63      0.91      0.74       170
          73       0.35      0.13      0.19        52

    accuracy                           0.64      2125
   macro avg       0.19      0.17      0.17      2125
veighted avg       0.59      0.64      0.60      2125
```

Training V/S validation accuracy:

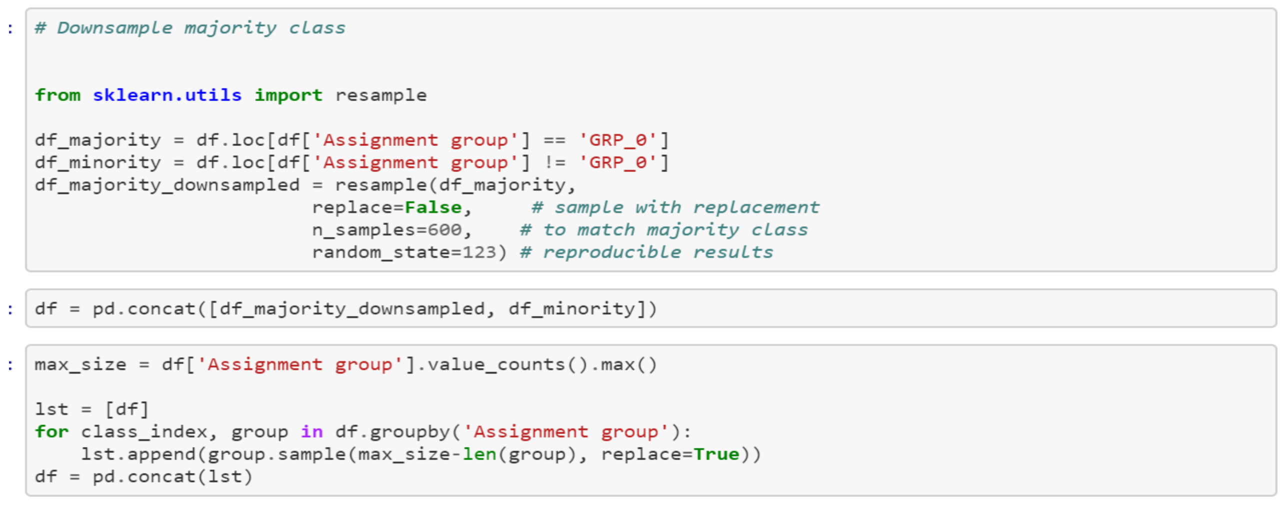
The below graph is showing the training and validation accuracy. The blue line denotes the training accuracy and green one denotes the validation accuracy.



Challenges in the model:

The overall accuracy of the model is getting affected due to the imbalanced structure of the dataset. As few classes are having very small amount of samples, the model may be biased.

For dealing this imbalance, we have sampled the data by under sampling the majority class and oversampling the minority.



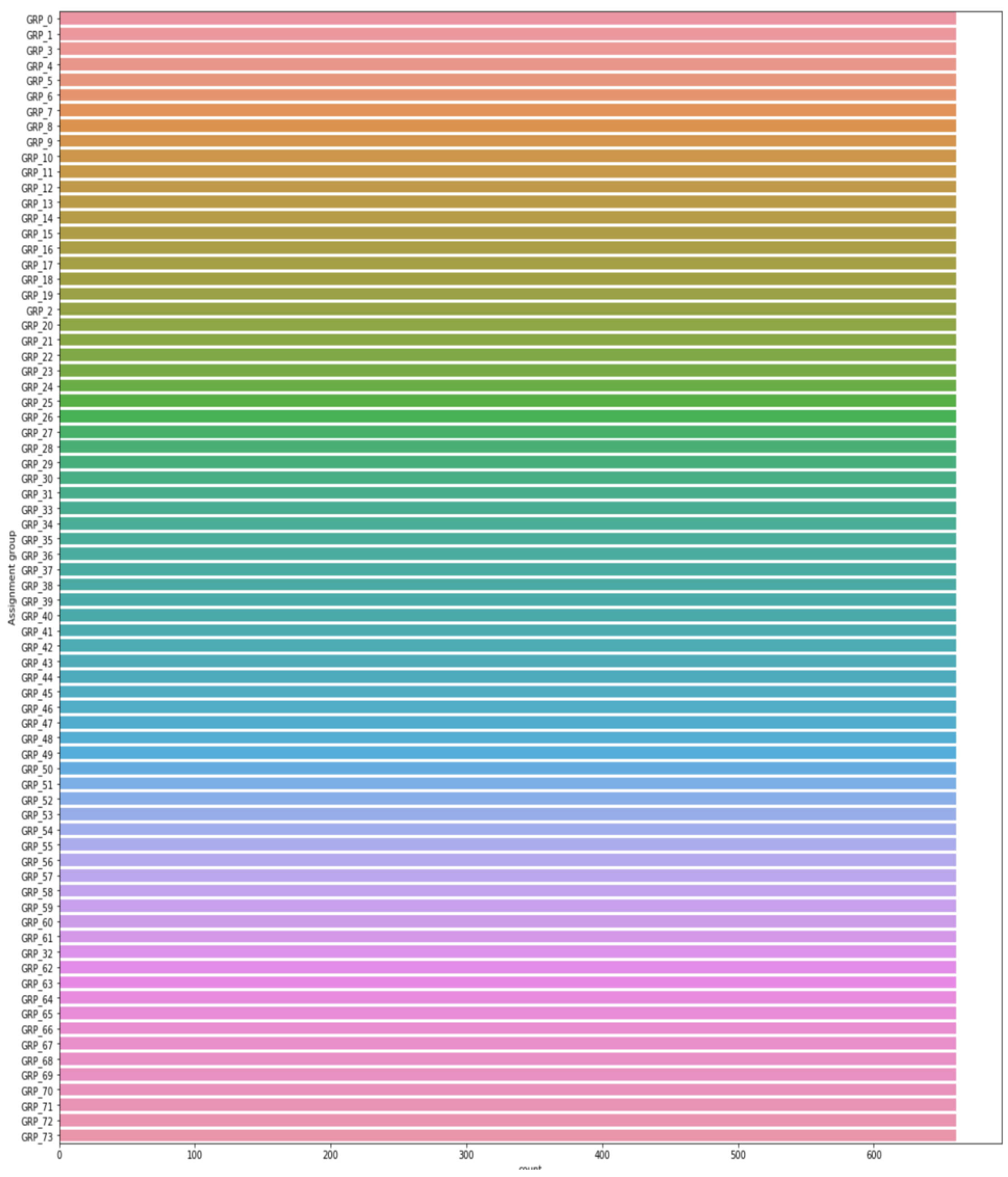
```
# Downsample majority class


from sklearn.utils import resample

df_majority = df.loc[df['Assignment group'] == 'GRP_0']
df_minority = df.loc[df['Assignment group'] != 'GRP_0']
df_majority_downsampled = resample(df_majority,
                        replace=False,     # sample with replacement
                        n_samples=600,    # to match majority class
                        random_state=123) # reproducible results
```

```
df = pd.concat([df_majority_downsampled, df_minority])
```

```
max_size = df['Assignment group'].value_counts().max()

lst = [df]
for class_index, group in df.groupby('Assignment group'):
    lst.append(group.sample(max_size-len(group), replace=True))
df = pd.concat(lst)
```

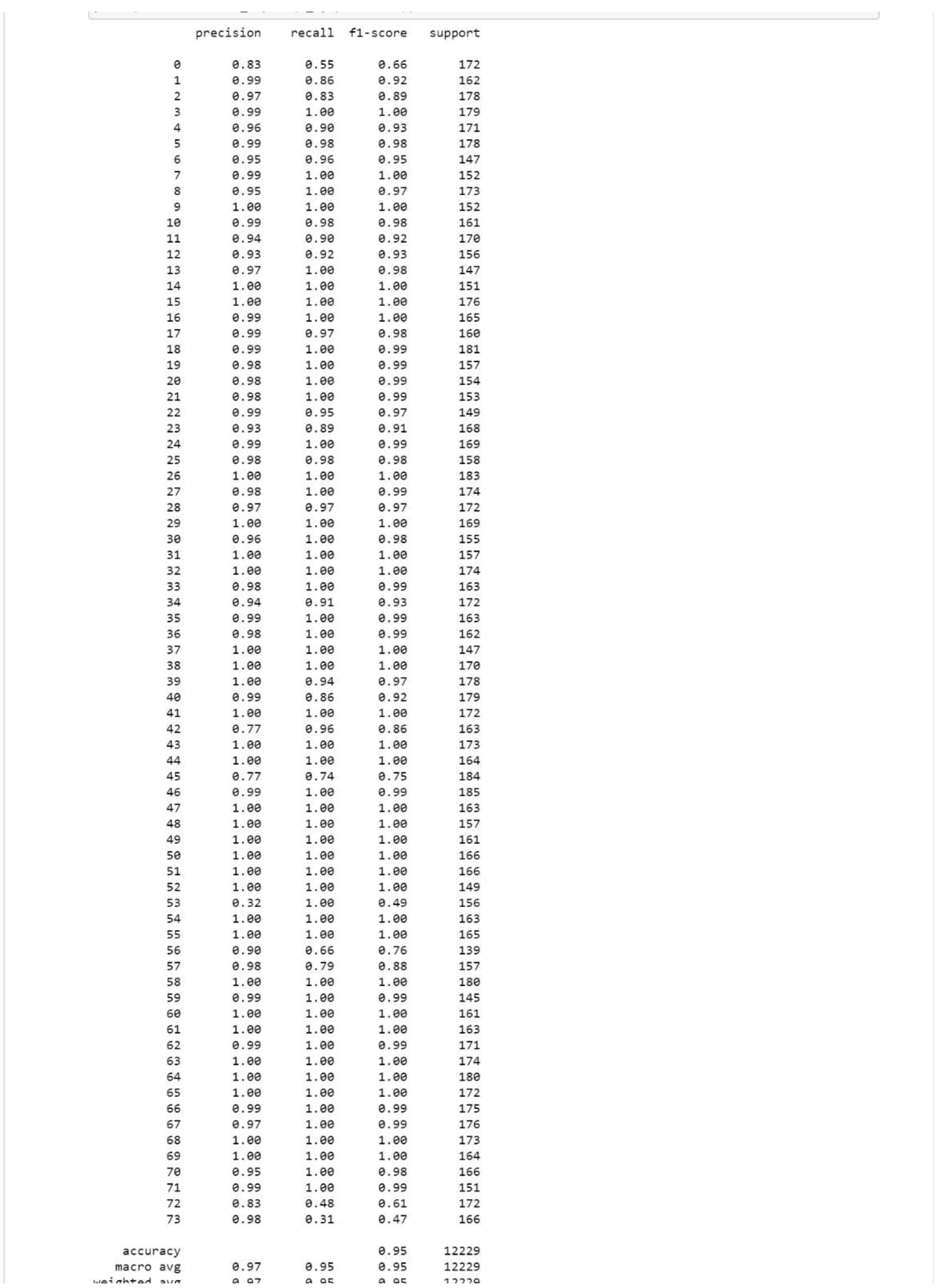
Now all the classes are having equal number of samples.

```
]: <matplotlib.axes._subplots.AxesSubplot at 0x225b95f8508>
```



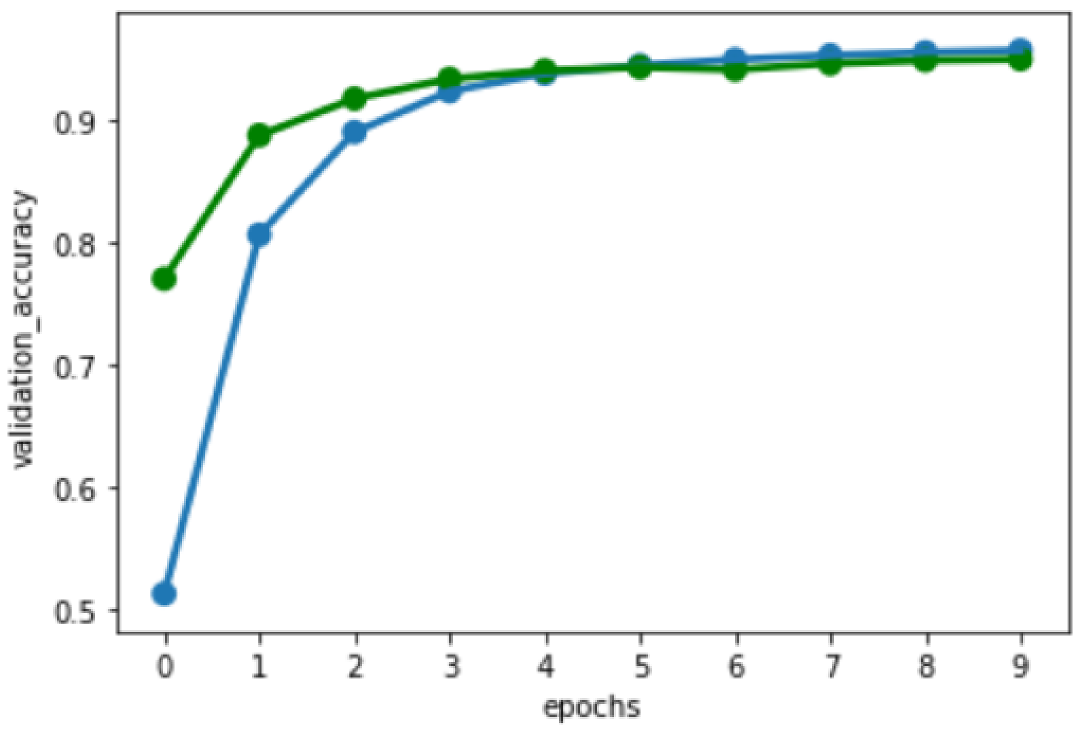
The model performance has been increased after sampling the data and we are getting an accuracy of 95% in the training set and about 94% in the validation set.

Classification report after sampling:



```
              precision    recall  f1-score   support

           0       0.83      0.55      0.66       172
           1       0.99      0.86      0.92       162
           2       0.97      0.83      0.89       178
           3       0.99      1.00      1.00       179
           4       0.96      0.90      0.93       171
           5       0.99      0.98      0.98       178
           6       0.95      0.96      0.95       147
           7       0.99      1.00      1.00       152
           8       0.95      1.00      0.97       173
           9       1.00      1.00      1.00       152
          10       0.99      0.98      0.98       161
          11       0.94      0.90      0.92       170
          12       0.93      0.92      0.93       156
          13       0.97      1.00      0.98       147
          14       1.00      1.00      1.00       151
          15       1.00      1.00      1.00       176
          16       0.99      1.00      1.00       165
          17       0.99      0.97      0.98       160
          18       0.99      1.00      0.99       181
          19       0.98      1.00      0.99       157
          20       0.98      1.00      0.99       154
          21       0.98      1.00      0.99       153
          22       0.99      0.95      0.97       149
          23       0.93      0.89      0.91       168
          24       0.99      1.00      0.99       169
          25       0.98      0.98      0.98       158
          26       1.00      1.00      1.00       183
          27       0.98      1.00      0.99       174
          28       0.97      0.97      0.97       172
          29       1.00      1.00      1.00       169
          30       0.96      1.00      0.98       155
          31       1.00      1.00      1.00       157
          32       1.00      1.00      1.00       174
          33       0.98      1.00      0.99       163
          34       0.94      0.91      0.93       172
          35       0.99      1.00      0.99       163
          36       0.98      1.00      0.99       162
          37       1.00      1.00      1.00       147
          38       1.00      1.00      1.00       170
          39       1.00      0.94      0.97       178
          40       0.99      0.86      0.92       179
          41       1.00      1.00      1.00       172
          42       0.77      0.96      0.86       163
          43       1.00      1.00      1.00       173
          44       1.00      1.00      1.00       164
          45       0.77      0.74      0.75       184
          46       0.99      1.00      0.99       185
          47       1.00      1.00      1.00       163
          48       1.00      1.00      1.00       157
          49       1.00      1.00      1.00       161
          50       1.00      1.00      1.00       166
          51       1.00      1.00      1.00       166
          52       1.00      1.00      1.00       149
          53       0.32      1.00      0.49       156
          54       1.00      1.00      1.00       163
          55       1.00      1.00      1.00       165
          56       0.90      0.66      0.76       139
          57       0.98      0.79      0.88       157
          58       1.00      1.00      1.00       180
          59       0.99      1.00      0.99       145
          60       1.00      1.00      1.00       161
          61       1.00      1.00      1.00       163
          62       0.99      1.00      0.99       171
          63       1.00      1.00      1.00       174
          64       1.00      1.00      1.00       180
          65       1.00      1.00      1.00       172
          66       0.99      1.00      0.99       175
          67       0.97      1.00      0.99       176
          68       1.00      1.00      1.00       173
          69       1.00      1.00      1.00       164
          70       0.95      1.00      0.98       166
          71       0.99      1.00      0.99       151
          72       0.83      0.48      0.61       172
          73       0.98      0.31      0.47       166

    accuracy                           0.95     12229
   macro avg       0.97      0.95      0.95     12229
weighted avg       0.97      0.95      0.95     12229
```

Training V/S Validation accuracy after sampling:



Observations:

The model without sampling was not performing well as the data was not evenly distributed among the classes. As few classes where having very less number of samples, the accuracy, precision and recall metrics were affected.

After resampling the data, the accuracy of the model increased up to 95% and precision, recall and f1-score values also showing good.

Fine tuning and improvements:

The model can be further improved by adding a bi-directional LSTM layer instead of normal LSTM layer.

There are few non-English sentences in the dataset which can be ignored.

The current most frequently occurring word in the dataset is “please” which can be ignored as it is irrelevant to the issue.

Adding few more layers into the model may increase the accuracy.