Import packages

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
import os
import string
import nltk
import warnings
%matplotlib inline

warnings.filterwarnings('ignore')
```

Data Acquisition

In [2]:
```
# mapping of n1 n2 n3 values to category names
category_mapping = {
    "1": "Coarse genre",
    "1.1": "Company Business, Strategy, etc. (elaborate in Section 3 [Topics])",
    "1.2": "Purely Personal",
    "1.3": "Personal but in a professional context (e.g., it was good working with
    "1.4": "Logistic Arrangements (meeting scheduling, technical support, etc)",
    "1.5": "Employment arrangements (job seeking, hiring, recommendations, etc)",
    "1.6": "Document editing/checking (collaboration)",
    "1.7": "Empty message (due to missing attachment)",
    "1.8": "Empty message",
    "2": "Included/forwarded information",
    "2.1": "Includes new text in addition to forwarded material",
    "2.2": "Forwarded email(s) including replies",
    "2.3": "Business letter(s) / document(s)",
    "2.4": "News article(s)",
    "2.5": "Government / academic report(s)",
    "2.6": "Government action(s) (such as results of a hearing, etc)",
    "2.7": "Press release(s)",
    "2.8": "Legal documents (complaints, lawsuits, advice)",
    "2.9": "Pointers to url(s)",
    "2.10": "Newsletters",
    "2.11": "Jokes, humor (related to business)",
    "2.12": "Jokes, humor (unrelated to business)",
    "2.13": "Attachment(s) (assumed missing)",
    "3": "Primary topics (if coarse genre 1.1 is selected)",
    "3.1": "Regulations and regulators (includes price caps)",
    "3.2": "Internal projects -- progress and strategy",
    "3.3": "Company image -- current",
    "3.4": "Company image -- changing / influencing",
    "3.5": "Political influence / contributions / contacts",
    "3.6": "California energy crisis / California politics",
    "3.7": "Internal company policy",
    "3.8": "Internal company operations",
    "3.9": "Alliances / partnerships",
    "3.10": "Legal advice",
    "3.11": "Talking points",
    "3.12": "Meeting minutes",
    "3.13": "Trip reports",
    "4": "Emotional tone (if not neutral)",
```

```
    "4.1": "Jubilation",
    "4.2": "Hope / anticipation",
    "4.3": "Humor",
    "4.4": "Camaraderie",
    "4.5": "Admiration",
    "4.6": "Gratitude",
    "4.7": "Friendship / affection",
    "4.8": "Sympathy / support",
    "4.9": "Sarcasm",
    "4.10": "Secrecy / confidentiality",
    "4.11": "Worry / anxiety",
    "4.12": "Concern",
    "4.13": "Competitiveness / aggressiveness",
    "4.14": "Triumph / gloating",
    "4.15": "Pride",
    "4.16": "Anger / agitation",
    "4.17": "Sadness / despair",
    "4.18": "Shame",
    "4.19": "Dislike / scorn"
}

#extract email text from all files
def decode_cats_file(file_content):
    decoded_info = []
    decoded_n2 = []
    lines = file_content.strip().split('\n')
    for line in lines:
        n1, n2, n3 = line.strip().split(',')
        category_n1 = category_mapping.get(n1, "Unknown Category")
        category_n2 = category_mapping.get(f"{n1}.{n2}", "Unknown Category")
        frequency = n3
        decoded_info.append(f"{n1},{n2},{n3}:\n"
                            f"n1 = {n1} ({category_n1})\n"
                            f"n2 = {n2} ({category_n2})\n"
                            f"n3 = {n3} (Frequency)")
        decoded_n2.append(category_n2)
    return decoded_info, decoded_n2

def extract_text_from_txt(file_path):
    try:
        with open(file_path, "r", encoding="utf-8") as txt_file:
            return txt_file.read()
    except Exception as e:
        print(f"Error reading text from {file_path}: {str(e)}")
        return ""

def process_files(root_folder):
    data = {"Folder": [], "File": [], "Decoded_Info": [], "Decoded_n2": [], "Text_C

    for root, _, files in os.walk(root_folder):
        for file in files:
            file_name, file_ext = os.path.splitext(file)
            file_path = os.path.join(root, file)

            if file_ext == ".cats":
                with open(file_path, "r", encoding="utf-8") as cats_file:
                    file_content = cats_file.read()
                    decoded_info, decoded_n2 = decode_cats_file(file_content)
            elif file_ext == ".txt":
                text_content = extract_text_from_txt(file_path)
            else:
                continue

            data["Folder"].append(root)
```

```
            data["File"].append(file_name)  # Store without extension
            data["Decoded_Info"].append(decoded_info if file_ext == ".cats" else [
            data["Decoded_n2"].append(decoded_n2 if file_ext == ".cats" else [])
            data["Text_Content"].append(text_content if file_ext == ".txt" else "")

    df = pd.DataFrame(data)
    return df

root_folder = "D:\V labs assignment"

df = process_files(root_folder)

# Extract the first element from each list in the Decoded_n2 column
df['First_Decoded_n2'] = df['Decoded_n2'].str[0]


df = df.drop(columns=["Decoded_Info", "Decoded_n2"])

# Group the DataFrame by the "File" column and aggregate the text content
df_merged = df.groupby("File").agg({
    "Text_Content": "\n".join,
    "First_Decoded_n2": "first"
}).reset_index()
```

In [3]:
```
df_merged.head()
```

Out[3]:

	File	Text_Content	First_Decoded_n2
0	10425	\nMessage-ID: <197504.1075840201539.JavaMail.e...	Company Business, Strategy, etc. (elaborate in...
1	106296	\nMessage-ID: <11991339.1075842536086.JavaMail...	Company Business, Strategy, etc. (elaborate in...
2	106298	\nMessage-ID: <7106753.1075842536132.JavaMail....	Company Business, Strategy, etc. (elaborate in...
3	106588	\nMessage-ID: <21267718.1075863331587.JavaMail...	Company Business, Strategy, etc. (elaborate in...
4	106590	\nMessage-ID: <20866019.1075863331634.JavaMail...	Company Business, Strategy, etc. (elaborate in...

In [14]:
```
# Rename the "First_Decoded_n2" column to "Category"
final_df= df_merged .rename(columns={"First_Decoded_n2": "Category", "Text_Content"
```

In [15]:
```
final_df.head()
```

Out[15]:

	File	Text	Category
0	10425	\nMessage-ID: <197504.1075840201539.JavaMail.e...	Company Business, Strategy, etc. (elaborate in...
1	106296	\nMessage-ID: <11991339.1075842536086.JavaMail...	Company Business, Strategy, etc. (elaborate in...
2	106298	\nMessage-ID: <7106753.1075842536132.JavaMail....	Company Business, Strategy, etc. (elaborate in...
3	106588	\nMessage-ID: <21267718.1075863331587.JavaMail...	Company Business, Strategy, etc. (elaborate in...
4	106590	\nMessage-ID: <20866019.1075863331634.JavaMail...	Company Business, Strategy, etc. (elaborate in...

In [16]:
```
final_df = df_merged.drop(1702)
```

In [17]:
```
final_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1702 entries, 0 to 1701
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   File      1702 non-null   object
 1   Text      1702 non-null   object
 2   Category  1702 non-null   object
dtypes: object(3)
memory usage: 40.0+ KB
```

In [18]:
```
# missing values
final_df.isnull().sum()
```

Out[18]:
```
File        0
Text        0
Category    0
dtype: int64
```

In [19]:
```
final_df.duplicated().sum()
```

Out[19]:
```
0
```

In [20]:
```
final_df.shape
```

Out[20]:
```
(1702, 3)
```

EDA

In [21]:
```
final_df.head()
```

Out[21]:

	File	Text	Category
0	10425	\nMessage-ID: <197504.1075840201539.JavaMail.e...	Company Business, Strategy, etc. (elaborate in...
1	106296	\nMessage-ID: <11991339.1075842536086.JavaMail...	Company Business, Strategy, etc. (elaborate in...
2	106298	\nMessage-ID: <7106753.1075842536132.JavaMail....	Company Business, Strategy, etc. (elaborate in...
3	106588	\nMessage-ID: <21267718.1075863331587.JavaMail...	Company Business, Strategy, etc. (elaborate in...
4	106590	\nMessage-ID: <20866019.1075863331634.JavaMail...	Company Business, Strategy, etc. (elaborate in...

In [22]:
```
final_df['Category'].value_counts()
```

Out[22]:
```
Company Business, Strategy, etc. (elaborate in Section 3 [Topics])                855
Logistic Arrangements (meeting scheduling, technical support, etc)                426
Document editing/checking (collaboration)                                         135
Personal but in a professional context (e.g., it was good working with you)       135
Employment arrangements (job seeking, hiring, recommendations, etc)                64
Purely Personal                                                                    48
Empty message (due to missing attachment)                                          21
Empty message                                                                      18
Name: Category, dtype: int64
```

In [23]:
```
#encoding
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
final_df['Category1'] = encoder.fit_transform(final_df['Category'])
final_df.head()
```

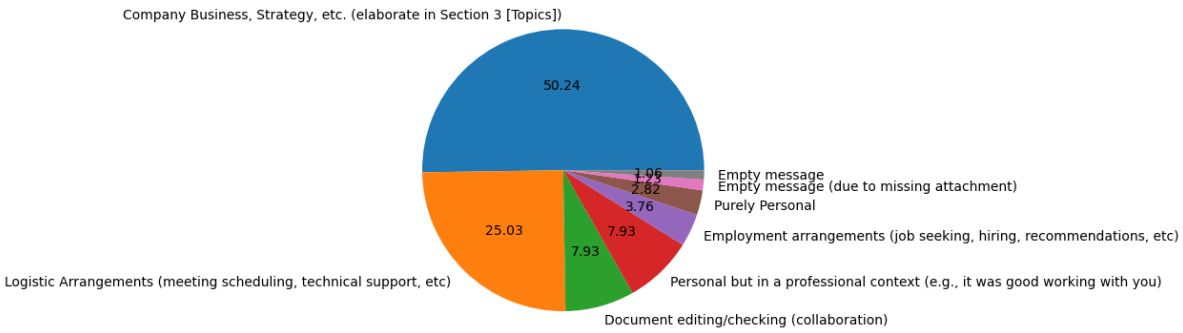
Out[23]:

	File	Text	Category	Category1
0	10425	\nMessage-ID: <197504.1075840201539.JavaMail.e...	Company Business, Strategy, etc. (elaborate in...	0
1	106296	\nMessage-ID: <11991339.1075842536086.JavaMail...	Company Business, Strategy, etc. (elaborate in...	0
2	106298	\nMessage-ID: <7106753.1075842536132.JavaMail....	Company Business, Strategy, etc. (elaborate in...	0
3	106588	\nMessage-ID: <21267718.1075863331587.JavaMail...	Company Business, Strategy, etc. (elaborate in...	0
4	106590	\nMessage-ID: <20866019.1075863331634.JavaMail...	Company Business, Strategy, etc. (elaborate in...	0

In [24]:
```
final_df['Category1'].value_counts()
```

Out[24]:
```
0    855
5    426
1    135
6    135
2     64
7     48
4     21
3     18
Name: Category1, dtype: int64
```

In [25]:
```
#data distribution
plt.pie(final_df['Category'].value_counts(), labels=['Company Business, Strategy, e
plt.show()
```



In [26]:
```
final_df['num_characters'] = final_df['Text'].apply(len)
```

In [27]:
```
final_df.head()
```

Out[27]:

	File	Text	Category	Category1	num_characters
0	10425	\nMessage-ID: <197504.1075840201539.JavaMail.e...	Company Business, Strategy, etc. (elaborate in...	0	1740
1	106296	\nMessage-ID: <11991339.1075842536086.JavaMail...	Company Business, Strategy, etc. (elaborate in...	0	1877
2	106298	\nMessage-ID: <7106753.1075842536132.JavaMail....	Company Business, Strategy, etc. (elaborate in...	0	2521
3	106588	\nMessage-ID: <21267718.1075863331587.JavaMail...	Company Business, Strategy, etc. (elaborate in...	0	2615
4	106590	\nMessage-ID: <20866019.1075863331634.JavaMail...	Company Business, Strategy, etc. (elaborate in...	0	2008

In [28]:
```
# num of words
final_df['num_words'] = final_df['Text'].apply(lambda x:len(nltk.word_tokenize(x))
```

In [29]:
```
final_df.head()
```

Out[29]:

	File	Text	Category	Category1	num_characters	num_words
0	10425	\nMessage-ID: <197504.1075840201539.JavaMail.e...	Company Business, Strategy, etc. (elaborate in...	0	1740	339
1	106296	\nMessage-ID: <11991339.1075842536086.JavaMail...	Company Business, Strategy, etc. (elaborate in...	0	1877	316
2	106298	\nMessage-ID: <7106753.1075842536132.JavaMail....	Company Business, Strategy, etc. (elaborate in...	0	2521	460
3	106588	\nMessage-ID: <21267718.1075863331587.JavaMail...	Company Business, Strategy, etc. (elaborate in...	0	2615	493
4	106590	\nMessage-ID: <20866019.1075863331634.JavaMail...	Company Business, Strategy, etc. (elaborate in...	0	2008	355

In [30]:
```
final_df['num_sentences'] = final_df['Text'].apply(lambda x:len(nltk.sent_tokenize(
```

In [31]:
```
final_df.head()
```

Out[31]:

	File	Text	Category	Category1	num_characters	num_words
0	10425	\nMessage-ID: <197504.1075840201539.JavaMail.e...	Company Business, Strategy, etc. (elaborate in...	0	1740	339
1	106296	\nMessage-ID: <11991339.1075842536086.JavaMail...	Company Business, Strategy, etc. (elaborate in...	0	1877	316
2	106298	\nMessage-ID: <7106753.1075842536132.JavaMail....	Company Business, Strategy, etc. (elaborate in...	0	2521	460
3	106588	\nMessage-ID: <21267718.1075863331587.JavaMail...	Company Business, Strategy, etc. (elaborate in...	0	2615	493
4	106590	\nMessage-ID: <20866019.1075863331634.JavaMail...	Company Business, Strategy, etc. (elaborate in...	0	2008	355

In [32]:
```
final_df[['num_characters','num_words','num_sentences']].describe()
```

Out[32]:

	num_characters	num_words	num_sentences
count	1702.000000	1702.000000	1702.000000
mean	7443.619271	1390.138660	40.250294
std	24280.846103	4605.935478	151.918280
min	394.000000	61.000000	1.000000
25%	1101.500000	204.000000	4.000000
50%	2047.500000	376.000000	9.000000
75%	3647.500000	654.000000	18.000000
max	225783.000000	44723.000000	2112.000000

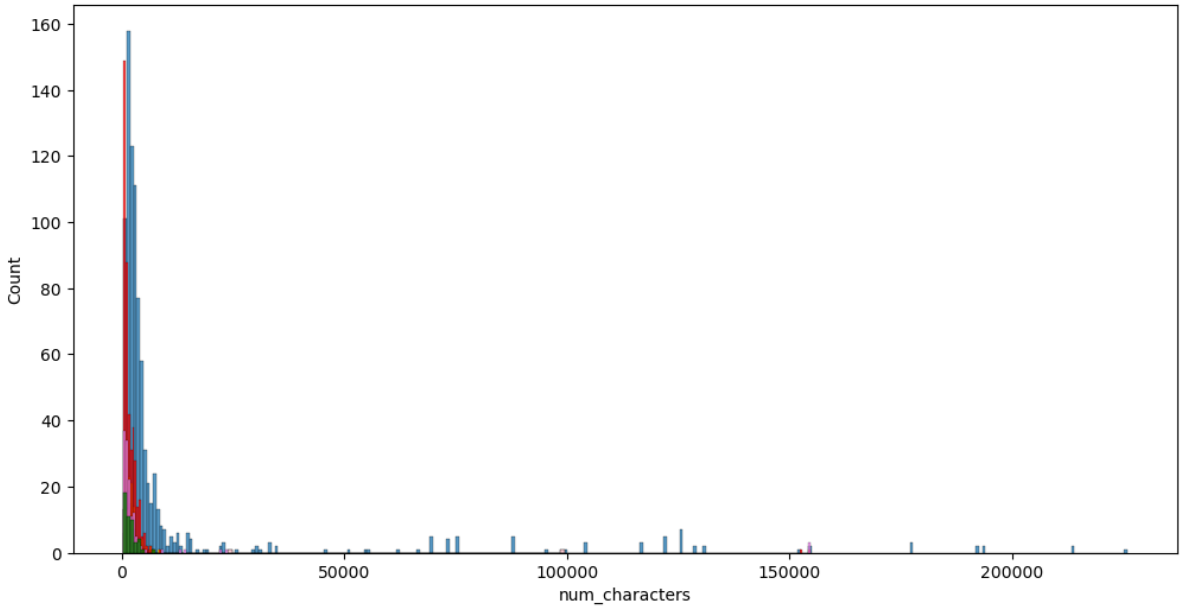
In [33]:
```
import seaborn as sns
```

In [34]:
```
#plot of num_characters of email text
plt.figure(figsize=(12,6))
sns.histplot(final_df[final_df['Category1'] == 0]['num_characters'])
sns.histplot(final_df[final_df['Category1'] == 1]['num_characters'],color='yellow')
```

```
sns.histplot(final_df[final_df['Category1'] == 2]['num_characters'],color='blue')
sns.histplot(final_df[final_df['Category1'] == 3]['num_characters'],color='pink')
sns.histplot(final_df[final_df['Category1'] == 4]['num_characters'],color='orange')
sns.histplot(final_df[final_df['Category1'] == 5]['num_characters'],color='red')
sns.histplot(final_df[final_df['Category1'] == 6]['num_characters'],color='violet')
sns.histplot(final_df[final_df['Category1'] == 7]['num_characters'],color='green')
```

Out[34]:

```
<AxesSubplot:xlabel='num_characters', ylabel='Count'>
```

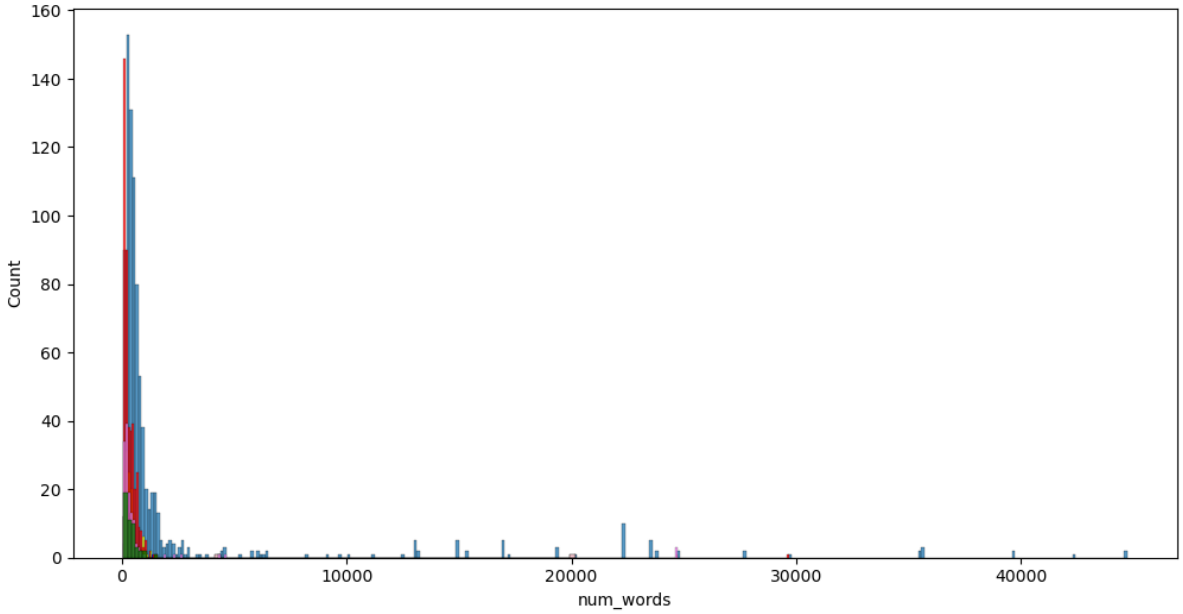


In [35]:

```
#plot of num_words of email text
plt.figure(figsize=(12,6))
sns.histplot(final_df[final_df['Category1'] == 0]['num_words'])
sns.histplot(final_df[final_df['Category1'] == 1]['num_words'],color='yellow')
sns.histplot(final_df[final_df['Category1'] == 2]['num_words'],color='blue')
sns.histplot(final_df[final_df['Category1'] == 3]['num_words'],color='pink')
sns.histplot(final_df[final_df['Category1'] == 4]['num_words'],color='orange')
sns.histplot(final_df[final_df['Category1'] == 5]['num_words'],color='red')
sns.histplot(final_df[final_df['Category1'] == 6]['num_words'],color='violet')
sns.histplot(final_df[final_df['Category1'] == 7]['num_words'],color='green')
```

Out[35]:

```
<AxesSubplot:xlabel='num_words', ylabel='Count'>
```



Data Cleaning and Preprocessing

In [36]:
```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to C:\Users\Kunal
[nltk_data]     Rane\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[36]:
```
True
```

In [37]:
```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to C:\Users\Kunal
[nltk_data]     Rane\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[37]:
```
True
```

In [38]:
```
from nltk.corpus import stopwords
stopwords.words('english')
```

Out[38]:
```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
 'his',
 'himself',
 'she',
 "she's",
 'her',
 'hers',
 'herself',
 'it',
 "it's",
 'its',
 'itself',
 'they',
 'them',
 'their',
 'theirs',
 'themselves',
 'what',
 'which',
 'who',
 'whom',
 'this',
 'that',
 "that'll",
 'these',
 'those',
 'am',
 'is',
 'are',
 'was',
 'were',
 'be',
 'been',
 'being',
 'have',
 'has',
 'had',
 'having',
 'do',
 'does',
 'did',
 'doing',
 'a',
 'an',
 'the',
 'and',
```

```
                'but',
                'if',
                'or',
                'because',
                'as',
                'until',
                'while',
                'of',
                'at',
                'by',
                'for',
                'with',
                'about',
                'against',
                'between',
                'into',
                'through',
                'during',
                'before',
                'after',
                'above',
                'below',
                'to',
                'from',
                'up',
                'down',
                'in',
                'out',
                'on',
                'off',
                'over',
                'under',
                'again',
                'further',
                'then',
                'once',
                'here',
                'there',
                'when',
                'where',
                'why',
                'how',
                'all',
                'any',
                'both',
                'each',
                'few',
                'more',
                'most',
                'other',
                'some',
                'such',
                'no',
                'nor',
                'not',
                'only',
                'own',
                'same',
                'so',
                'than',
                'too',
                'very',
                's',
                't',
```

```
 'can',
 'will',
 'just',
 'don',
 "don't",
 'should',
 "should've",
 'now',
 'd',
 'll',
 'm',
 'o',
 're',
 've',
 'y',
 'ain',
 'aren',
 "aren't",
 'couldn',
 "couldn't",
 'didn',
 "didn't",
 'doesn',
 "doesn't",
 'hadn',
 "hadn't",
 'hasn',
 "hasn't",
 'haven',
 "haven't",
 'isn',
 "isn't",
 'ma',
 'mightn',
 "mightn't",
 'mustn',
 "mustn't",
 'needn',
 "needn't",
 'shan',
 "shan't",
 'shouldn',
 "shouldn't",
 'wasn',
 "wasn't",
 'weren',
 "weren't",
 'won',
 "won't",
 'wouldn',
 "wouldn't"]
```

In [39]:
```
import string
string.punctuation
```

Out[39]:
```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

In [40]:
```
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()
```

In [41]:
```
#lowering, removing stopwords,removing punctuation and stemming
def transform_text(text):
    text = text.lower()
```

```
    text = nltk.word_tokenize(text)

    y = []
    for i in text:
        if i.isalnum():
            y.append(i)

    text = y[:]
    y.clear()

    for i in text:
        if i not in stopwords.words('english') and i not in string.punctuation:
            y.append(i)

    text = y[:]
    y.clear()

    for i in text:
        y.append(ps.stem(i))


    return " ".join(y)
```

In [42]:
```
transform_text("I'm gonna be home soon and i don't want to talk about this stuff ar
```

Out[42]:
```
'gon na home soon want talk stuff anymor tonight k cri enough today'
```

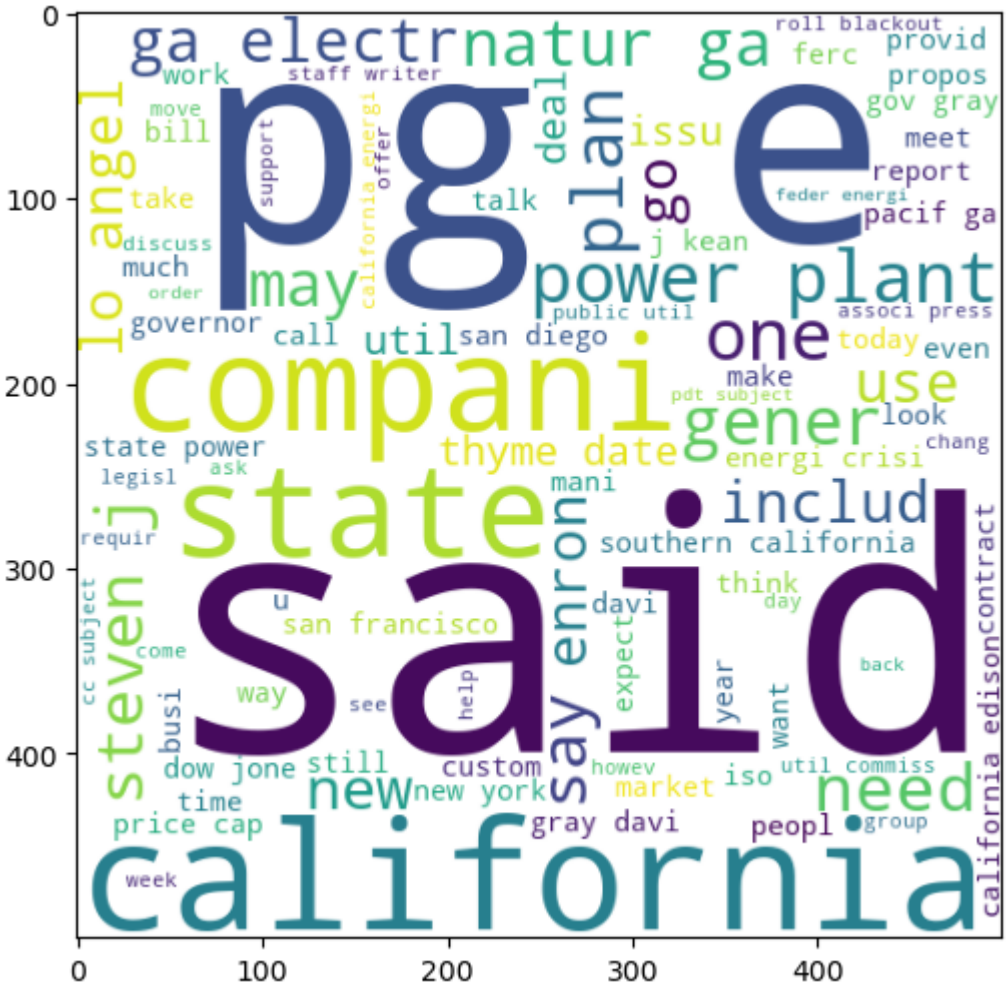
In [43]:
```
final_df['transformed_text'] = final_df["Text"].apply(transform_text)
```

In [44]:
```
from wordcloud import WordCloud
wc = WordCloud(width=500,height=500,min_font_size=10,background_color='white')
```

In [45]:
```
Company = wc.generate(final_df[final_df['Category1'] == 0]['transformed_text'].str
```

In [46]:
```
plt.figure(figsize=(15,6))
plt.imshow(Company)
```

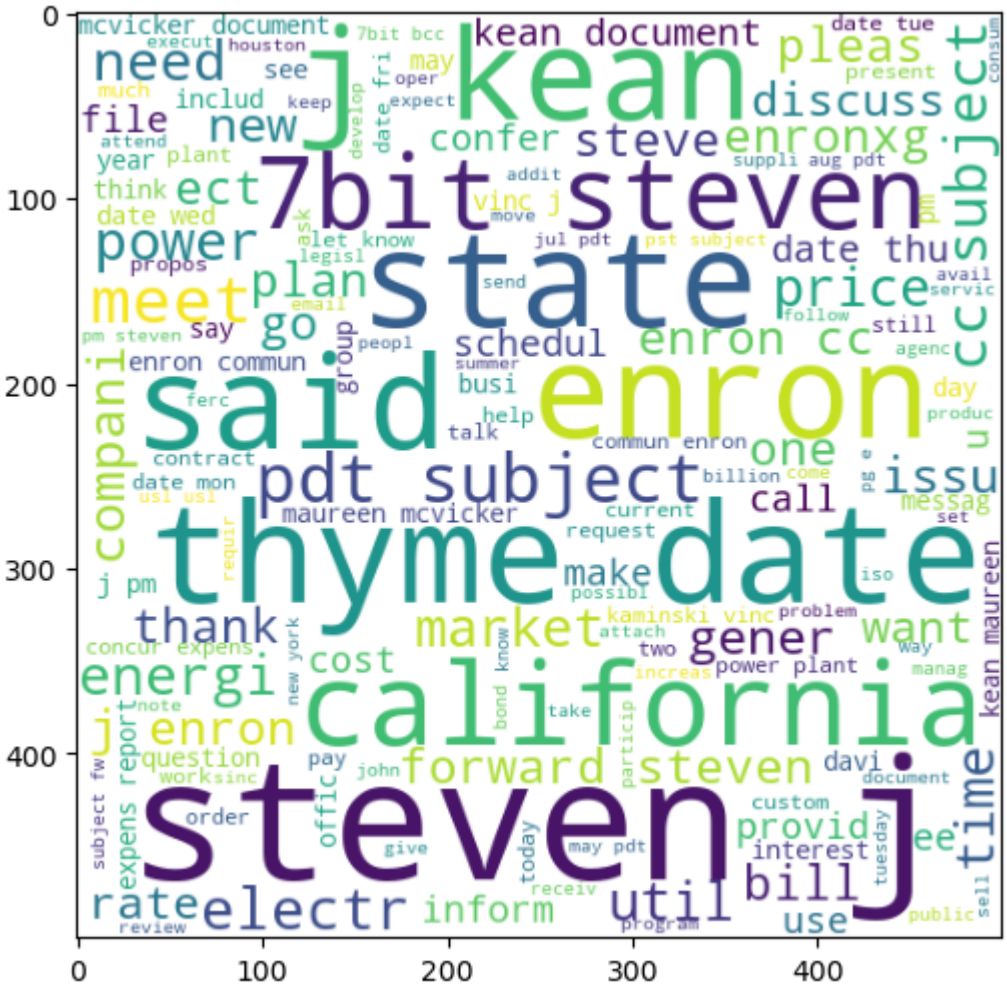
Out[46]:
```
<matplotlib.image.AxesImage at 0x1df12c34880>
```



In [47]:
```
Logistic = wc.generate(final_df[final_df['Category1'] == 5]['transformed_text'].str
```

In [48]:
```
plt.figure(figsize=(15,6))
plt.imshow(Logistic)
```

Out[48]:
```
<matplotlib.image.AxesImage at 0x1df12bfaee0>
```



Train Test Split

In [49]:
```
from sklearn.model_selection import train_test_split
X = final_df.Text
y = final_df.Category1
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [50]:
```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[50]:
```
((1361,), (341,), (1361,), (341,))
```

Applying Logistic Regression

In [51]:
```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import accuracy_score
```

In [52]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfTransformer

lr = Pipeline([('vect', CountVectorizer()),
               ('tfidf', TfidfTransformer()),
               ('clf', LogisticRegression()),
              ])

lr.fit(X_train,y_train)
```

```
y_pred1 = lr.predict(X_test)

print(f"Accuracy is : {accuracy_score(y_pred1,y_test)}")
```

```
Accuracy is : 0.6598240469208211
```

Applying Naive Bayes Classifier

In [53]:
```
from sklearn.naive_bayes import MultinomialNB


naivebayes = Pipeline([('vect', CountVectorizer()),
               ('tfidf', TfidfTransformer()),
               ('clf', MultinomialNB()),
              ])
naivebayes.fit(X_train, y_train)

y_pred = naivebayes.predict(X_test)

print(f'accuracy {accuracy_score(y_pred,y_test)}')
```

```
accuracy 0.5219941348973607
```

Applying Xgboost Classifier

In [54]:
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
final_df['Category1'] = encoder.fit_transform(final_df['Category'])
final_df.head()
```

Out[54]:

	File	Text	Category	Category1	num_characters	num_words
0	10425	\nMessage-ID: <197504.1075840201539.JavaMail.e...	Company Business, Strategy, etc. (elaborate in...	0	1740	339
1	106296	\nMessage-ID: <11991339.1075842536086.JavaMail...	Company Business, Strategy, etc. (elaborate in...	0	1877	316
2	106298	\nMessage-ID: <7106753.1075842536132.JavaMail....	Company Business, Strategy, etc. (elaborate in...	0	2521	460
3	106588	\nMessage-ID: <21267718.1075863331587.JavaMail...	Company Business, Strategy, etc. (elaborate in...	0	2615	493
4	106590	\nMessage-ID: <20866019.1075863331634.JavaMail...	Company Business, Strategy, etc. (elaborate in...	0	2008	355

In [55]:
```
from sklearn.model_selection import train_test_split
X = final_df.Text
y = final_df.Category1
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [56]:
```
from xgboost import XGBClassifier

xgboost = Pipeline([('vect', CountVectorizer()),
                ('tfidf', TfidfTransformer()),
                ('clf', XGBClassifier()),
               ])
xgboost.fit(X_train, y_train)

y_pred = xgboost.predict(X_test)

print(f'accuracy {accuracy_score(y_pred,y_test)}')
```

```
accuracy 0.6979472140762464
```

Applying Random forest classifier

In [57]:
```
from sklearn.ensemble import RandomForestClassifier

rf_classifier = Pipeline([('vect', CountVectorizer()),
```

```
                ('tfidf', TfidfTransformer()),
                ('clf', RandomForestClassifier()),
               ])

rf_classifier.fit(X_train, y_train)

y_pred = rf_classifier.predict(X_test)

print(f'accuracy {accuracy_score(y_pred,y_test)}')
```

```
accuracy 0.6304985337243402
```

Smote for balancing data

In [58]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder


# Text preprocessing (TF-IDF vectorization)
tfidf_vectorizer = TfidfVectorizer(max_features=1000)  # Adjust max_features as nee
X = tfidf_vectorizer.fit_transform(final_df['transformed_text'])

# Encoding target labels
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(final_df['Category'])
```

In [59]:
```
from imblearn.combine import SMOTEENN
from sklearn.datasets import make_classification

# Create an instance of SMOTE-ENN resampler
smote_enn = SMOTEENN(sampling_strategy='auto', random_state=42)

# Resample the dataset using SMOTE-ENN
X_resampled, y_resampled = smote_enn.fit_resample(X , y )

# Check the class distribution after resampling
unique, counts = np.unique(y_resampled, return_counts=True)
class_distribution = dict(zip(unique, counts))
print("Class distribution after SMOTE-ENN resampling:")
print(class_distribution)
```

```
Class distribution after SMOTE-ENN resampling:
{0: 170, 1: 815, 2: 849, 3: 851, 4: 853, 5: 583, 6: 824, 7: 851}
```

Train Test Split after balancing data

In [60]:
```
from sklearn.model_selection import train_test_split
X = final_df.Text
y = final_df.Category
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_
```

In [61]:
```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[61]:
```
((4636, 1000), (1160, 1000), (4636,), (1160,))
```

Applying Logistic Regression after balancing data

In [62]:
```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import accuracy_score
```

In [63]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfTransformer

lr = Pipeline([
               ('clf', LogisticRegression()),
              ])

lr.fit(X_train,y_train)
y_pred1 = lr.predict(X_test)

print(f"Accuracy is : {accuracy_score(y_pred1,y_test)}")
```

```
Accuracy is : 0.9568965517241379
```

Applying Naive Bayes Classifier after balancing data

In [64]:
```
from sklearn.naive_bayes import MultinomialNB


naivebayes = Pipeline([
               ('clf', MultinomialNB()),
              ])
naivebayes.fit(X_train, y_train)

y_pred = naivebayes.predict(X_test)

print(f'accuracy {accuracy_score(y_pred,y_test)}')
```

```
accuracy 0.8663793103448276
```

Applying Xgboost Classifier after balancing data

In [65]:
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
final_df['Category1'] = encoder.fit_transform(final_df['Category'])
```

In [66]:
```
from sklearn.model_selection import train_test_split
X = final_df.Text
y = final_df.Category1
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_
```

In [67]:
```
from xgboost import XGBClassifier

xgboost = Pipeline([
               ('clf', XGBClassifier()),
              ])
xgboost.fit(X_train, y_train)

y_pred = xgboost.predict(X_test)
```

```
print(f'accuracy {accuracy_score(y_pred,y_test)}')
```

```
accuracy 0.9905172413793103
```

Applying Random forest classifier after balancing data

In [68]:
```
from sklearn.ensemble import RandomForestClassifier

rf_classifier = Pipeline([
                ('clf', RandomForestClassifier()),
               ])

rf_classifier.fit(X_train, y_train)

y_pred = rf_classifier.predict(X_test)

print(f'accuracy {accuracy_score(y_pred,y_test)}')
from sklearn.ensemble import RandomForestClassifier

rf_classifier = Pipeline([
                ('clf', RandomForestClassifier()),
               ])

rf_classifier.fit(X_train, y_train)

y_pred = rf_classifier.predict(X_test)

print(f'accuracy {accuracy_score(y_pred,y_test)}')
```

```
accuracy 0.9896551724137931
accuracy 0.9905172413793103
```

In [ ]:
```

```