AIML Online Capstone Project

AUTOMATIC TICKET ASSIGNMENT

INTERIM REPORT

TABLE OF CONTENT

Real Problem

In any IT industry, Incident Management plays an important role in delivering a quality support to customers. An incident ticket is created by various groups of people within the organization to resolve an issue as quickly as possible based on its severity. Whenever an incident is created, it reaches the Service desk team, and then it gets assigned to the respective teams to work on the incident. The Service Desk team (L1/L2) will perform basic analysis on the user's requirement, identify the issue based on given descriptions and assign it to the respective teams.

The manual assignment of these incidents might have the below disadvantages:

- More resource usage and expenses.
- Human errors - Incidents get assigned to the wrong assignment groups
- Delay in assigning the tickets
- More resolution times
- If a particular ticket takes more time in analysis, other productive tasks get affected for the Service Desk

Objective

From the given problem description, we could see that the existing system can assign 75% of the tickets correctly. So, our objective here is to build an AI-based classifier model to assign the tickets to the right functional groups by analyzing the given description with an accuracy of at least 80%.

1. Summary of the problem statement, data, and findings:

We have received unstructured data as our dataset. The dataset has 8500 rows and 4 columns. The 4 columns are:

1. **Short Description:** This column gives us a glance at the broad categorization of the complaint. Example: login issue, outlook, can't log in to VPN, unable to access hr_tool page, skype error
2. **Description:** This column gives us a little more detailed insight into the complaint that the end-user has. It talks about a brief description of the complaint/issue the end-user is facing. For example, my meetings/skype meetings are not appearing in my outlook calendar; can somebody please advise how to correct this?
3. **Caller:** This column has the end user's name who is filing the complaint or raising the request. Example: spxjnwir pjlcoqds, hmjdrvpb komuaywn
4. **Assignment Group:** This column talks about the category/group into which the complaint is classified for it to be directed to the right department for the issue to be resolved. Example: GRP_0, GRP_1

Observation in the dataset:

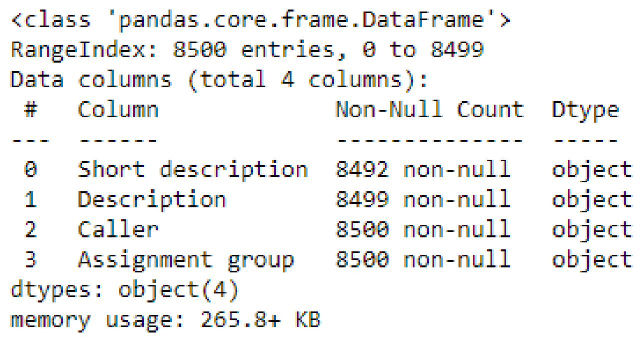
- The total number of incidents reported in the dataset is 8500.
- Caller names are randomly (may not be useful for training data).
- The dataset has English and German Language words.
- Email/chat format found in the description.
- There are a total number of 8 null records in the Short Description column and there is 1 null record in the Description column. There are no null records in the Caller and Assignment Group columns.
- There are a total of 74 different groups for the Assignment Group column.
- Approximately 50% of the dataset comprises complaints that are corresponding to GRP_0.
- There are a few rows of data that have the same text for the Short Description and the Description column.
- Few words were combined.
- Spelling mistakes and typo errors are found.

2. Summary of the Approach to EDA and Pre-processing:

a) Data Cleaning

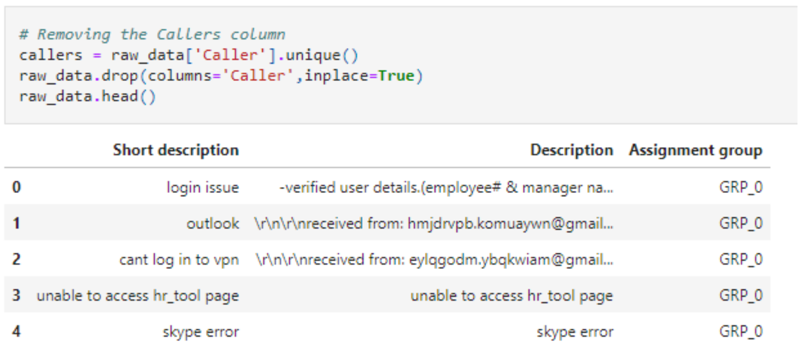
We performed the stated actions

- We started exploratory data cleaning by getting the basic information of our dataset.



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

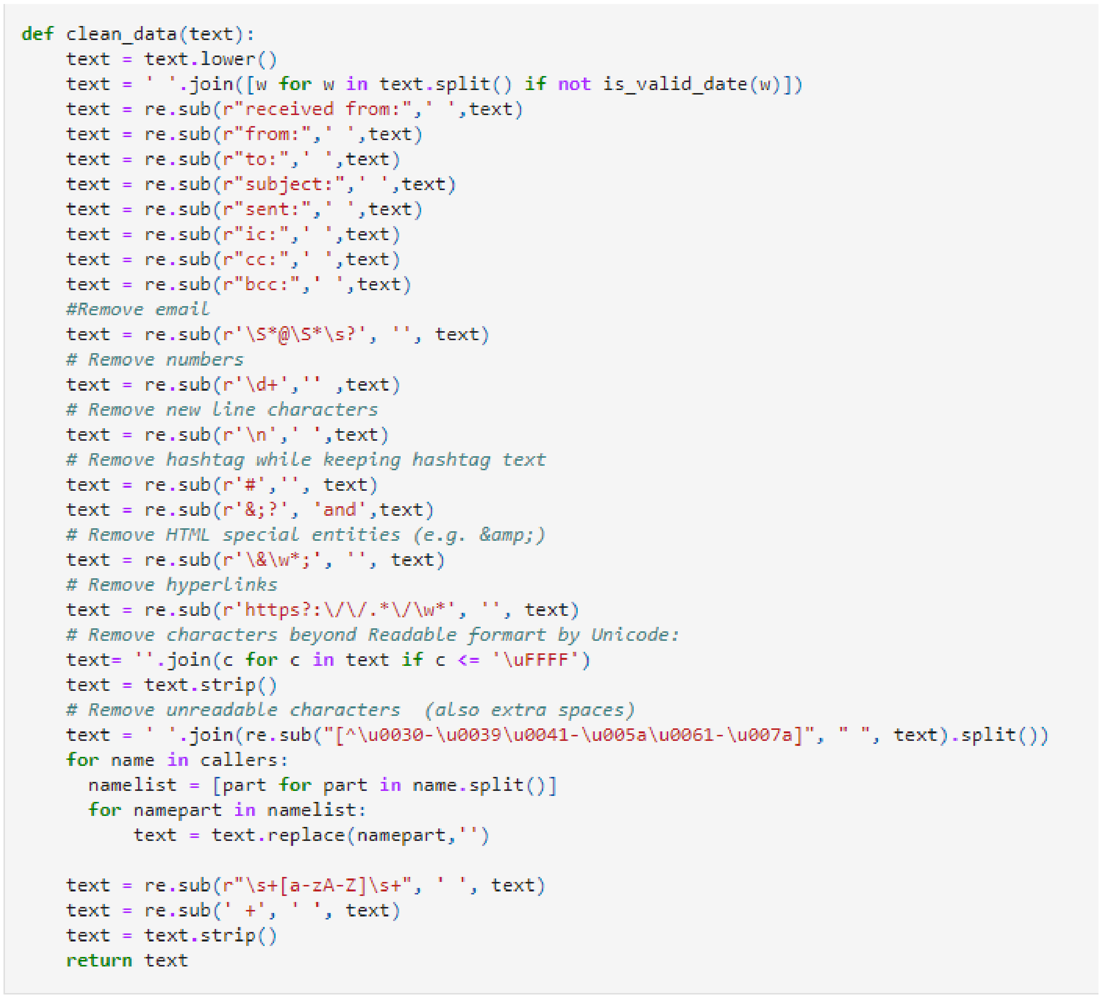
- Dropping the Callers Column from Dataset



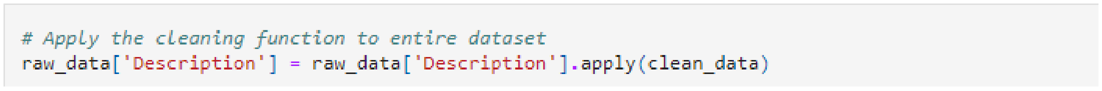
```
# Removing the Callers column
callers = raw_data['Caller'].unique()
raw_data.drop(columns='Caller',inplace=True)
raw_data.head()
```

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

- Removal of special characters, trailing spaces, numbers
- Converting strings to lower case

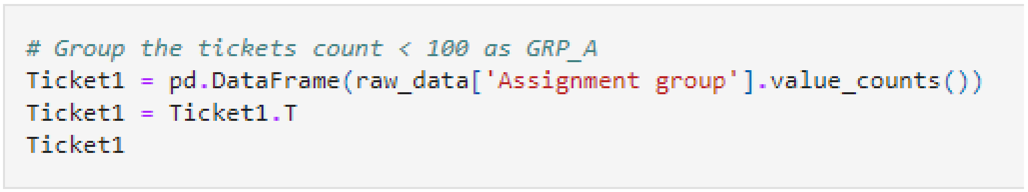


```python
def clean_data(text):
    text = text.lower()
    text = ' '.join([w for w in text.split() if not is_valid_date(w)])
    text = re.sub(r"received from:",' ',text)
    text = re.sub(r"from:",' ',text)
    text = re.sub(r"to:",' ',text)
    text = re.sub(r"subject:",' ',text)
    text = re.sub(r"sent:",' ',text)
    text = re.sub(r"ic:",' ',text)
    text = re.sub(r"cc:",' ',text)
    text = re.sub(r"bcc:",' ',text)
    #Remove email
    text = re.sub(r'\S*@\S*\s?', '', text)
    # Remove numbers
    text = re.sub(r'\d+','' ,text)
    # Remove new line characters
    text = re.sub(r'\n',' ',text)
    # Remove hashtag while keeping hashtag text
    text = re.sub(r'#','', text)
    text = re.sub(r'&;?', 'and',text)
    # Remove HTML special entities (e.g. &amp;)
    text = re.sub(r'\&\w*;', '', text)
    # Remove hyperlinks
    text = re.sub(r'https?:\/\/.*\/\w*', '', text)
    # Remove characters beyond Readable formart by Unicode:
    text= ''.join(c for c in text if c <= '\uFFFF')
    text = text.strip()
    # Remove unreadable characters  (also extra spaces)
    text = ' '.join(re.sub("[^\u0030-\u0039\u0041-\u005a\u0061-\u007a]", " ", text).split())
    for name in callers:
      namelist = [part for part in name.split()]
      for namepart in namelist:
          text = text.replace(namepart,'')

    text = re.sub(r"\s+[a-zA-Z]\s+", ' ', text)
    text = re.sub(' +', ' ', text)
    text = text.strip()
    return text
```



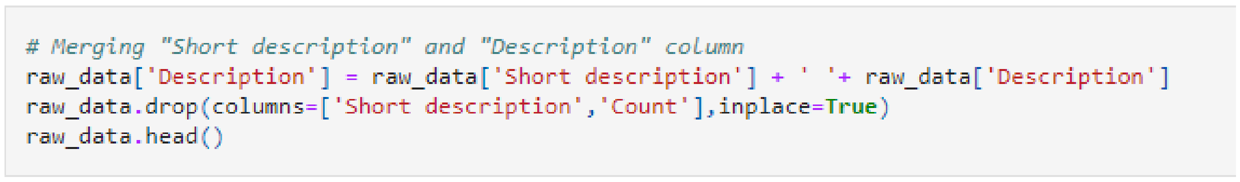
```python
# Apply the cleaning function to entire dataset
raw_data['Description'] = raw_data['Description'].apply(clean_data)
```

- Merging small groups

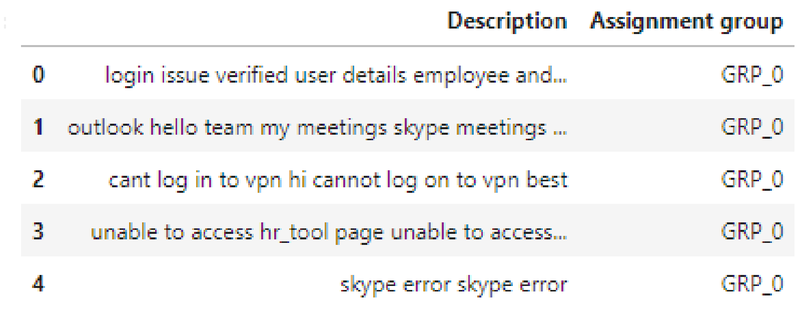


```python
# Group the tickets count < 100 as GRP_A
Ticket1 = pd.DataFrame(raw_data['Assignment group'].value_counts())
Ticket1 = Ticket1.T
Ticket1
```

- Merging both Description Columns



```python
# Merging "Short description" and "Description" column
raw_data['Description'] = raw_data['Short description'] + ' '+ raw_data['Description']
raw_data.drop(columns=['Short description','Count'],inplace=True)
raw_data.head()
```



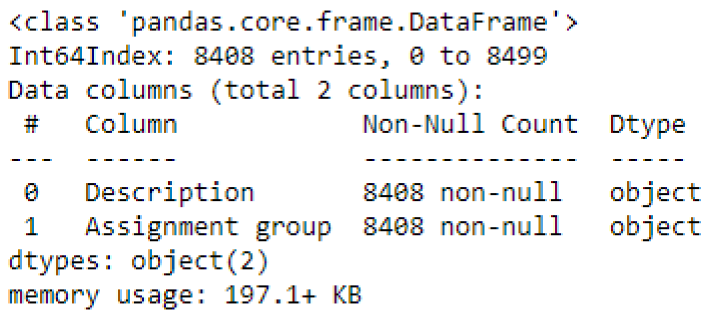
	Description	Assignment group
0	login issue verified user details employee and...	GRP_0
1	outlook hello team my meetings skype meetings ...	GRP_0
2	cant log in to vpn hi cannot log on to vpn best	GRP_0
3	unable to access hr_tool page unable to access...	GRP_0
4	skype error skype error	GRP_0

Observations:

- The entire dataset is converted into lower case.
- User's email addresses will add NO value to our analysis, even though the user id is given in the caller column. So, all email addresses are removed from the dataset
- All numerals are removed because they were dominating the dataset if we were converting them into their word representation otherwise.
- All punctuation marks are removed which used to be a hindrance in lemmatization.
- All occurrences of more than one blank space, horizontal tab spaces, new line breaks, etc. have been replaced with a single blank space.

b) EDA

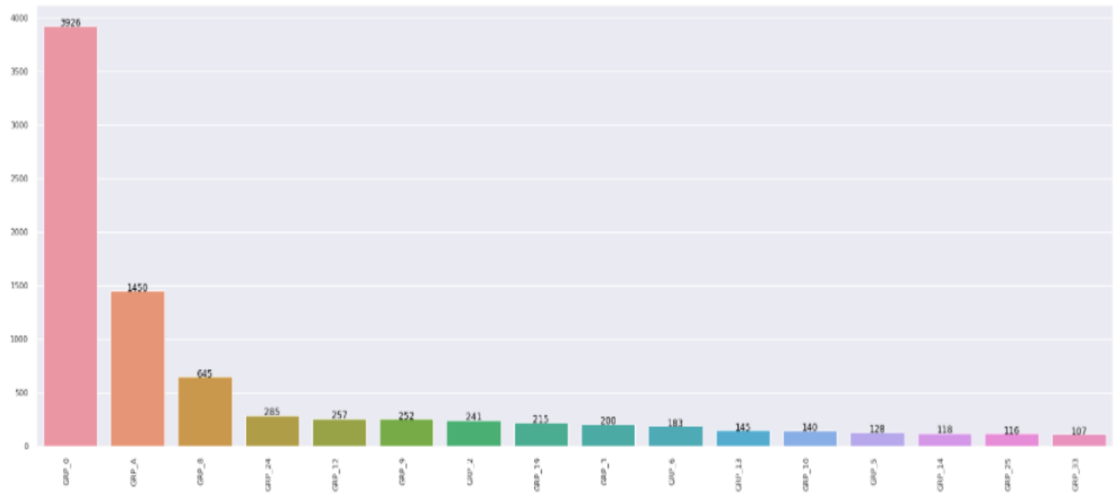
- We merged all the small groups into one group. We also converted the entire dataset into lower case. Here is the new information on the new dataset.



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8408 entries, 0 to 8499
Data columns (total 2 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Description       8408 non-null   object
 1   Assignment group  8408 non-null   object
dtypes: object(2)
memory usage: 197.1+ KB
```

- The assignment group was analyzed again and it was 16 groups.

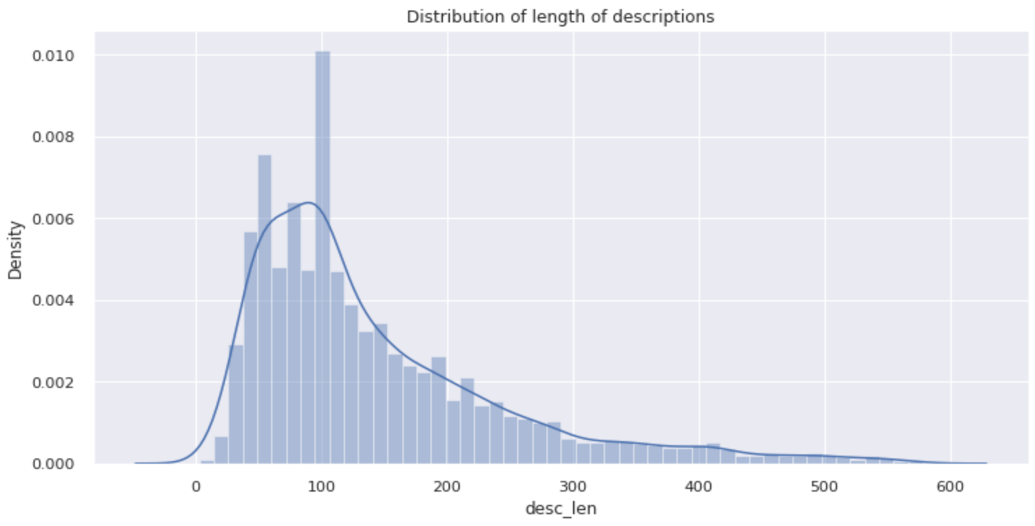
- As we observe in the below bar plot, the new dataset is extremely right-skewed. GRP_0, GRP_A, and GRP_8 together make to around 70% of the available data.



Observation:

- The Target class distribution is extremely right-skewed.
- GRP_0 has max occurrences (amounting to 3926) which makes for almost ~50% of the data.
- The merged entries GRP_A have around 1450 occurrences.
- GRP_0, GRP_A, and GRP_8 together make to around 70% of the available data.

We then figured out the length of each description. Here is the analysis.



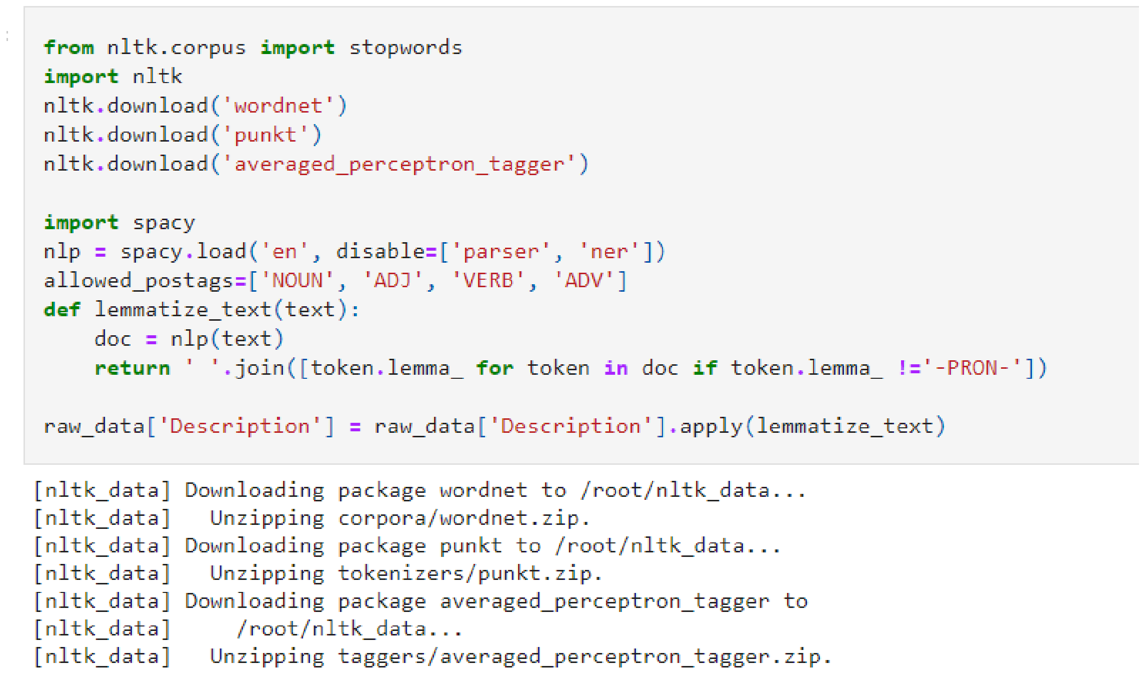
Summary of the EDA:

- There is no clear pattern visible to describe the relation between group and description lengths.
- It is however visible that groups have outliers and some groups fall in the lower range while most of them have lengths in the range of 100 to 400.
- Half of the records have less than 120 words.
- 90% of descriptions have less than 620 words.
- The remaining 10% records are very long and extremely skewed in comparison with the remaining portion.

c) Text-Pre-processing

Lemmatization & Stop words removal

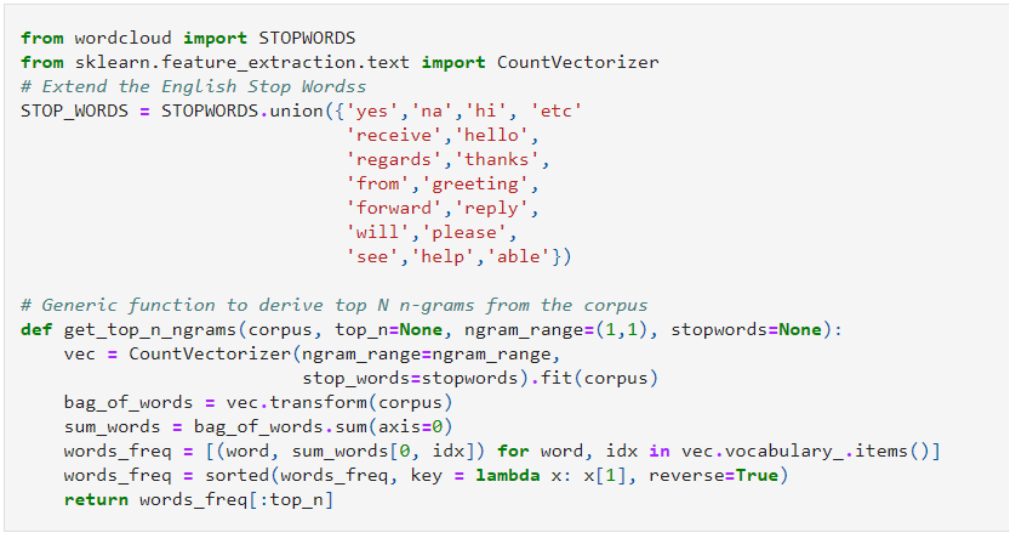
- Stop words have been removed using nltk corpus modules.
- Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item. Lemmatization is similar to Stemming but it brings context to the words. So, it links words with similar meanings to one word.
- Here we have preferred Lemmatization over Stemming because lemmatization does morphological analysis of the words.



```
from nltk.corpus import stopwords
import nltk
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

import spacy
nlp = spacy.load('en', disable=['parser', 'ner'])
allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']
def lemmatize_text(text):
    doc = nlp(text)
    return ' '.join([token.lemma_ for token in doc if token.lemma_ !='-PRON-'])

raw_data['Description'] = raw_data['Description'].apply(lemmatize_text)
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
```

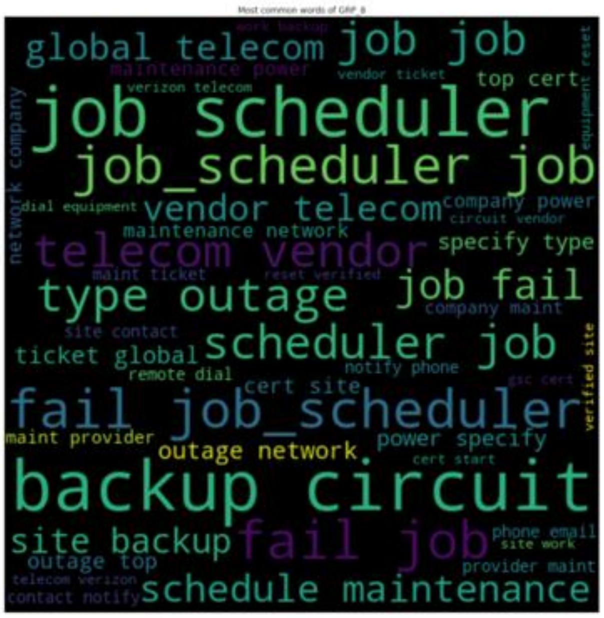
- We removed all the stop words to better understand our description and n-gram analysis.

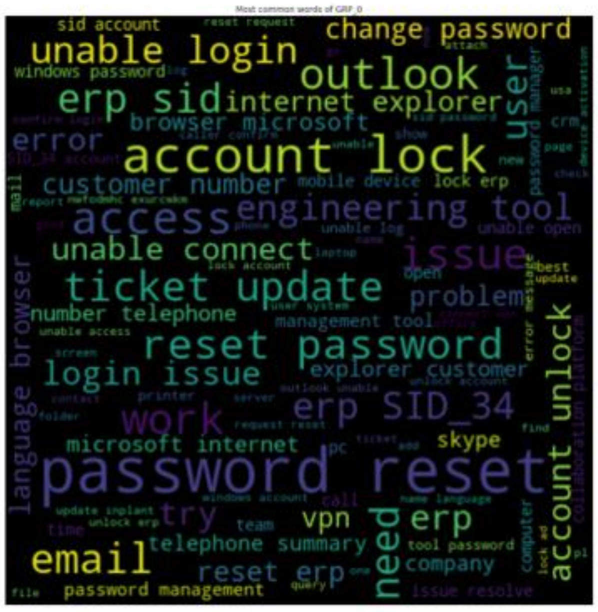


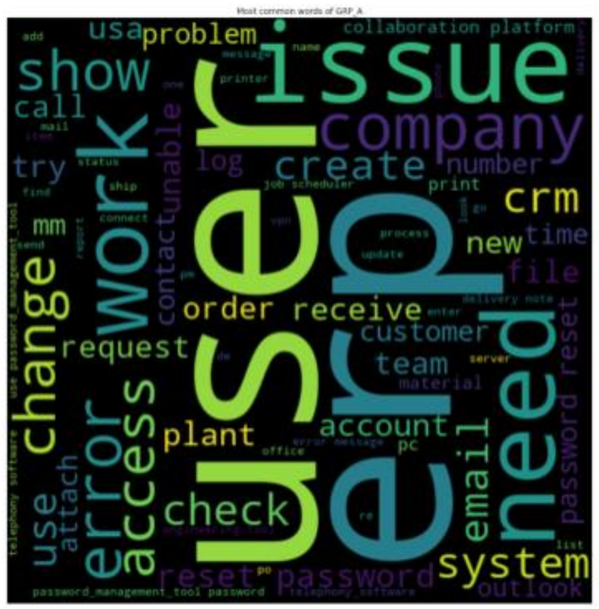
```
from wordcloud import STOPWORDS
from sklearn.feature_extraction.text import CountVectorizer
# Extend the English Stop Wordss
STOP_WORDS = STOPWORDS.union({'yes','na','hi', 'etc'
                              'receive','hello',
                              'regards','thanks',
                              'from','greeting',
                              'forward','reply',
                              'will','please',
                              'see','help','able'})

# Generic function to derive top N n-grams from the corpus
def get_top_n_ngrams(corpus, top_n=None, ngram_range=(1,1), stopwords=None):
    vec = CountVectorizer(ngram_range=ngram_range,
                          stop_words=stopwords).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq = sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:top_n]
```

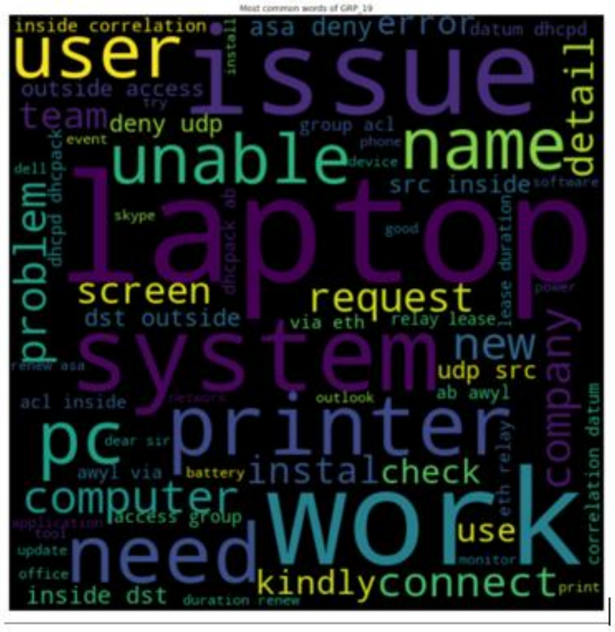
Analysis using Word Cloud

Word cloud is a collection, or cluster, of words depicted in different sizes. The bigger and bolder the word appears, the more often it's mentioned within a given text and the more important it is. Also known as tag clouds or text clouds, these are ideal ways to pull out the most pertinent parts of textual data, often also help business users compare and contrast two different pieces of text to find the wording similarities between the two.









Summary of the Text-Preprocessing:

It's indicative from the n-gram analysis and the word cloud is that the entire dataset speaks more about issues around

- password reset
- fail job & scheduler

Analysis of GRP_0 which is the most frequent group to assign a ticket to reveals that this group deals with mostly the maintenance problems such as password reset, account lock, login issue, ticket update, etc.

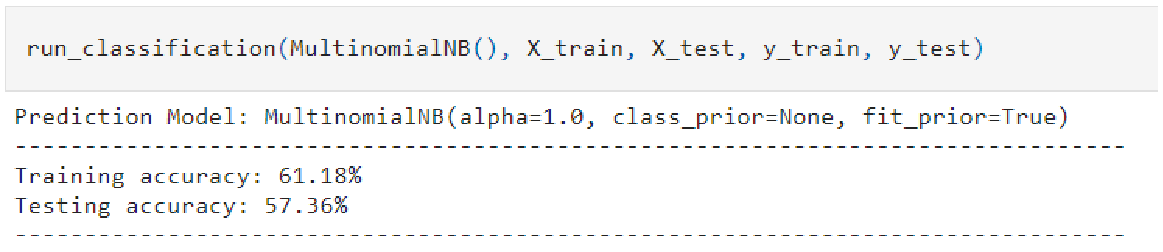
3. Deciding Models and Model Building

- Multinomial Naive Bayes
- K Nearest Neighbor (KNN)
- Support Vector Machine
- Decision Tree
- Random Forest
- Bidirectional LSTM

Model Performance (Part 1)

We tried to implement all these models to the data but the result we got very inaccurate. All the models are suffering from low test accuracy.

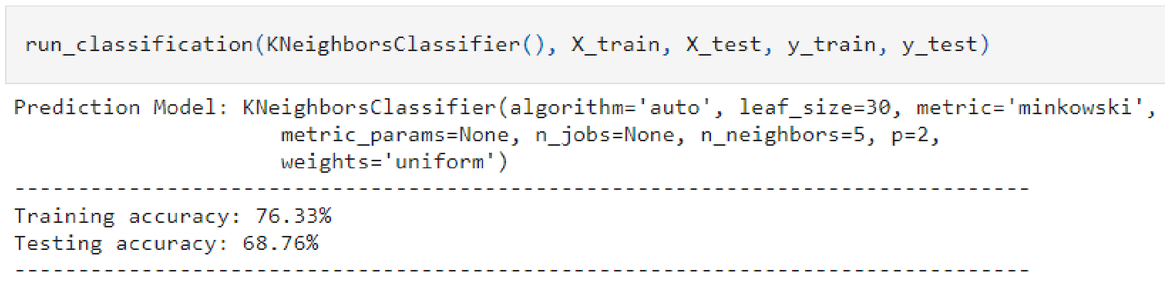
Multinomial Naive Bayes



```
run_classification(MultinomialNB(), X_train, X_test, y_train, y_test)
```

```
Prediction Model: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
--------------------------------------------------------------------------------
Training accuracy: 61.18%
Testing accuracy: 57.36%
--------------------------------------------------------------------------------
```

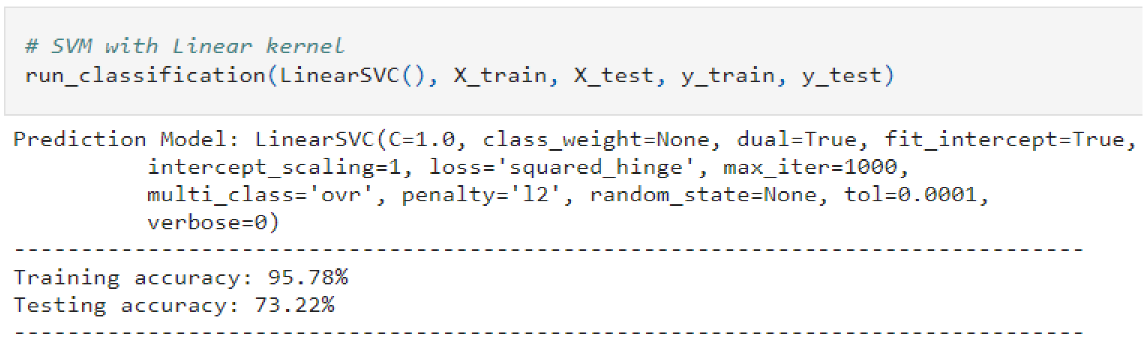
K Nearest Neighbor



```
run_classification(KNeighborsClassifier(), X_train, X_test, y_train, y_test)
```

```
Prediction Model: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
--------------------------------------------------------------------------------
Training accuracy: 76.33%
Testing accuracy: 68.76%
--------------------------------------------------------------------------------
```

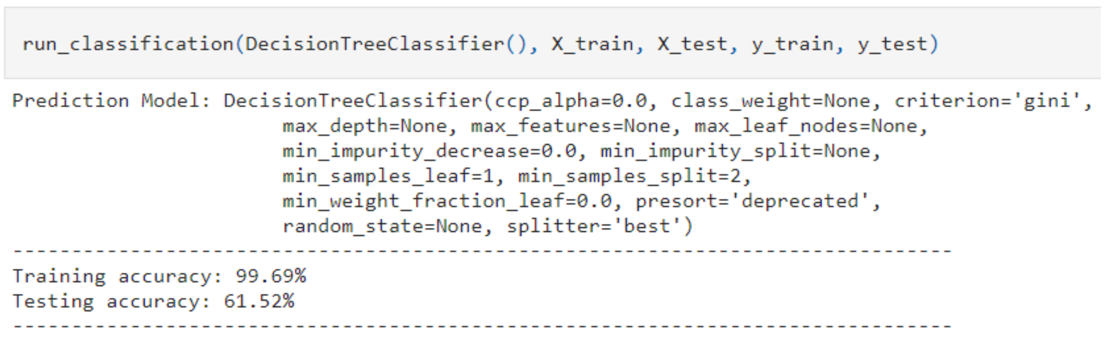
Support Vector Machine



```
# SVM with Linear kernel
run_classification(LinearSVC(), X_train, X_test, y_train, y_test)
```

```
Prediction Model: LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
--------------------------------------------------------------------------------
Training accuracy: 95.78%
Testing accuracy: 73.22%
--------------------------------------------------------------------------------
```

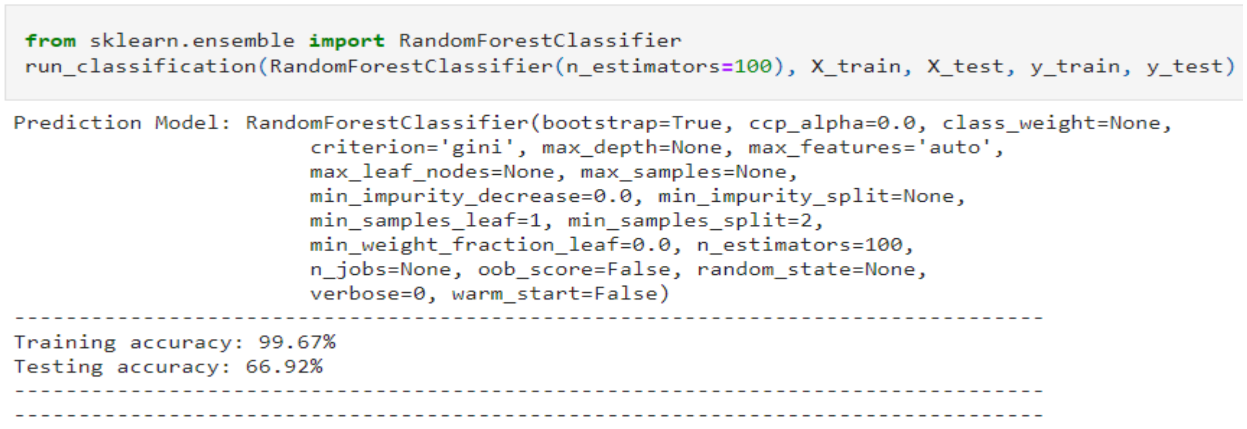
Decision Tree



```
run_classification(DecisionTreeClassifier(), X_train, X_test, y_train, y_test)
```

```
Prediction Model: DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
--------------------------------------------------------------------------------
Training accuracy: 99.69%
Testing accuracy: 61.52%
--------------------------------------------------------------------------------
```

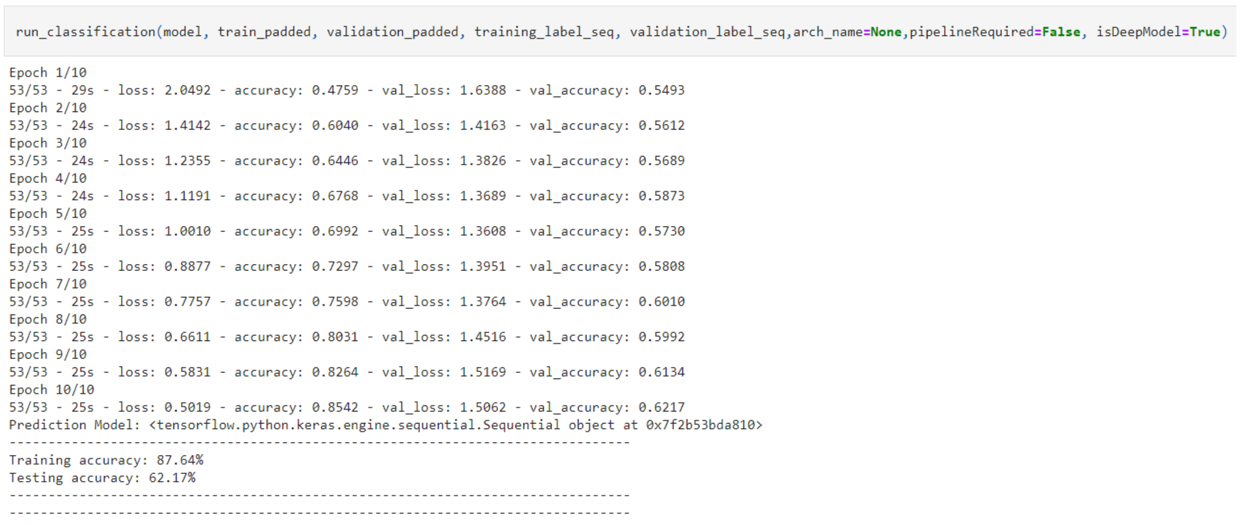
Random Forest



```
from sklearn.ensemble import RandomForestClassifier
run_classification(RandomForestClassifier(n_estimators=100), X_train, X_test, y_train, y_test)
```

```
Prediction Model: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
--------------------------------------------------------------------------------
Training accuracy: 99.67%
Testing accuracy: 66.92%
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

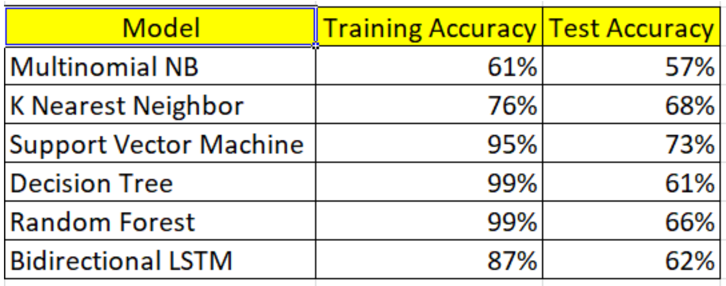
Bidirectional LSTM Classifier



```
run_classification(model, train_padded, validation_padded, training_label_seq, validation_label_seq,arch_name=None,pipelineRequired=False, isDeepModel=True)
```

```
Epoch 1/10
53/53 - 29s - loss: 2.0492 - accuracy: 0.4759 - val_loss: 1.6388 - val_accuracy: 0.5493
Epoch 2/10
53/53 - 24s - loss: 1.4142 - accuracy: 0.6040 - val_loss: 1.4163 - val_accuracy: 0.5612
Epoch 3/10
53/53 - 24s - loss: 1.2355 - accuracy: 0.6446 - val_loss: 1.3826 - val_accuracy: 0.5689
Epoch 4/10
53/53 - 24s - loss: 1.1191 - accuracy: 0.6768 - val_loss: 1.3689 - val_accuracy: 0.5873
Epoch 5/10
53/53 - 25s - loss: 1.0010 - accuracy: 0.6992 - val_loss: 1.3608 - val_accuracy: 0.5730
Epoch 6/10
53/53 - 25s - loss: 0.8877 - accuracy: 0.7297 - val_loss: 1.3951 - val_accuracy: 0.5808
Epoch 7/10
53/53 - 25s - loss: 0.7757 - accuracy: 0.7598 - val_loss: 1.3764 - val_accuracy: 0.6010
Epoch 8/10
53/53 - 25s - loss: 0.6611 - accuracy: 0.8031 - val_loss: 1.4516 - val_accuracy: 0.5992
Epoch 9/10
53/53 - 25s - loss: 0.5831 - accuracy: 0.8264 - val_loss: 1.5169 - val_accuracy: 0.6134
Epoch 10/10
53/53 - 25s - loss: 0.5019 - accuracy: 0.8542 - val_loss: 1.5062 - val_accuracy: 0.6217
Prediction Model: <tensorflow.python.keras.engine.sequential.Sequential object at 0x7f2b53bda810>
--------------------------------------------------------------------------------
Training accuracy: 87.64%
Testing accuracy: 62.17%
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

The summary of the results we got are attached below:

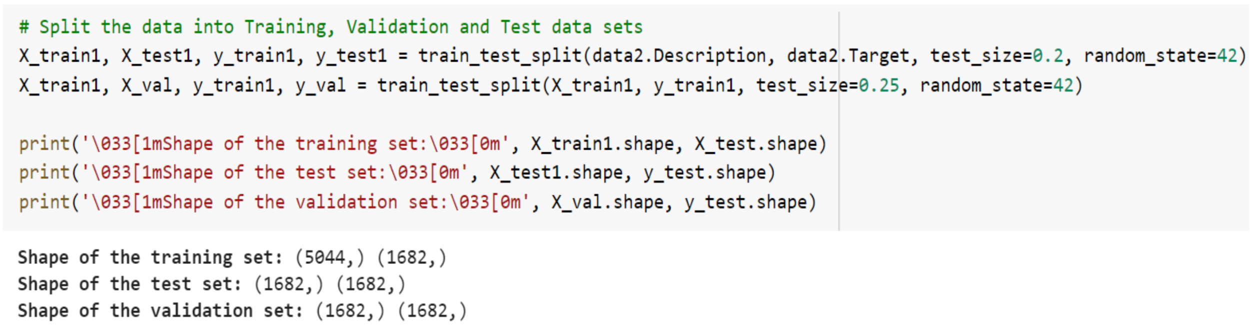


Model	Training Accuracy	Test Accuracy
Multinomial NB	61%	57%
K Nearest Neighbor	76%	68%
Support Vector Machine	95%	73%
Decision Tree	99%	61%
Random Forest	99%	66%
Bidirectional LSTM	87%	62%

Model Performance (Part 2)

In all the models we have tried, the accuracy of each of the Statistical models is over-fitted to a higher degree. One obvious reason is the dataset is highly imbalanced.
So to deal with the imbalanced dataset, we are going to consider two approaches.

Approach 1: Split the data into Training, Validation, and Test data sets

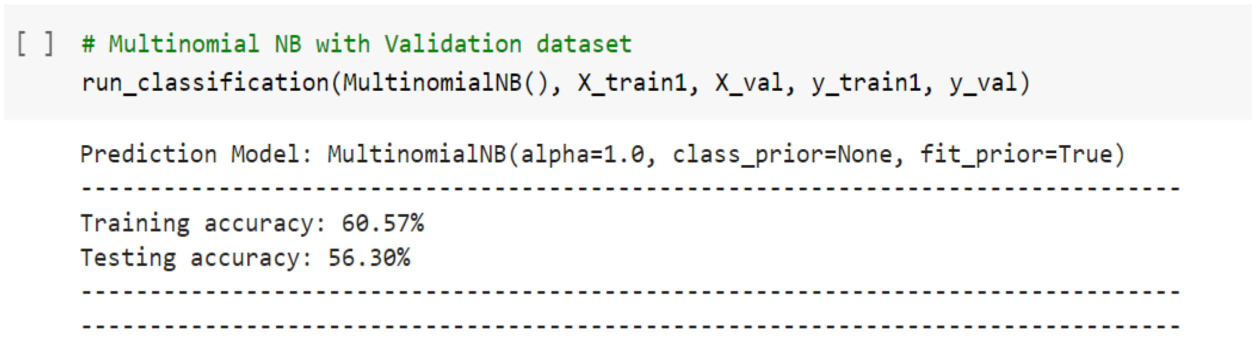


```
# Split the data into Training, Validation and Test data sets
X_train1, X_test1, y_train1, y_test1 = train_test_split(data2.Description, data2.Target, test_size=0.2, random_state=42)
X_train1, X_val, y_train1, y_val = train_test_split(X_train1, y_train1, test_size=0.25, random_state=42)

print('\033[1mShape of the training set:\033[0m', X_train1.shape, X_test.shape)
print('\033[1mShape of the test set:\033[0m', X_test1.shape, y_test.shape)
print('\033[1mShape of the validation set:\033[0m', X_val.shape, y_test.shape)
```

```
Shape of the training set: (5044,) (1682,)
Shape of the test set: (1682,) (1682,)
Shape of the validation set: (1682,) (1682,)
```

- Implemented the models on Approach 1:

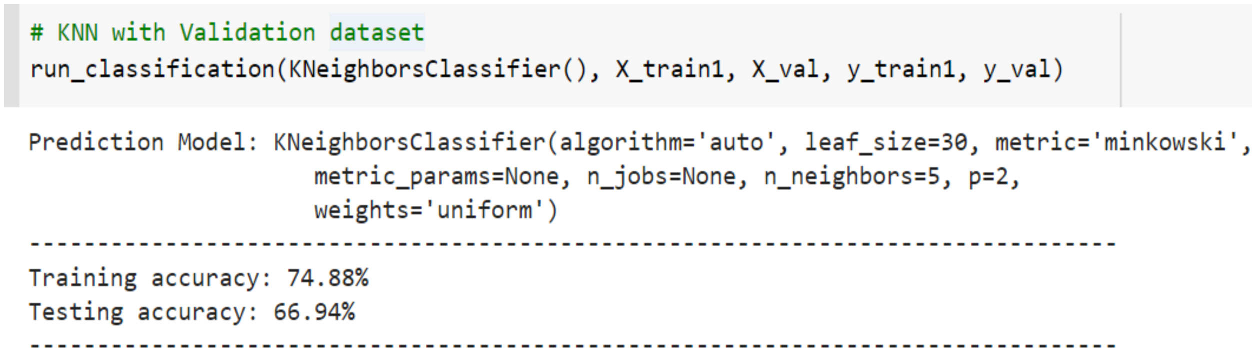
Multinomial Naive Bayes with the validation dataset



```
[ ]  # Multinomial NB with Validation dataset
     run_classification(MultinomialNB(), X_train1, X_val, y_train1, y_val)
```

```
Prediction Model: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
--------------------------------------------------------------------------------
Training accuracy: 60.57%
Testing accuracy: 56.30%
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

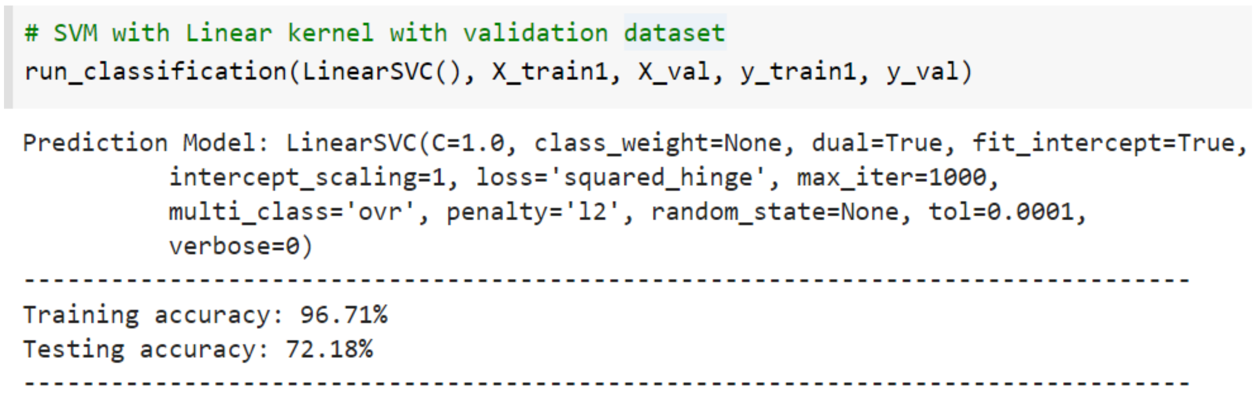
KNN with the validation dataset



```
# KNN with Validation dataset
run_classification(KNeighborsClassifier(), X_train1, X_val, y_train1, y_val)
```

```
Prediction Model: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
--------------------------------------------------------------------------------
Training accuracy: 74.88%
Testing accuracy: 66.94%
--------------------------------------------------------------------------------
```

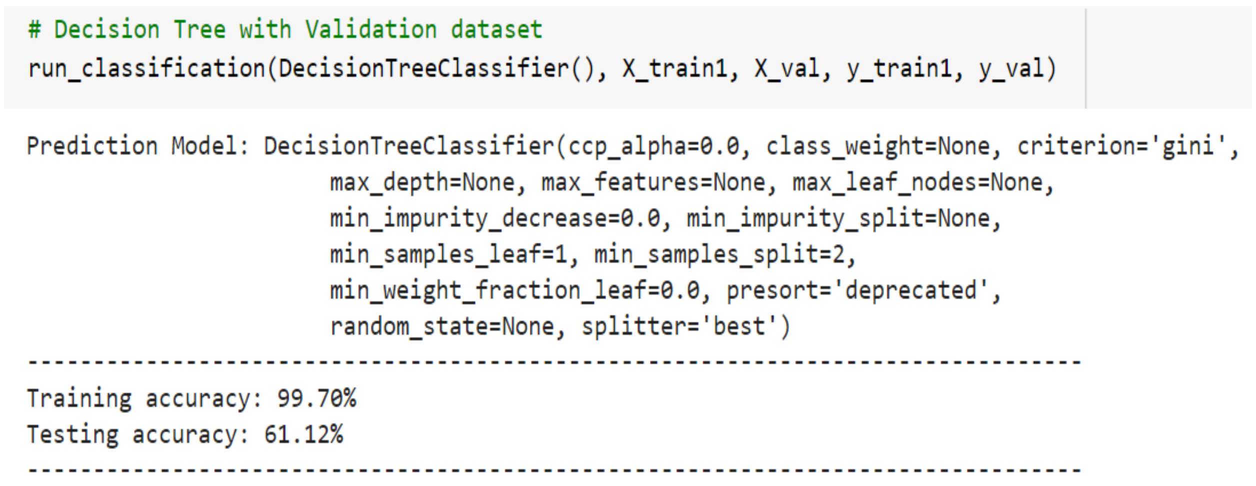
Support Vector Machine



```
# SVM with Linear kernel with validation dataset
run_classification(LinearSVC(), X_train1, X_val, y_train1, y_val)
```

```
Prediction Model: LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
--------------------------------------------------------------------------------
Training accuracy: 96.71%
Testing accuracy: 72.18%
--------------------------------------------------------------------------------
```

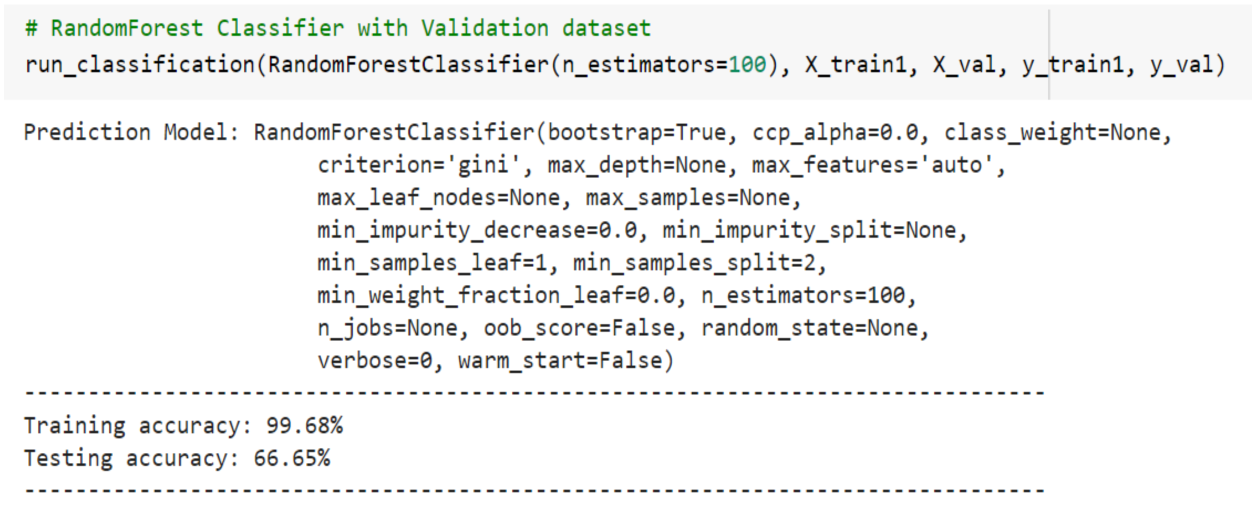
Decision Tree



```
# Decision Tree with Validation dataset
run_classification(DecisionTreeClassifier(), X_train1, X_val, y_train1, y_val)
```

```
Prediction Model: DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
--------------------------------------------------------------------------------
Training accuracy: 99.70%
Testing accuracy: 61.12%
--------------------------------------------------------------------------------
```

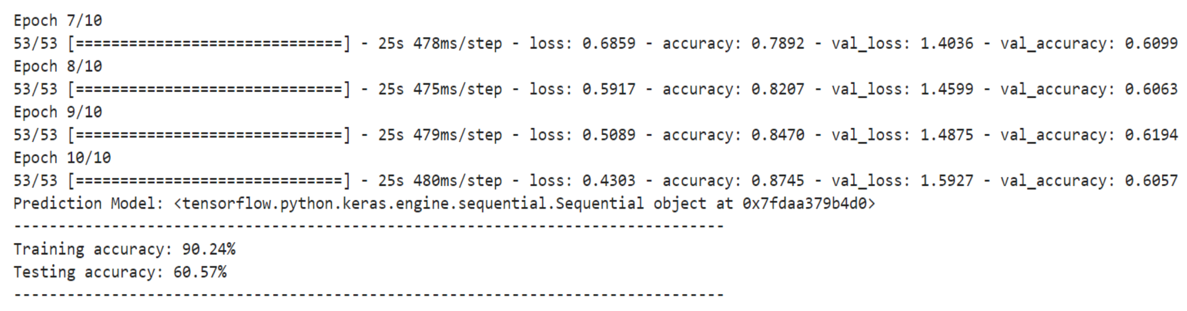
Random Forest



```
# RandomForest Classifier with Validation dataset
run_classification(RandomForestClassifier(n_estimators=100), X_train1, X_val, y_train1, y_val)
```

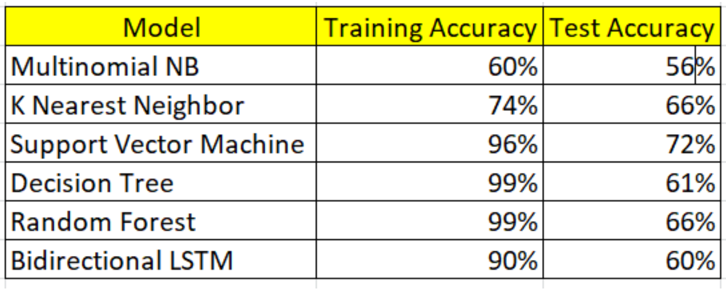
```
Prediction Model: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
--------------------------------------------------------------------------------
Training accuracy: 99.68%
Testing accuracy: 66.65%
--------------------------------------------------------------------------------
```

Bidirectional LSTM



```
Epoch 7/10
53/53 [==============================] - 25s 478ms/step - loss: 0.6859 - accuracy: 0.7892 - val_loss: 1.4036 - val_accuracy: 0.6099
Epoch 8/10
53/53 [==============================] - 25s 475ms/step - loss: 0.5917 - accuracy: 0.8207 - val_loss: 1.4599 - val_accuracy: 0.6063
Epoch 9/10
53/53 [==============================] - 25s 479ms/step - loss: 0.5089 - accuracy: 0.8470 - val_loss: 1.4875 - val_accuracy: 0.6194
Epoch 10/10
53/53 [==============================] - 25s 480ms/step - loss: 0.4303 - accuracy: 0.8745 - val_loss: 1.5927 - val_accuracy: 0.6057
Prediction Model: <tensorflow.python.keras.engine.sequential.Sequential object at 0x7fdaa379b4d0>
--------------------------------------------------------------------------------
Training accuracy: 90.24%
Testing accuracy: 60.57%
--------------------------------------------------------------------------------
```

Summary of the results:

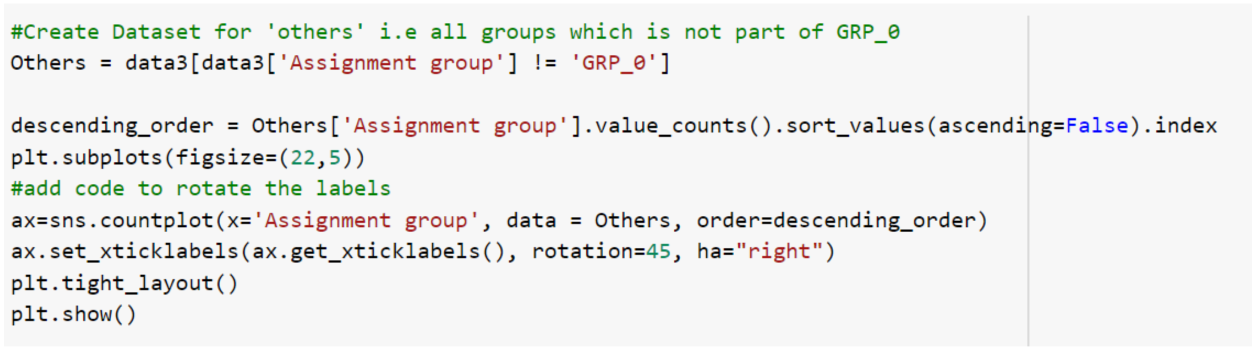


Model	Training Accuracy	Test Accuracy
Multinomial NB	60%	56%
K Nearest Neighbor	74%	66%
Support Vector Machine	96%	72%
Decision Tree	99%	61%
Random Forest	99%	66%
Bidirectional LSTM	90%	60%

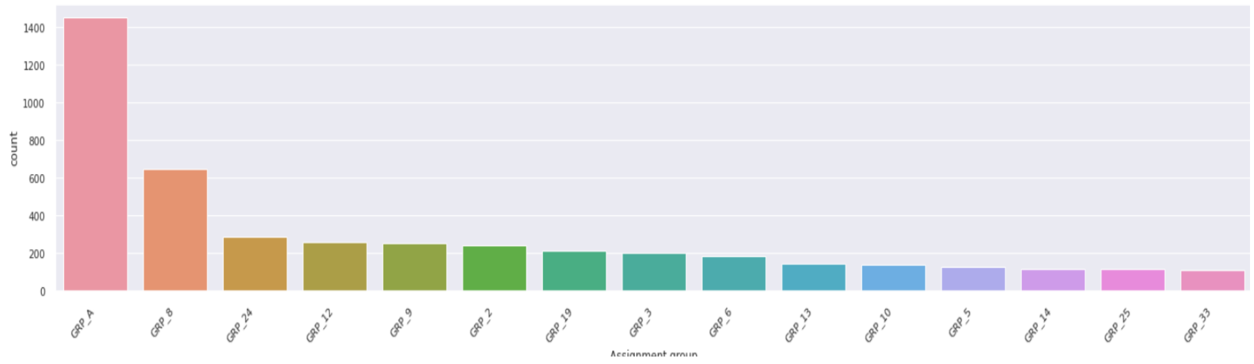
Approach 2: Resampling Technique (Upsampling)/Split the data into Training, Validation and Test data sets

Steps included in this process:

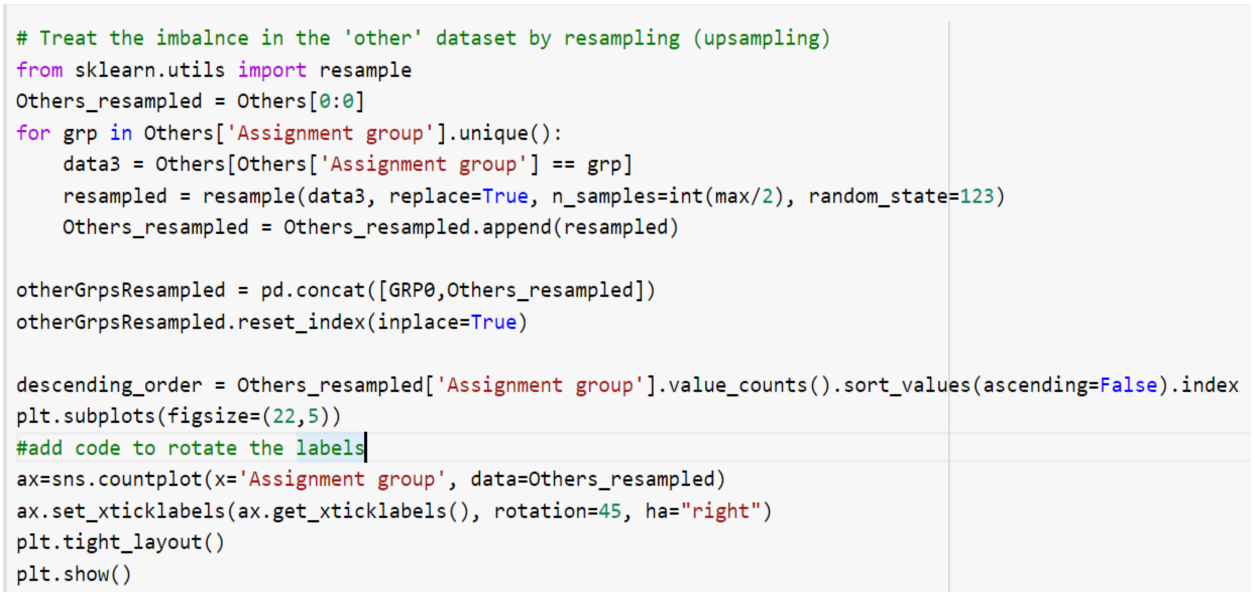
- Creating a dataset for the groups other than GRP_0



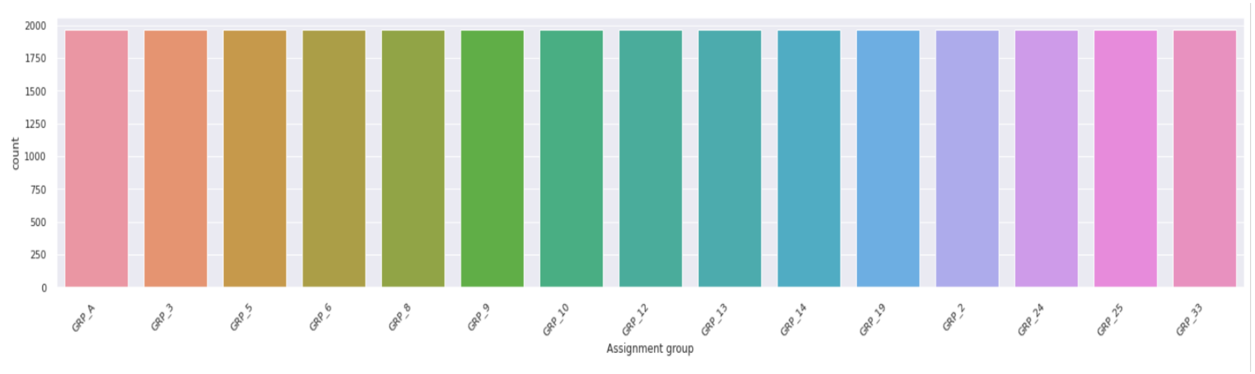
```
#Create Dataset for 'others' i.e all groups which is not part of GRP_0
Others = data3[data3['Assignment group'] != 'GRP_0']

descending_order = Others['Assignment group'].value_counts().sort_values(ascending=False).index
plt.subplots(figsize=(22,5))
#add code to rotate the labels
ax=sns.countplot(x='Assignment group', data = Others, order=descending_order)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha="right")
plt.tight_layout()
plt.show()
```



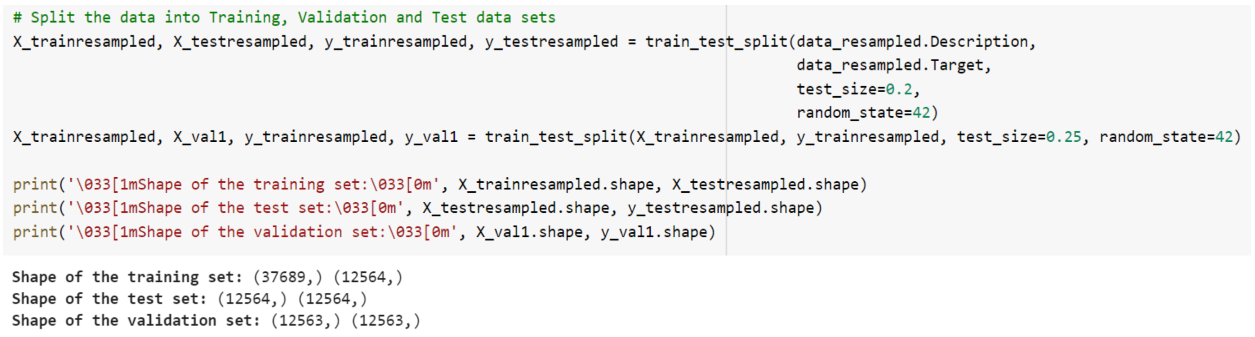
- Now applying resampling technique (upsampling) on the groups other than GRP_0



```
# Treat the imbalnce in the 'other' dataset by resampling (upsampling)
from sklearn.utils import resample
Others_resampled = Others[0:0]
for grp in Others['Assignment group'].unique():
    data3 = Others[Others['Assignment group'] == grp]
    resampled = resample(data3, replace=True, n_samples=int(max/2), random_state=123)
    Others_resampled = Others_resampled.append(resampled)

otherGrpsResampled = pd.concat([GRP0,Others_resampled])
otherGrpsResampled.reset_index(inplace=True)

descending_order = Others_resampled['Assignment group'].value_counts().sort_values(ascending=False).index
plt.subplots(figsize=(22,5))
#add code to rotate the labels
ax=sns.countplot(x='Assignment group', data=Others_resampled)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha="right")
plt.tight_layout()
plt.show()
```



- As we can see that, groups other than GRP_0 are resampled and it's evident from the above histogram as well.
- Split the data into Training, Validation, and Test Data

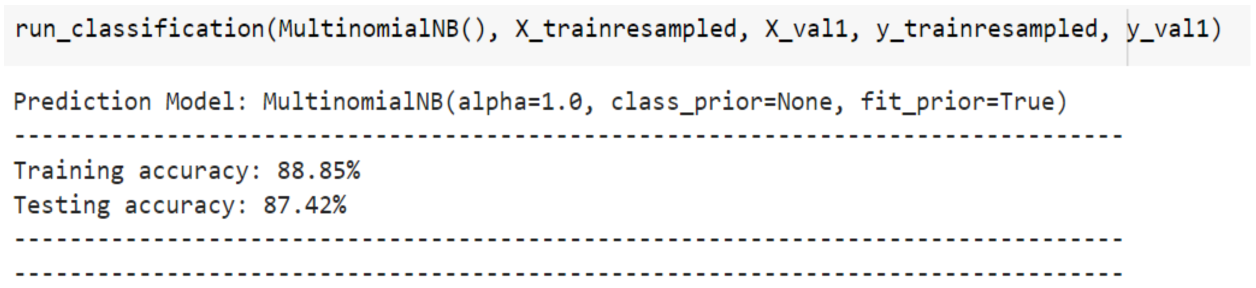


```
# Split the data into Training, Validation and Test data sets
X_trainresampled, X_testresampled, y_trainresampled, y_testresampled = train_test_split(data_resampled.Description,
                                                                                         data_resampled.Target,
                                                                                         test_size=0.2,
                                                                                         random_state=42)
X_trainresampled, X_val1, y_trainresampled, y_val1 = train_test_split(X_trainresampled, y_trainresampled, test_size=0.25, random_state=42)

print('\033[1mShape of the training set:\033[0m', X_trainresampled.shape, X_testresampled.shape)
print('\033[1mShape of the test set:\033[0m', X_testresampled.shape, y_testresampled.shape)
print('\033[1mShape of the validation set:\033[0m', X_val1.shape, y_val1.shape)
```

```
Shape of the training set: (37689,) (12564,)
Shape of the test set: (12564,) (12564,)
Shape of the validation set: (12563,) (12563,)
```

- Implementing models on the above dataset:

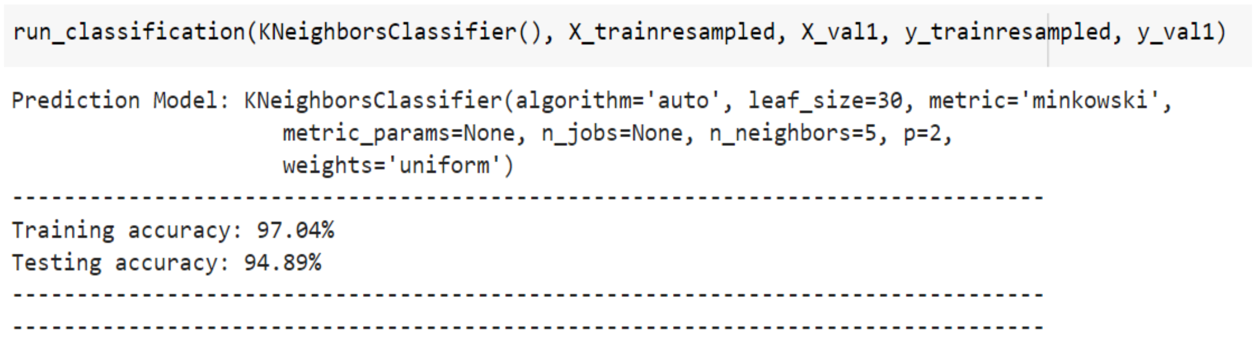
Multinomial Naive Bayes



```
run_classification(MultinomialNB(), X_trainresampled, X_val1, y_trainresampled, y_val1)
```

```
Prediction Model: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
--------------------------------------------------------------------------------
Training accuracy: 88.85%
Testing accuracy: 87.42%
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

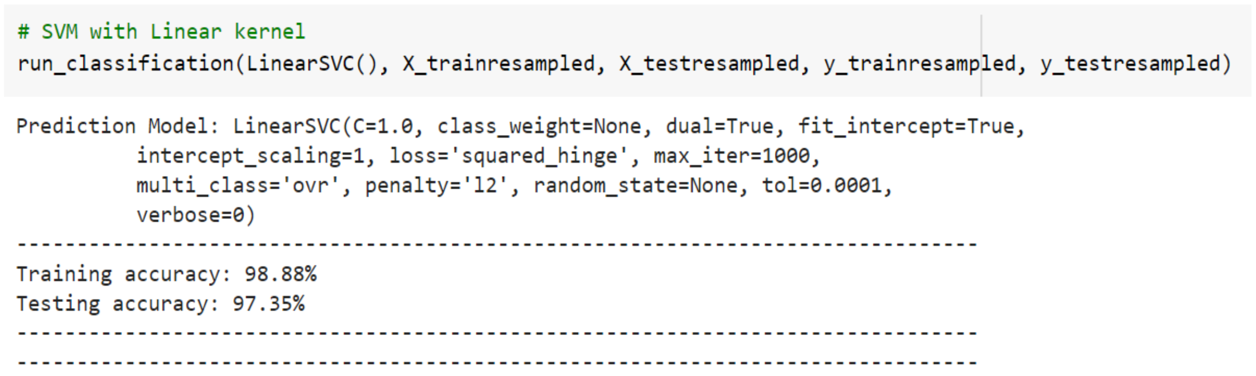
KNN



```
run_classification(KNeighborsClassifier(), X_trainresampled, X_val1, y_trainresampled, y_val1)
```

```
Prediction Model: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
--------------------------------------------------------------------------------
Training accuracy: 97.04%
Testing accuracy: 94.89%
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

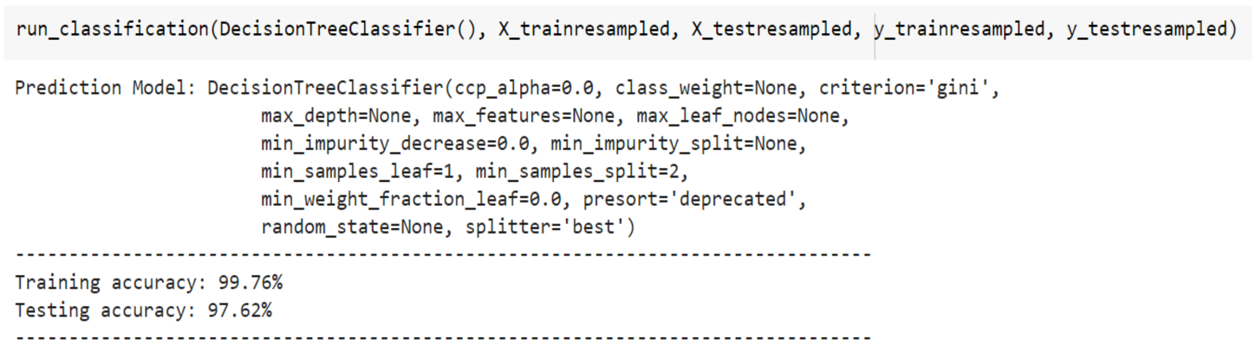
Support Vector Machine



```
# SVM with Linear kernel
run_classification(LinearSVC(), X_trainresampled, X_testresampled, y_trainresampled, y_testresampled)
```

```
Prediction Model: LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
--------------------------------------------------------------------------------
Training accuracy: 98.88%
Testing accuracy: 97.35%
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

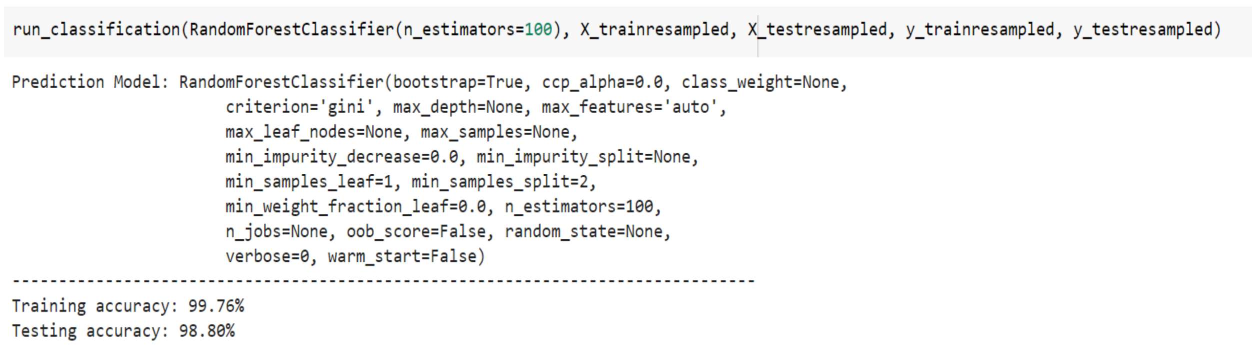
Decision Tree



```
run_classification(DecisionTreeClassifier(), X_trainresampled, X_testresampled, y_trainresampled, y_testresampled)
```

```
Prediction Model: DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
--------------------------------------------------------------------------------
Training accuracy: 99.76%
Testing accuracy: 97.62%
--------------------------------------------------------------------------------
```

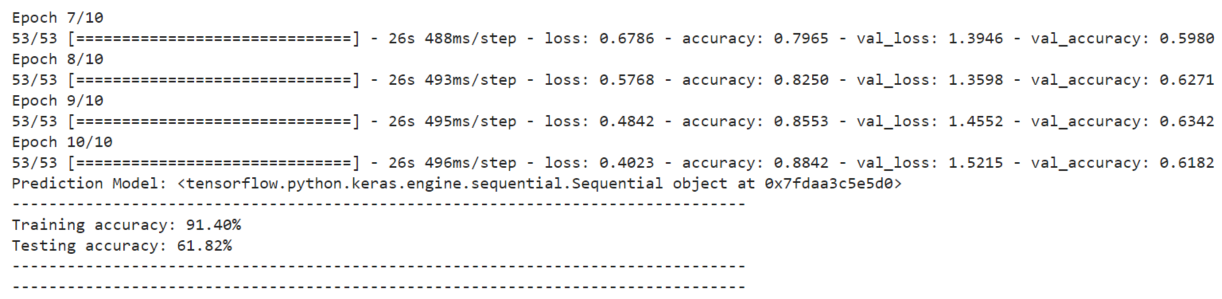
Random Forest



```
run_classification(RandomForestClassifier(n_estimators=100), X_trainresampled, X_testresampled, y_trainresampled, y_testresampled)
```

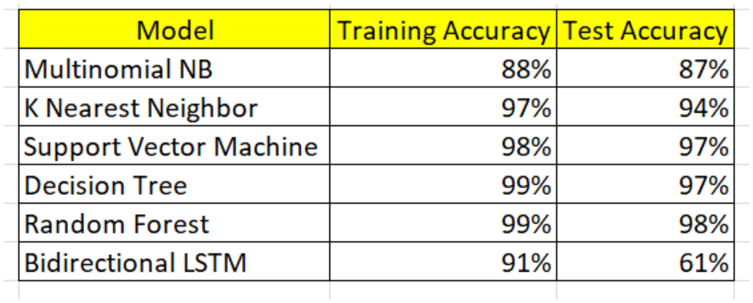
```
Prediction Model: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
--------------------------------------------------------------------------------
Training accuracy: 99.76%
Testing accuracy: 98.80%
```

Bidirectional LSTM



```
Epoch 7/10
53/53 [==============================] - 26s 488ms/step - loss: 0.6786 - accuracy: 0.7965 - val_loss: 1.3946 - val_accuracy: 0.5980
Epoch 8/10
53/53 [==============================] - 26s 493ms/step - loss: 0.5768 - accuracy: 0.8250 - val_loss: 1.3598 - val_accuracy: 0.6271
Epoch 9/10
53/53 [==============================] - 26s 495ms/step - loss: 0.4842 - accuracy: 0.8553 - val_loss: 1.4552 - val_accuracy: 0.6342
Epoch 10/10
53/53 [==============================] - 26s 496ms/step - loss: 0.4023 - accuracy: 0.8842 - val_loss: 1.5215 - val_accuracy: 0.6182
Prediction Model: <tensorflow.python.keras.engine.sequential.Sequential object at 0x7fdaa3c5e5d0>
--------------------------------------------------------------------------------
Training accuracy: 91.40%
Testing accuracy: 61.82%
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Summary of the results:



Model	Training Accuracy	Test Accuracy
Multinomial NB	88%	87%
K Nearest Neighbor	97%	94%
Support Vector Machine	98%	97%
Decision Tree	99%	97%
Random Forest	99%	98%
Bidirectional LSTM	91%	61%

4. How to improve your model performance?

Till now we saw that after cleaning our data, we implemented few models on our dataset which gave us not-so-satisfactory results. So we initiated two approaches for getting a better accuracy which consisted of implementing models on Training, validation, and test datasets. This approach also did not give us the expected results.

Going forward we considered another approach where we were able to upsample our dataset and implementing models on the particular dataset was our next step.

The above approach gave us the results which indicated that our models are biased through the accuracy was exceptionally good.

This gave us important insight and forced us to reconsider a different approach towards handling our dataset for Milestone 2.

Future Approach

1. We will try to reduce the number of classes from 16 to optimal number using K-means clustering (Elbow Method)
2. We will try to translate non-English words into English.
3. We will try to handle our imbalanced dataset using upsampling and downsampling.
4. We will use SMOTE technique for handling an imbalanced dataset
5. We will implement 2 deep learning models as well.