In [129]:
```
## importing all necessary packages and libraries
import pandas as pd
import re
import gensim
from gensim.utils import simple_preprocess
import nltk
from nltk.corpus import stopwords
from wordcloud import WordCloud
import pyLDAvis.gensim
import os
import pickle
import pyLDAvis
import warnings
warnings.filterwarnings('ignore')
import csv
from sklearn.metrics.pairwise import cosine_similarity
import gensim.corpora as corpora
from itertools import chain
from collections import defaultdict
from scipy.stats import percentileofscore
from gensim.parsing.preprocessing import STOPWORDS
from nltk.stem import WordNetLemmatizer, SnowballStemmer
from nltk.stem.porter import *
from nltk.corpus import wordnet
import numpy as np
np.random.seed(42)
from pprint import pprint
nltk.download('wordnet')
nltk.download('omw-1.4')
from gensim.models import CoherenceModel
from gensim.corpora import Dictionary
from gensim.models import LdaModel
from sklearn.feature_extraction.text import TfidfVectorizer
from gensim.models import TfidfModel
from sklearn.metrics.pairwise import pairwise_distances
from sklearn.metrics import pairwise_distances
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Manu\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\Manu\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

In [89]:
```
## Reading the dataframe into the master dataframe.
# File name - 'Contact Center Ticket Description.csv' should be present in the working directory for the code to fetch the da
masData = pd.read_csv('Contact Center Ticket Description.csv', sep='|',header = 0, dtype = {0: str}, names = ['Description',
```

In [90]:
```
#Creating a copy of the master dataframe in df.
df = masData.copy()
```

In [91]:
```
## Filter for 1 source org and define the intial preprocessing steps.
print("Kindly choose from the below list of source orgs for the input variable: \n")
unique_source_org = masData['Source Org'].drop_duplicates().tolist()

print(unique_source_org)
```

```
Kindly choose from the below list of source orgs for the input variable:

['TAA APAC', 'EU GBS Log Cases', 'EU NL BaseCone', 'NA Accounting Services', 'NA AVA', 'NA Canada', 'NA TeamMate', 'TAA NA', 'Heal
R', 'GRC ELM', 'GRC Lien', 'LR US', 'LR Belgium', 'LR Netherlands', 'LR Italy', 'LR Poland']
```

In [92]:
```
source_org = input("Enter the source org of choice from the list provided: ")
```

```
Enter the source org of choice from the list provided: NA Canada
```

In [93]:
```
print(f"Topic modelling will proceed for the source org: {source_org}")
```

```
Topic modelling will proceed for the source org: NA Canada
```

In [94]:
```
#Pre-processing steps for the dataframe df for the specified source org.
def pre_processing(data):
    data=data[data['Source Org']==source_org]
    data=data.dropna()
    data['Description'] = data['Description'].astype(str)
    data['Description'] = data['Description'].str.lower()
    pattern = r'[^a-zA-Z0-9\s]'
    data=data.apply(lambda x: x.str.strip())
    data['Description'] = data['Description'].apply(lambda x:re.sub(pattern, '', x))
    return data
```

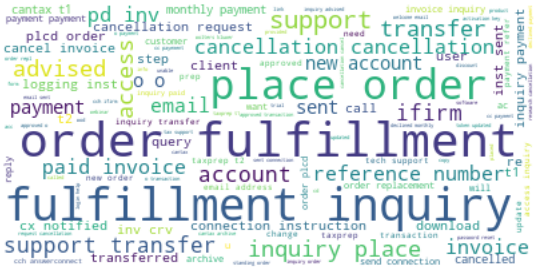
In [95]:
```
## Perform the preprocessing using generated function.
df = pre_processing(df)
```

In [96]:
```
#Checking shape (Rows, Columns) of the dataframe for topic modelling.
df.shape
```

Out[96]: (59467, 2)

In [97]:
```
#Generating wordcloud to find identify word composition.
long_string = ','.join(list(df['Description'].values))
wordcloud = WordCloud(background_color="white", max_words=5000, contour_width=3, contour_color='steelblue')
wordcloud.generate(long_string)
wordcloud.to_image()
```

Out[97]:



In [98]:
```
## Generating default stopwordlist using inbuilt python library.
stop_words = stopwords.words('english')

#Extending stopword list by adding words of less relevance in topic modelling, derived from the wordcloud.
#The list is subject to change, for a source org in consideration.
stop_words.extend(['wolters', 'kluwer', 'from', 'subject', 're', 'edu', 'use', 'fw', 'Fw', 'FWD', 'fwd', 'hi', 'may', 'know',
                   'regards', 'iphone', 'android', 'interested', 'australia', 'australian', 'wolterskluwer'])

#Defining a function to breakdown each sentence to individual tokens.
def sent_to_words(sentences):
    for sentence in sentences:
        yield(gensim.utils.simple_preprocess(str(sentence), deacc=True))

#Defining a function to remove stopwords.
def remove_stopwords(texts):
    return [[word for word in simple_preprocess(str(doc)) if word not in stop_words] for doc in texts]

#Converting individual tokens into a list of lists, where the inner list is a list of individual tokens and naming it data_wo
data = df['Description'].values.tolist()
data_words = list(sent_to_words(data))

#Removing defined stopwords from the generated stopword list from data_words.
data_words = remove_stopwords(data_words)

#Retaining all words with a length greater than 2.
data_words = [[word for word in sublist if len(word) > 2] for sublist in data_words]
```

In [99]:
```
#Lemmatizing words to remove verbs and changing to root word.
def lemmatize(text):
    return WordNetLemmatizer().lemmatize(text, pos='v')

def preprocess(text_list):
    result = []
    for text in text_list:
        processed_text = []
        for token in gensim.utils.simple_preprocess(' '.join(text)):
            if token not in gensim.parsing.preprocessing.STOPWORDS and len(token) >= 3:
                processed_text.append(lemmatize(token))
        result.append(processed_text)
    return result

data_words=preprocess(data_words)

#Lemmatizing words to remove nouns and changing to root word.
def lemmatize(text):
    return WordNetLemmatizer().lemmatize(text, pos='n')

data_words=preprocess(data_words)

#Lemmatizing words to remove adjectives and changing to root word.
def lemmatize(text):
    return WordNetLemmatizer().lemmatize(text, pos='a')

data_words=preprocess(data_words)
```

In [100]:
```
# Create a dictionary to keep track of word frequencies.
frequency = defaultdict(int)

# Iterate over each document in data_words and updating word frequencies.
for text in data_words:
    for word in text:
        frequency[word] += 1
```

In [101]:
```
# Creating a list of dictionaries from the the words frequencies of each word.
my_dict = frequency
my_list = list(my_dict.items())
```

In [102]:
```
# Total count of all words in the dataset.
total_count = sum(count for word, count in my_list)

# Creating a list of percentanges of frequency of each word.
word_freq = [(word, round(count/total_count*100, 2)) for word, count in my_list]

# Calculate percentile ranks of the frequencies.
percentiles = [round(percentileofscore([freq for word, freq in word_freq], freq), 2) for word, freq in word_freq]

# Create a list of tuples with the word, frequency, and percentile rank.
word_freq_percentile = [(word, freq, percentile) for (word, freq), percentile in zip(word_freq, percentiles)]

# Sort the list by percentile rank in descending order.
word_freq_percentile_sorted = sorted(word_freq_percentile, key=lambda x: x[2], reverse=True)
```

In [103]:
```
# Creating a stop word list with words below the 80th Percentile threshold.
percentile_stop_words = [word for word, freq, percentile in word_freq_percentile_sorted if percentile < 80]
```

In [104]:
```
#Retaining data_words before removing percentile based stop words.
d1 = data_words
v1 = len(sum(d1, []))

#Removing stop words from data_words using percentile based stop words.
stop_words.extend(percentile_stop_words)
data_words = remove_stopwords(data_words)

#Retaining data_words after removing percentile based stop words.
d2 = data_words
v2 = len(sum(d2, []))

#Calculating percentage of words retained
percent_retained = (v2/v1)* 100

print("Word count before stopword removal is: ", v1)
print("Word count after stopword removal is: ", v2)
print("Percentage of words retained is: ", percent_retained)
```

```
Word count before stopword removal is:  175774
Word count after stopword removal is:  165342
Percentage of words retained is:  94.06510632971884
```

In [105]:
```
# Creating a corpora dictionary of unique tokens for topic modelling.
id2word = corpora.Dictionary(data_words)

# Storing a copy of data_words in texts field
texts = data_words

# Creating a corpus for topic modelling.
corpus = [id2word.doc2bow(text) for text in texts]
```

In [106]:
```
# Choosing the number of topics to in categorize into issue categories
num_topics = 7

# Giving number of topics,bag of words as inputs at an assigned random state 42 to excute our topic modelling
lda_model = gensim.models.LdaMulticore(corpus=corpus,
                                       id2word=id2word,
                                       num_topics=num_topics, random_state = 42)

## The output of this LDA represents the topic number with the weightage of the words contained within the topic
pprint(lda_model.print_topics())
doc_lda = lda_model[corpus]
if not os.path.exists('./results'):
    os.makedirs('./results')
```

```
[(0,
  '0.099*"payment" + 0.071*"inquiry" + 0.068*"order" + 0.026*"place" + '
  '0.025*"fulfillment" + 0.022*"pay" + 0.020*"download" + 0.020*"account" + '
  '0.018*"cantax" + 0.018*"invoice"'),
 (1,
  '0.088*"cancellation" + 0.056*"invoice" + 0.050*"inquiry" + 0.036*"order" + '
  '0.028*"update" + 0.026*"email" + 0.025*"payment" + 0.024*"cantax" + '
  '0.019*"support" + 0.019*"cancel"'),
 (2,
  '0.105*"support" + 0.093*"transfer" + 0.046*"email" + 0.032*"order" + '
  '0.030*"send" + 0.027*"inquiry" + 0.020*"tax" + 0.018*"cantax" + '
  '0.017*"taxprep" + 0.015*"change"'),
 (3,
  '0.053*"update" + 0.046*"taxprep" + 0.039*"token" + 0.035*"send" + '
  '0.026*"ifirm" + 0.026*"support" + 0.026*"contact" + 0.021*"invoice" + '
  '0.019*"payment" + 0.018*"order"'),
 (4,
  '0.080*"order" + 0.068*"inquiry" + 0.058*"place" + 0.049*"request" + '
  '0.048*"fulfillment" + 0.034*"inv" + 0.031*"account" + 0.027*"payment" + '
  '0.022*"crv" + 0.020*"number"'),
 (5,
  '0.058*"account" + 0.047*"invoice" + 0.047*"ifirm" + 0.041*"access" + '
  '0.035*"inquiry" + 0.031*"send" + 0.026*"taxprep" + 0.022*"pay" + '
  '0.020*"download" + 0.019*"order"'),
 (6,
  '0.201*"order" + 0.142*"inquiry" + 0.140*"place" + 0.133*"fulfillment" + '
  '0.029*"payment" + 0.027*"ifirm" + 0.017*"plcd" + 0.009*"cancel" + '
  '0.008*"refer" + 0.008*"send"')]
```

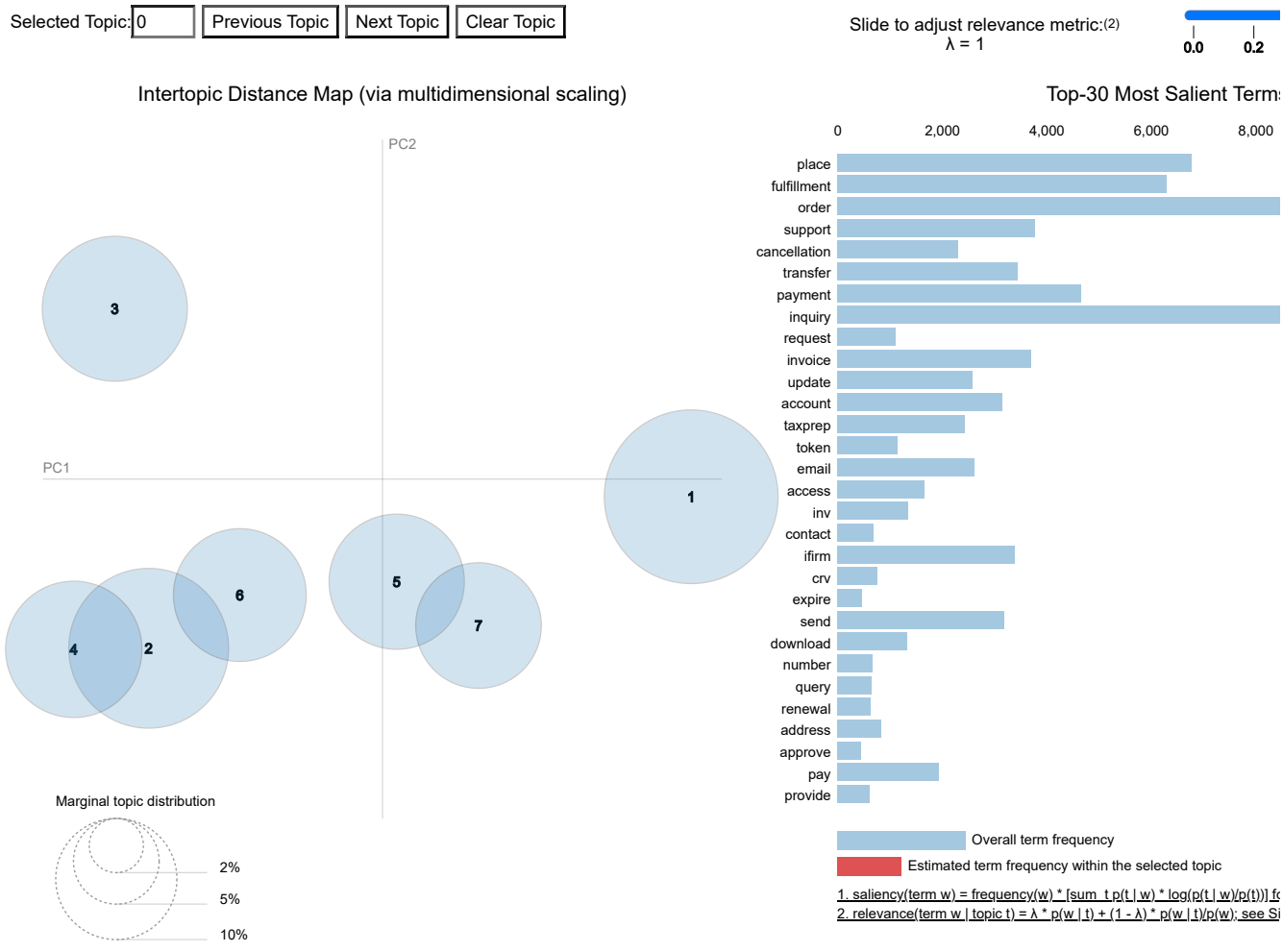
In [107]:
```
# Printing topics in a text format for better readability.
for topic_idx, topic_terms in lda_model.show_topics(num_topics=num_topics, num_words=40, formatted=False):
    print("Topic #{}:".format(topic_idx))
    for term in topic_terms:
        print("  {} ({:.3f})".format(term[0], term[1]))
    print("\n")
```

```
  research (0.007)
  replacement (0.007)
  cancel (0.007)
  cra (0.006)
  create (0.006)
  want (0.006)
  registration (0.006)
  token (0.006)
  issue (0.005)


Topic #1:
  cancellation (0.088)
  invoice (0.056)
  inquiry (0.050)
  order (0.036)
  update (0.028)
  email (0.026)
```

In [108]:

```
# Initializing pyLDAvis and defining the file path for the prepared data
pyLDAvis.enable_notebook()
LDAvis_data_filepath = os.path.join('./results/ldavis_prepared_'+str(num_topics))

# Visualizing the topic modeling results using pyLDAvis
if 1 == 1:
    LDAvis_prepared = pyLDAvis.gensim.prepare(lda_model, corpus, id2word)
    with open(LDAvis_data_filepath, 'wb') as f:
        pickle.dump(LDAvis_prepared, f)

# Saving the HTML visualization
with open(LDAvis_data_filepath, 'rb') as f:
    LDAvis_prepared = pickle.load(f)
pyLDAvis.save_html(LDAvis_prepared, './results/ldavis_prepared_'+ str(num_topics) +'lda.html')
LDAvis_prepared


# The above code creates two visualizations.

# Clusters representing the chosen number of topics in multidimentional scaling, where each access represents principal compo

# To the right we see top 30 salient terms/words across these topics and the length of the histogram is the representation of

# When we select one of the topics on the left, we see the top 30 salient words/terms which represent that particluar topic.

# The lambda
```

Out[108]:



In [109]:
```
#Computing perplexity score
print('\nPerplexity: ', lda_model.log_perplexity(corpus))

# instantiate topic coherence model
cm = CoherenceModel(model=lda_model, corpus=corpus, texts=texts, coherence='c_v')
# get topic coherence score
coherence_lda = cm.get_coherence()
print('\nCoherence: ', coherence_lda)
```

```
Perplexity:  -5.336973669946822

Coherence:  0.3485594785242257
```

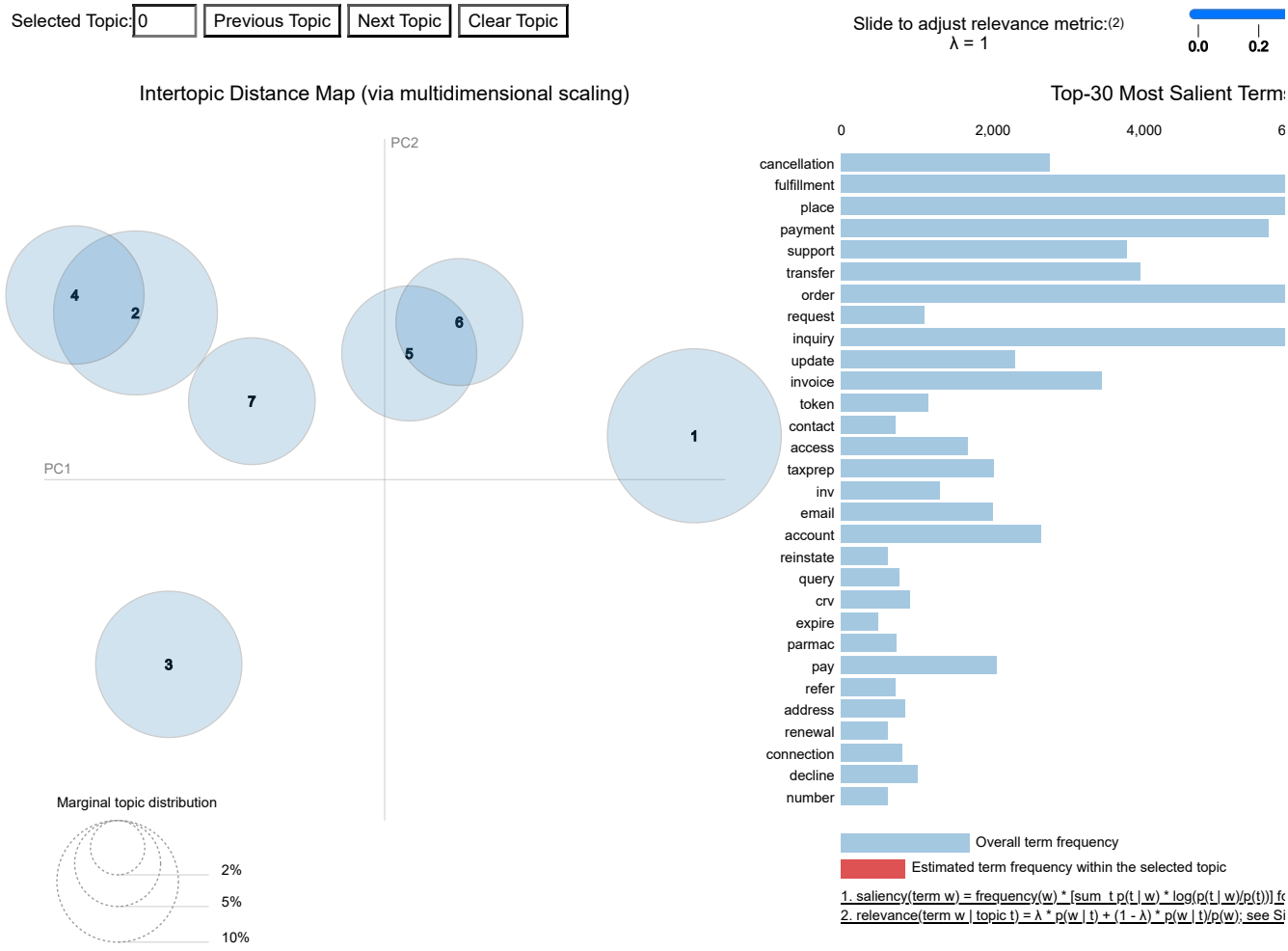
In [110]:
```
# Initializing LDA using tfidf approach to compute tfid weight of each word.
tfidf = TfidfModel(corpus)
corpus_tfidf = tfidf[corpus]
lda_model = gensim.models.LdaMulticore(corpus=corpus_tfidf, id2word=id2word, num_topics=num_topics, random_state=42)

if not os.path.exists('./results'):
    os.makedirs('./results')
```

In [111]:
```
# Printing topics in a text format for better readability for the tfidf approach.
for topic_idx, topic_terms in lda_model.show_topics(num_topics=num_topics, num_words=40, formatted=False):
    print("Topic #{}:".format(topic_idx))
    for term in topic_terms:
        print("  {} ({:.3f})".format(term[0], term[1]))
    print("\n")
```

```
Topic #0:
  payment (0.137)
  inquiry (0.057)
  order (0.041)
  fulfillment (0.027)
  place (0.026)
  pay (0.025)
  reinstate (0.021)
  invoice (0.018)
  download (0.017)
  renewal (0.016)
  account (0.014)
  transfer (0.014)
  cantax (0.013)
  decline (0.012)
  step (0.010)
  monthly (0.009)
  notify (0.009)
```

In [112]:
```
# Initializing pyLDAvis and defining the file path for the prepared data
pyLDAvis.enable_notebook()
LDAvis_data_filepath = os.path.join('./results/ldavis_prepared_'+str(num_topics))

# Visualizing the topic modeling results using pyLDAvis
if 1 == 1:
    LDAvis_prepared = pyLDAvis.gensim.prepare(lda_model, corpus, id2word)
    with open(LDAvis_data_filepath, 'wb') as f:
        pickle.dump(LDAvis_prepared, f)

# Saving the HTML visualization
with open(LDAvis_data_filepath, 'rb') as f:
    LDAvis_prepared = pickle.load(f)
pyLDAvis.save_html(LDAvis_prepared, './results/ldavis_prepared_'+ str(num_topics) +'lda.html')
LDAvis_prepared
```

Out[112]:



In [113]:
```
#Computing perplexity score
print('\nPerplexity: ', lda_model.log_perplexity(corpus_tfidf))

from gensim.models import CoherenceModel
# instantiate topic coherence model
cm = CoherenceModel(model=lda_model, corpus=corpus_tfidf, texts=texts, coherence='c_v')
# get topic coherence score
coherence_lda = cm.get_coherence()
print('\nCoherence: ', coherence_lda)
```

```
Perplexity:  -6.055344199642336

Coherence:  0.38721797477646663
```

In [114]:
```
#Saving output of the topic modelling in text format in output_csv file in the current working directory.
with open('output.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Source Org', 'Topic Number', 'Word Name', 'Weightage'])
    for topic_idx, topic_terms in lda_model.show_topics(num_topics=num_topics, num_words=40, formatted=False):
        for term in topic_terms:
            writer.writerow([source_org, topic_idx, term[0], term[1]])
```

In [115]:
```
# Using TAA Priority key mapping table as lookup table.
lookup_table = pd.read_excel('TAA Priority Key Word Mapping.xlsx')

#Lemmitizing words in the lookup table by leemitizing nouns, verbs and adjectives.
def lemmatize_lookup(text):
    return WordNetLemmatizer().lemmatize(text, pos='v')

def preprocess_lookup(df, text_column):
    result = []
    for text in df[text_column]:
        if isinstance(text, float):
            text = str(float(text)) # convert float to int, then to string
        processed_text = []
        for token in gensim.utils.simple_preprocess(text):
            if token not in gensim.parsing.preprocessing.STOPWORDS and len(token) >= 3:
                processed_text.append(lemmatize_lookup(token))
        result.append(' '.join(processed_text))
    df_processed = df.copy()
    df_processed[text_column] = result
    return df_processed
lookup_table = preprocess_lookup(lookup_table, 'Keyword')

def lemmatize_lookup(text):
    return WordNetLemmatizer().lemmatize(text, pos='n')
lookup_table = preprocess_lookup(lookup_table, 'Keyword')
def lemmatize_lookup(text):
    return WordNetLemmatizer().lemmatize(text, pos='a')
lookup_table = preprocess_lookup(lookup_table, 'Keyword')
```

In [116]:
```
#Doing vlookup of words in lookup table with words in the topic modelling output.csv
def lookup(keyword):
    result = lookup_table.loc[lookup_table['Keyword'] == keyword]
    if not result.empty:
        return result.iloc[0]['Issue Category'], result.iloc[0]['Contact Reason']
    else:
        return '***No_Match***', '***No_Match***'

with open('updated_output.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Source Org', 'Topic Number', 'Word Name', 'Weightage', 'Issue Category', 'Contact Reason'])

    for topic_idx, topic_terms in lda_model.show_topics(num_topics=num_topics, num_words=40, formatted=False):
        for term in topic_terms:
            issue_category, contact_reason = lookup(term[0])
            writer.writerow([source_org, topic_idx, term[0], term[1], issue_category, contact_reason])
```

In [117]:
```
#########################################################################################################################################
##The below code automatically generates sentences for each topic and gives the percentage contribution of a particular issue
#########################################################################################################################################
```

In [118]:

```
with open('updated_output.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Source Org', 'Topic Number', 'Word Name', 'Weightage', 'Issue Category', 'Contact Reason'])

    for topic_idx, topic_terms in lda_model.show_topics(num_topics=num_topics, num_words=40, formatted=False):
        no_match_count = 0
        admin_count = 0
        inquiry_product_support_count = 0
        orders_payments_count = 0
        sales_sub_ren_count = 0
        sme_inq_count = 0
        unclear_count = 0
        abandoned_count = 0
        redacted_count = 0
        for term in topic_terms:
            issue_category, contact_reason = lookup(term[0])
            if issue_category == '***No_Match***':
                no_match_count += 1
            elif issue_category == 'Admin':
                admin_count += 1
            elif issue_category == 'Orders and Payments':
                orders_payments_count += 1
            elif issue_category == 'Sales / Subscriptions / Renewals':
                sales_sub_ren_count += 1
            elif issue_category == 'SME Inquiries':
                sme_inq_count += 1
            elif issue_category == 'Unclear description':
                unclear_count += 1
            elif issue_category == 'Abandoned':
                abandoned_count += 1
            elif issue_category == 'Redacted':
                redacted_count += 1
            elif issue_category == 'Inquiries and Product Support':
                inquiry_product_support_count += 1
            writer.writerow([source_org, topic_idx, term[0], term[1], issue_category, contact_reason])

        total_count = no_match_count + admin_count + inquiry_product_support_count + orders_payments_count + sales_sub_ren_co
        no_match_percent = round((no_match_count / total_count) * 100, 2)
        admin_percent = round((admin_count / total_count) * 100, 2)
        inquiry_product_support_percent = round((inquiry_product_support_count / total_count) * 100, 2)
        orders_payments_percent = round((orders_payments_count / total_count) * 100, 2)
        sales_sub_ren_percent = round((sales_sub_ren_count / total_count) * 100, 2)
        sme_inq_percent = round((sme_inq_count / total_count) * 100, 2)
        unclear_percent = round((unclear_count / total_count) * 100, 2)
        abandoned_percent = round((abandoned_count / total_count) * 100, 2)
        redacted_percent = round((redacted_count / total_count) * 100, 2)


        sentence_parts = []
        if no_match_percent > 0:
            sentence_parts.append(f"{no_match_percent}% words which are **No Match**")
        if admin_percent > 0:
            sentence_parts.append(f"{admin_percent}% words which belong to Admin")
        if inquiry_product_support_percent > 0:
            sentence_parts.append(f"{inquiry_product_support_percent}% words which belong to Inquiry and Product Support")
        if orders_payments_percent > 0:
            sentence_parts.append(f"{orders_payments_percent}% words which belong to Orders and Payments")
        if sales_sub_ren_percent > 0:
            sentence_parts.append(f"{sales_sub_ren_percent}% words which belong to Sales Subscriptions and Renewals")
        if sme_inq_percent > 0:
            sentence_parts.append(f"{sme_inq_percent}% words which belong to SME Inquiries")
        if unclear_percent > 0:
            sentence_parts.append(f"{unclear_percent}% words which are Unclear")
        if abandoned_percent > 0:
            sentence_parts.append(f"{abandoned_percent}% words which are Abandoned")
        if redacted_percent > 0:
            sentence_parts.append(f"{redacted_percent}% words which are Redacted")

        sentence = f"Topic {topic_idx} has " + ", ".join(sentence_parts) + "."
        print(sentence)
```

```
Topic 0 has 57.5% words which are **No Match**, 7.5% words which belong to Admin, 17.5% words which belong to Inquiry and Product
and Payments, 5.0% words which belong to Sales Subscriptions and Renewals.
Topic 1 has 62.5% words which are **No Match**, 12.5% words which belong to Admin, 12.5% words which belong to Inquiry and Product
s and Payments, 2.5% words which belong to Sales Subscriptions and Renewals.
Topic 2 has 65.0% words which are **No Match**, 15.0% words which belong to Admin, 10.0% words which belong to Inquiry and Product
s and Payments.
Topic 3 has 62.5% words which are **No Match**, 10.0% words which belong to Admin, 15.0% words which belong to Inquiry and Product
s and Payments, 2.5% words which belong to SME Inquiries.
Topic 4 has 60.0% words which are **No Match**, 12.5% words which belong to Admin, 15.0% words which belong to Inquiry and Product
s and Payments.
Topic 5 has 47.5% words which are **No Match**, 22.5% words which belong to Admin, 17.5% words which belong to Inquiry and Product
s and Payments, 2.5% words which belong to SME Inquiries.
Topic 6 has 57.5% words which are **No Match**, 12.5% words which belong to Admin, 15.0% words which belong to Inquiry and Product
s and Payments, 2.5% words which belong to Sales Subscriptions and Renewals.
```

In [119]:

```
# The below code first removes the basic english stopwords and extended list of stopwords from the priority mapping key table
lookup_table.rename(columns={'Issue Category':'Topic Number'},inplace=True)
lookup_table=lookup_table.dropna()

stop_words = stopwords.words('english')
stop_words.extend(['wolters', 'kluwer', 'from', 'subject', 're', 'edu', 'use', 'fw', 'Fw', 'FWD', 'fwd', 'hi', 'may', 'know',
                   'regards', 'iphone', 'android', 'interested', 'australia', 'australian', 'wolterskluwer'])
```

In [120]:

```
# Performing pre-processing steps on the updated lookup table.
lookup=lookup_table[['Topic Number','Keyword']].drop_duplicates()
lookup=lookup.dropna()
lookup_sw=lookup[lookup['Keyword'].isin(stop_words)]
lookup=lookup[~lookup['Keyword'].isin(stop_words)] ###
lookup=lookup.sort_values(by='Keyword',ascending=True)###
lookup.head(10)
```

Out[120]:

	Topic Number	Keyword
129	Admin	
126	Orders and Payments	
22	Inquiries and Product Support	
173	SME Inquiries	
219	Abandoned	abandon
214	Abandoned	abondoned
169	Admin	accept
107	Admin	access
109	Admin	account
114	Admin	acct

In [121]:

```
# Removing empty keywords from the key mapping table
lookup=lookup[lookup['Keyword']!='']
```

In [123]:

```
# Creating a pivot table to represent each issue in terms of all the words n the data dictionary.
lookup['val']=1
lookup_pivot= lookup.pivot_table(index='Topic Number', columns='Keyword', values='val', fill_value=0) ###
lookup_pivot=lookup_pivot.reset_index()
lookup_pivot.head(10)
```

Out[123]:

Keyword	Topic Number	abandon	abondoned	accept	access	account	acct	ach	add	address	...	troubleshoot	try	unable	update	upload
0	Abandoned	1	1	0	0	0	0	0	0	0	...	0	0	0	0	0
1	Admin	0	0	1	1	1	1	0	1	1	...	0	0	0	0	0
2	Inquiries and Product Support	0	0	0	0	0	0	0	0	0	...	1	1	1	1	1
3	Orders and Payments	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0
4	Redacted	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
5	SME Inquiries	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
6	Sales / Subscriptions / Renewals	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
7	Unclear description	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0

8 rows × 159 columns

In [124]:
```
# Retaining issue categories where there was a match in the keywords in the vloop section above.
lda_output = pd.read_csv('updated_output.csv')
lda_output=lda_output[lda_output['Issue Category']!='***No_Match***']
lda_output=lda_output.sort_values(by='Word Name',ascending=True)
lda_output=lda_output[['Topic Number','Word Name']].drop_duplicates()

# Creating a pivot table to represent each topic in terms of all the words n the data dictionary.
lda_output['val']=1
lda_output_pivot = lda_output.pivot_table(index='Topic Number', columns='Word Name', values='val', fill_value=0)
lda_output_pivot=lda_output_pivot.reset_index()
lda_output_pivot.head(10)
```

Out[124]:

Word Name	Topic Number	access	account	add	address	archive	cch	change	contact	contract	...	renewal	request	reset	return	ship	subscription	sup
0	0	0	1	0	0	0	1	0	0	0	...	1	0	0	1	0	1	
1	1	0	1	1	1	1	0	0	0	0	...	1	0	0	0	0	0	
2	2	1	1	0	0	1	0	1	0	0	...	0	0	0	0	1	0	
3	3	0	1	0	1	0	0	0	1	0	...	0	0	0	0	0	0	
4	4	1	1	0	1	0	0	0	0	0	...	0	1	0	0	0	0	
5	5	1	1	0	0	1	1	1	0	0	...	0	0	1	0	0	0	
6	6	0	1	1	0	1	1	1	0	1	...	0	1	0	0	0	1	

7 rows × 41 columns

In [125]:
```
## The Unions of Keywords - adding missing columns to dataframe lookup_pivot and filling with 0s.
lookup_pivot=lookup_pivot.reindex(columns=lookup_pivot.columns.union(lda_output_pivot.columns), fill_value=0)
# Adding missing columns to dataframe lda_output_pivot and filling with 0s.
lda_output_pivot=lda_output_pivot.reindex(columns=lookup_pivot.columns.union(lda_output_pivot.columns), fill_value=0)
```

In [126]:
```
lookup_pivot
```

Out[126]:

	Topic Number	abandon	abondoned	accept	access	account	acct	ach	add	address	...	troubleshoot	try	unable	update	upload	uptoda
0	Abandoned	1	1	0	0	0	0	0	0	0	...	0	0	0	0	0	
1	Admin	0	0	1	1	1	1	0	1	1	...	0	0	0	0	0	
2	Inquiries and Product Support	0	0	0	0	0	0	0	0	0	...	1	1	1	1	1	
3	Orders and Payments	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0	
4	Redacted	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
5	SME Inquiries	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
6	Sales / Subscriptions / Renewals	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
7	Unclear description	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	

8 rows × 159 columns

In [127]:
```
lda_output_pivot
```

Out[127]:

	Topic Number	abandon	abondoned	accept	access	account	acct	ach	add	address	...	troubleshoot	try	unable	update	upload	uptodate	user	voice
0	0	0	0	0	0	1	0	0	0	0	...	0	0	0	1	0	0	0	
1	1	0	0	0	0	1	0	0	1	1	...	0	0	0	1	0	0	0	
2	2	0	0	0	1	1	0	0	0	0	...	0	0	0	1	0	0	0	
3	3	0	0	0	0	1	0	0	0	1	...	0	0	0	1	0	0	0	
4	4	0	0	0	1	1	0	0	0	1	...	0	0	0	1	0	0	0	
5	5	0	0	0	1	1	0	0	0	0	...	0	0	0	1	0	0	1	
6	6	0	0	0	0	1	0	0	1	0	...	0	0	0	1	0	0	0	

7 rows × 159 columns

In [128]:
```
## It can be seen from the above two pivot tables, that the number of columns is the same.
```

In [130]:
```
# Creating a similarity matrix using Jacard Measure to find relevance between topics and issue categories.
similarity_ps = 1 - pairwise_distances(lookup_pivot.values, lda_output_pivot.values, metric='jaccard')

similarity_df = pd.DataFrame(similarity_ps,index=lookup_pivot.index, columns=lda_output_pivot.index)

similarity_df.index.name = 'issue category'
similarity_df.columns.name = 'Topic Number'

category_dict = {0: 'Abandoned', 1: 'Admin', 2: 'Inquiries and Product Support', 3: 'Orders and Payments', 4: 'Redacted', 5:

similarity_df.index = similarity_df.index.to_series().replace(category_dict)

similarity_df
```

Out[130]:

Topic Number	0	1	2	3	4	5	6
issue category							
Abandoned	0.000000	0.052632	0.055556	0.052632	0.050000	0.040000	0.047619
Admin	0.051724	0.111111	0.134615	0.090909	0.109091	0.178571	0.107143
Inquiries and Product Support	0.074468	0.063830	0.053191	0.075269	0.074468	0.081633	0.073684
Orders and Payments	0.178571	0.185185	0.192308	0.185185	0.222222	0.151515	0.214286
Redacted	0.000000	0.058824	0.062500	0.058824	0.055556	0.043478	0.052632
SME Inquiries	0.000000	0.038462	0.040000	0.080000	0.037037	0.064516	0.035714
Sales / Subscriptions / Renewals	0.095238	0.100000	0.050000	0.047619	0.045455	0.037037	0.090909
Unclear description	0.000000	0.058824	0.062500	0.058824	0.055556	0.043478	0.052632

In [131]:
```
# The below code automatically prints the topic number distribution in each issue category based on the top 3 similarity scor
for index, row in similarity_df.iterrows():
    sorted_topics = row.sort_values(ascending=False).index.values
    top_topic = sorted_topics[0]
    second_topic = sorted_topics[1]
    third_topic = sorted_topics[2]
    print(f"The issue category '{index}' is most similar to the topic number '{top_topic}' '{second_topic}' and '{third_topic
```

```
The issue category 'Abandoned' is most similar to the topic number '2' '1' and '3'.

The issue category 'Admin' is most similar to the topic number '5' '2' and '1'.

The issue category 'Inquiries and Product Support' is most similar to the topic number '5' '3' and '0'.

The issue category 'Orders and Payments' is most similar to the topic number '4' '6' and '2'.

The issue category 'Redacted' is most similar to the topic number '2' '1' and '3'.

The issue category 'SME Inquiries' is most similar to the topic number '3' '5' and '2'.

The issue category 'Sales / Subscriptions / Renewals' is most similar to the topic number '1' '0' and '6'.

The issue category 'Unclear description' is most similar to the topic number '2' '1' and '3'.
```

In [132]:
```
# The below code automatically prints the issue category distribution in each topic based on the top 3 similarity scores.
for col in similarity_df.columns:
    sorted_categories = similarity_df[col].sort_values(ascending=False).index.values
    top_category = sorted_categories[0]
    second_category = sorted_categories[1]
    third_category = sorted_categories[2]
    print(f"The topic number '{col}' is most similar to the issue category '{top_category}' '{second_category}' and '{third_c
```

```
The topic number '0' is most similar to the issue category 'Orders and Payments' 'Sales / Subscriptions / Renewals' and 'Inquiries

The topic number '1' is most similar to the issue category 'Orders and Payments' 'Admin' and 'Sales / Subscriptions / Renewals'.

The topic number '2' is most similar to the issue category 'Orders and Payments' 'Admin' and 'Redacted'.

The topic number '3' is most similar to the issue category 'Orders and Payments' 'Admin' and 'SME Inquiries'.

The topic number '4' is most similar to the issue category 'Orders and Payments' 'Admin' and 'Inquiries and Product Support'.

The topic number '5' is most similar to the issue category 'Admin' 'Orders and Payments' and 'Inquiries and Product Support'.

The topic number '6' is most similar to the issue category 'Orders and Payments' 'Admin' and 'Sales / Subscriptions / Renewals'.
```

In [133]:
```
########################################################################## END OF CODE ##########################################
```