```
import pandas as pd

# Data Preprocessing Util
import re
import nltk
import string
import numpy as np
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

# load nltk's SnowballStemmer as variabled 'stemmer'
from nltk.stem.snowball import SnowballStemmer
from nltk.stem.porter import PorterStemmer

# Import dependencies for Clsutering and PCA
from gensim.models import Word2Vec
from sklearn.cluster import KMeans
from sklearn import cluster
from sklearn import metrics
from sklearn.decomposition import PCA
from scipy.cluster import hierarchy
from sklearn.cluster import AgglomerativeClustering

# For Rake used in filling missing values
from rake_nltk import Rake

# For Reducing longer text values to limited so that maxlen can be limited for the corpus
from gensim.summarization.summarizer import summarize

# Dependencies for Upsampling
from imblearn.over_sampling import RandomOverSampler

def removeString(data, regex):
   return data.str.lower().str.replace(regex.lower(), ' ')


def getRegexList():
   """
    Adding regex list as per the given data set to flush off the unnecessary text
   :return:
   """
   regex_list = []
   regex_list += ['From:(.*)\r\n']  # from line
   regex_list += ['Sent:(.*)\r\n']  # sent to line
   regex_list += ['received from:(.*)\r\n']  # received data line
   regex_list += ['received']  # received data line
   regex_list += ['To:(.*)\r\n']  # to line
   regex_list += ['CC:(.*)\r\n']  # cc line
```

```
    regex_list += ['https?:[^\]\n\r]+']  # https & http
    regex_list += ['[\w\d\-\_\.]+@[\w\d\-\_\.]+']  # emails are not required
    regex_list += ['[0-9][\-0–90-9 ]+']  # phones are not required
    regex_list += ['[0-9]']  # numbers not needed
    regex_list += ['[^a-zA-z 0-9]+']  # anything that is not a letter
    regex_list += ['[\r\n]']  # \r\n
    regex_list += [' [a-zA-Z] ']  # single letters makes no sense
    regex_list += [' [a-zA-Z][a-zA-Z] ']  # two-letter words makes no sense
    regex_list += ["  "]  # double spaces

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

    return regex_list


def cleanDataset(dataset, column, regex_list):
    for regex in regex_list:
        dataset[column] = removeString(dataset[column], regex)
    return dataset


def clean_text(text):
    # Lowercase all characters in input 'text'
    text = text.lower()

    # Removes any Website URL from text
    pattern = re.compile('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-
F]))+')
    text = pattern.sub('', text)

    #Extract 'firstname' from email id : firstname@email.xyz
    text = " ".join(filter(lambda x:x[0]!='@', text.split()))

    # Remove Emojis
    emoji = re.compile("["
                       u"\U0001F600-\U0001FFFF"  # emoticons
                       u"\U0001F300-\U0001F5FF"  # symbols&  pictographs
                       u"\U0001F680-\U0001F6FF"  # transport & map symbols
                       u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                       u"\U00002702-\U000027B0"
                       u"\U000024C2-\U0001F251"
                       "]+", flags=re.UNICODE)
    text = emoji.sub(r'', text)
    # 6th June Edit Stemming for Login suggestions by Charan/Vaishakh
    '''text = re.sub(r"log in", "login", text)
    text = re.sub(r"log on", "login", text)
    text = re.sub(r"logon", "login", text)
    text = re.sub(r"logged", "login", text)
    text = re.sub(r"logging", "login", text)'''
    text = re.sub(r"tologin", "login", text)

    # 6th June Edit Lemmitization suggestions by Charan/Vaishakh
    text = re.sub(r"Job_", "job ", text)
    text = re.sub(r"mm_", "mm ", text)
    text = re.sub(r"time cards", "timecards", text)
    text = re.sub(r"engg", "engineer", text)
    text = re.sub(r"nx 9", "nx9", text)
    text = re.sub(r"installing", "install", text)
    text = re.sub(r"installation", "install", text)
    text = re.sub(r"us time", "US time", text)

    # A list of Contractions from http://stackoverflow.com/questions/19790188/expanding-english-
language-contractions-in-python
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"ain't", "am not", text)
    text = re.sub(r"aren't", "are not", text)
    text = re.sub(r"can't", "can not", text)
    text = re.sub(r"can't've", "cannot have", text)
    text = re.sub(r"'cause", "because", text)
    text = re.sub(r"could've", "could have", text)
    text = re.sub(r"couldn't", "could not", text)
    text = re.sub(r"couldn't", "could not",  text)
    text = re.sub(r"couldn't've", "could not have", text)
    text = re.sub(r"didn't", "did not", text)
    text = re.sub(r"doesn't", "does not", text)
    text = re.sub(r"don't", "do not", text)
    text = re.sub(r"hadn't", "had not", text)
    text = re.sub(r"hadn't've", "had not have", text)
    text = re.sub(r"hasn't", "has not", text)
    text = re.sub(r"have't", "have not", text)
    text = re.sub(r"haven't", "have not", text)
    text = re.sub(r"he'd", "he would", text)
    text = re.sub(r"he'd've", "he would have", text)
    text = re.sub(r"he'll", "he will", text)
    text = re.sub(r"he's", "he is", text)
    text = re.sub(r"how'd", "how did", text)
    text = re.sub(r"how'll", "how will", text)
    text = re.sub(r"how's", "how is", text)
    text = re.sub(r"i'd", "i would", text)
    text = re.sub(r"i'll", "i will", text)
    text = re.sub(r"i'm", "i am", text)
    text = re.sub(r"i've", "i have", text)
    text = re.sub(r"isn't", "is not", text)
    text = re.sub(r"it'd", "it would", text)
    text = re.sub(r"it'll", "it will", text)
    text = re.sub(r"it's", "it is", text)
    text = re.sub(r"let's", "let us", text)
    text = re.sub(r"ma'am", "madam", text)
    text = re.sub(r"mayn't", "may not", text)
    text = re.sub(r"might've", "might have", text)
    text = re.sub(r"mightn't", "might not", text)
    text = re.sub(r"must've", "must have", text)
    text = re.sub(r"mustn't", "must not", text)
    text = re.sub(r"needn't", "need not", text)
    text = re.sub(r"oughtn't", "ought not", text)
    text = re.sub(r"shan't", "shall not", text)
    text = re.sub(r"sha'n't", "shall not", text)
    text = re.sub(r"she'd", "she would", text)
    text = re.sub(r"she'll", "she will", text)
    text = re.sub(r"she's", "she is", text)
    text = re.sub(r"should've", "should have", text)
    text = re.sub(r"shouldn't", "should not", text)
    text = re.sub(r"that'd", "that would", text)
    text = re.sub(r"that's", "that is", text)
    text = re.sub(r"there'd", "there had", text)
    text = re.sub(r"there's", "there is", text)
    text = re.sub(r"they'd", "they would", text)
    text = re.sub(r"they'll", "they will", text)
    text = re.sub(r"they're", "they are", text)
    text = re.sub(r"they've", "they have", text)
    text = re.sub(r"wasn't", "was not", text)
    text = re.sub(r"we'd", "we would", text)
    text = re.sub(r"we'll", "we will", text)
    text = re.sub(r"we're", "we are", text)
    text = re.sub(r"we've", "we have", text)
    text = re.sub(r"weren't", "were not", text)
    text = re.sub(r"what'll", "what will", text)
    text = re.sub(r"what're", "what are", text)
    text = re.sub(r"what's", "what is", text)
    text = re.sub(r"what've", "what have", text)
    text = re.sub(r"where'd", "where did", text)
    text = re.sub(r"where's", "where is", text)
    text = re.sub(r"who'll", "who will", text)
    text = re.sub(r"who's", "who is", text)
    text = re.sub(r"won't", "will not", text)
```

```
    text = re.sub(r"wouldn't", "would not", text)
    text = re.sub(r"you'd", "you would", text)
    text = re.sub(r"you'll", "you will", text)
    text = re.sub(r"you're", "you are", text)

    # Getting Rid of Punctuations
    text = re.sub(r"[,.\"\'!@#$%^&*(){}?/;`~:<>+=-]", "", text)

    # Remove Emails
    text = re.sub(r"\S*@\S*\s?", '', text)

    # Remove new line characters
    text = re.sub('\s+', ' ', text)

    # Remove distracting single quotes
    text = re.sub("\'", "", text)

    # Tokenizing and Stemming with  PorterStemmer
    # stemmer = PorterStemmer()
    filtered_tokens = []

    stemmer = SnowballStemmer("english")
    tokens = word_tokenize(text)
    for token in tokens:
        if re.search('[a-zA-Z]', token):
            filtered_tokens.append(token)
    stemmed_tokens = [stemmer.stem(token) for token in filtered_tokens]
    #Rejoining the Stemmed Tokens back again to form new text
    text = ' '.join(stemmed_tokens)



    # Final Lowercasing text after conversion
    text = text.lower()
    return text

# Function to remove Stopwords
def remove_stopwords(df):
    """ Removes stopwords based on a known set of stopwords
    available in the nltk package. In addition, we include our
    made up word in here.
    """
    # Luckily nltk already has a set of stopwords that we can remove from the texts.
    stopwords = nltk.corpus.stopwords.words('english')
    # we'll add our own special word in here 'qwerty'
    stopwords.append(our_special_word)
    stopwords.remove
    df['stopwords_removed'] = list(map(lambda doc:
                                 [word for word in doc if word not in stopwords],
                                 df['tokenized_text']))



###### Function for Implementing Rake
# Define Rake Model
r = Rake()

def rake_implement(x) :
    r.extract_keywords_from_text(x)
    return r.get_ranked_phrases()


def create_summarized_feature(x):
    str_local = ""
    try :
        if len(x.split()) > 200:
            str_local = summarize(x, word_count = 200)
        else:
            str_local = x

    except ValueError:
        str_local_Error = ". ".join(rake_implement(x))
        str_local = summarize(str_local_Error, word_count = 200)
        print("Can't Summarize this sentence as input has only one sentence. Hence, replacing with
(Rake + Summarized Value)" )
    return str_local


def check_label_split(train_y, test_y, label_encoded_dict):
    # np.setdiff1d(list_2,list_1) yields the elements in 'list_2' that are NOT in 'list_1'

    missing_test_labels = np.setdiff1d(train_y.unique(), test_y.unique())
    missing_train_labels = np.setdiff1d(test_y.unique(), train_y.unique())


    print("The following Target Labels are missing from Test Data : \n")
    for value in missing_test_labels:
        for key, val in label_encoded_dict.items():
         if val == value:
             print(key)

    print("\nThe following Target Labels are missing from Train Data :")
    for value in missing_train_labels:
        for key, val in label_encoded_dict.items():
         if val == value:
            print(key)

def replaceEmailIds(dfColumn):
    newDF = pd.DataFrame()
    newDF['datacolumn'] = dfColumn
    for i, row in newDF.iterrows():
        #print(i)
        if pd.notna(newDF.at[i,'datacolumn']):
            if not re.findall('^[0-9]*$',str(newDF.at[i,'datacolumn'])):
                lstEmails = re.findall('\S+@\S+', newDF.at[i,'datacolumn'])
                #print(lstEmails)
                if lstEmails:
                    for email in lstEmails:
                        newDF['datacolumn'][i] = newDF['datacolumn'][i].replace(email, "emailaddress")
                        #print(newDF['datacolumn'][i])
    return newDF['datacolumn']

def applyDetRules(datadf,rulesdf,Description,ShortDescription):
    datadf['pred_group'] = np.nan
    for i, row in rulesdf.iterrows():
         #hardcoding GRP25
        for j, row in datadf.iterrows():
            if pd.notna(datadf[ShortDescription][j]):
                if (('erp' in datadf[ShortDescription][j]) and (('EU_tool' in datadf[ShortDescription][j]))):
                        datadf['pred_group'][j] = 'GRP_25'

        #Hardcoding GRP17
        for j, row in datadf.iterrows():
            if pd.notna(datadf[Description][j]):
                if (datadf[Description][j] == 'the'):
                        datadf['pred_group'][j] = 'GRP_17'
                #Hardcoding GRP55
                if (('finance_app' in datadf[ShortDescription][j]) and ('HostName_1132' not in
datadf[ShortDescription][j])):
                    datadf['pred_group'][j] = 'GRP_55'
                #Hardcoding GRP58
                if (('processor' in datadf[Description][j]) and ('engg' in datadf[Description][j])):
                    datadf['pred_group'][j] = 'GRP_58'

        if rulesdf['Short Desc Rule'][i] == 'begins with' and rulesdf['Desc Rule'][i] == 'begins with' and
pd.isna(rulesdf['User'][i]):
            for j, row in datadf.iterrows():
                if pd.notna(datadf[ShortDescription][j]) and pd.notna(datadf[Description][j]):
                    if ((datadf[ShortDescription][j].startswith(rulesdf['Short Dec Keyword'][i])) and
(datadf[Description][j].startswith(rulesdf['Dec keyword'][i]))):
                        datadf['pred_group'][j] = rulesdf['Group'][i]
        if pd.isna(rulesdf['Short Desc Rule'][i]) and rulesdf['Desc Rule'][i] == 'begins with' and
pd.notna(rulesdf['User'][i]):
```

```
            for j, row in datadf.iterrows():
                if pd.notna(datadf[Description][j]) and pd.notna(datadf['Caller'][j]):
                    if ((datadf[Description][j].startswith(rulesdf['Desc Rule'][i]) and (rulesdf['User'][i] ==
datadf['Caller'][j]))):
                        datadf['pred_group'][j] = rulesdf['Group'][i]
        if rulesdf['Short Desc Rule'][i] == 'contains' and pd.notna(rulesdf['User'][i]):
            for j, row in datadf.iterrows():
                if (pd.notna(datadf[ShortDescription][j]) and pd.notna(datadf['Caller'][j])):
                    if ((rulesdf['Short Dec Keyword'][i] in datadf[ShortDescription][j]) and (rulesdf['User'][i]
== datadf['Caller'][j])):
                        datadf['pred_group'][j] = rulesdf['Group'][i]
        if rulesdf['Short Desc Rule'][i] == 'contains' and pd.isna(rulesdf['Desc Rule'][i]) and
pd.isna(rulesdf['User'][i]):
            for j, row in datadf.iterrows():
                #print(j)
                if pd.notna(datadf[ShortDescription][j]):
                    if (rulesdf['Short Dec Keyword'][i] in datadf[ShortDescription][j]):
                        datadf['pred_group'][j] = rulesdf['Group'][i]
        if pd.isna(rulesdf['Short Desc Rule'][i]) and rulesdf['Desc Rule'][i] == 'begins with' and
pd.isna(rulesdf['User'][i]):
            for j, row in datadf.iterrows():
                if pd.notna(datadf[Description][j]):
                    if (datadf[Description][j].startswith(rulesdf['Dec keyword'][i])):
                        datadf['pred_group'][j] = rulesdf['Group'][i]
        if pd.isna(rulesdf['Short Desc Rule'][i]) and rulesdf['Desc Rule'][i] == 'contains' and
pd.isna(rulesdf['User'][i]):
            for j, row in datadf.iterrows():
                if pd.notna(datadf[Description][j]):
                    if (rulesdf['Dec keyword'][i] in datadf[Description][j]):
                        datadf['pred_group'][j] = rulesdf['Group'][i]


    return datadf



def divide_group_0(translated_df):
    #Create an index column
    translated_df["index"]=translated_df.index.values

    translated_df_Grp0 = translated_df[["Assignment
group","Translated_Description","Translated_Short
description","index"]].where(translated_df["Assignment group"]=="GRP_0")

    translated_df_Grp0.dropna(how="all")
    translated_df_Grp0 = translated_df_Grp0.dropna()
    translated_df_Grp0["index"] = translated_df_Grp0["index"].astype(int)
    #translated_df_Grp0.tail(5)
```

```
m=Word2Vec(translated_df_Grp0["Translated_Description"],size=200,min_count=1,sg=1)
def vectorizer(sent,m):
    vec=[]
    numw=0
    for w in sent:
        try:
            if numw==0:
                vec=m[w]
            else:
                vec=np.add(vec,m[w])
            numw+=1
        except:
            pass

        return np.asarray(vec)/numw
l=[]
for i in translated_df_Grp0["Translated_Description"]:
    l.append(vectorizer(i,m))
X=np.array(l)

# Using KMeans clustering to find sub-clusters within Group_0 Rows
n_clusters=8
clf=KMeans(n_clusters=n_clusters, max_iter=100,init='k-means++',n_init=1)
labels=clf.fit_predict(X)

translated_df_Grp0["New Assignment grp"]= "0"

a=0
for index_label, row in translated_df_Grp0.iterrows():
    translated_df_Grp0.at[index_label,'New Assignment grp'] = "GRP_0"+"_"+str(labels[a])
    a=a+1

# translated_df_Grp0["New Assignment grp"].value_counts().to_list()

# Changing Group_0 values to new labels at their respective indices
for index_label, row in translated_df_Grp0.iterrows():
    val=translated_df_Grp0.at[index_label,'index']
    val=val.astype(int)
    translated_df.at[val , 'Assignment group'] = translated_df_Grp0.at[index_label,'New Assignment
grp']
```