Use case 1: GenAI

The accompanying file titled transcripts_v3.zip contains examples of call transcripts with both the agent and customer transcripts being provided.

Question 1:

```
Use a large language model of your choice to analyse the customer side
of the transcript only and:

    -Identify the sentiment (positive, negative, neutral) of the call

    -Determine call outcome (issue resolved, follow-up action needed)
```

Question 2:

```
-Use appropriate metrics to monitor the performance of your model.
```

Question 3:

```
-Use methods of your choice (e.g. exploratory data analysis,
statistical methods, visualisations etc.)  to extract useful insights
from the data.
```

In [1]:

```
import os
import zipfile
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
import openai
from openai import OpenAI
from transformers import pipeline
from wordcloud import WordCloud
from sklearn.feature_extraction.text import CountVectorizer

tqdm.pandas()

zip_path = "transcripts_v3.zip"
extract_path = "transcripts_data"

if not os.path.exists(extract_path):
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(extract_path)

print(f"Data extracted to {extract_path}")
```

```
Data extracted to transcripts_data
```

In [2]:

```
folder_path = "transcripts_data/transcripts_v3"
customer_texts = []
for file_name in os.listdir(folder_path):
    file_path = os.path.join(folder_path, file_name)

    if file_path.endswith(".txt"):  # Process only .txt files
```

```
        with open(file_path, 'r') as file:
            lines = file.readlines()
            # Extract lines where the speaker is 'Member'
            customer_lines = [line.split(":", 1)[1].strip() for line in lines if line
            # Combine customer lines into a single text
            customer_text = " ".join(customer_lines)
            customer_texts.append({"file_name": file_name, "customer_text": customer_

print(f"Processed {len(customer_texts)} files.")
```

```
Processed 200 files.
```

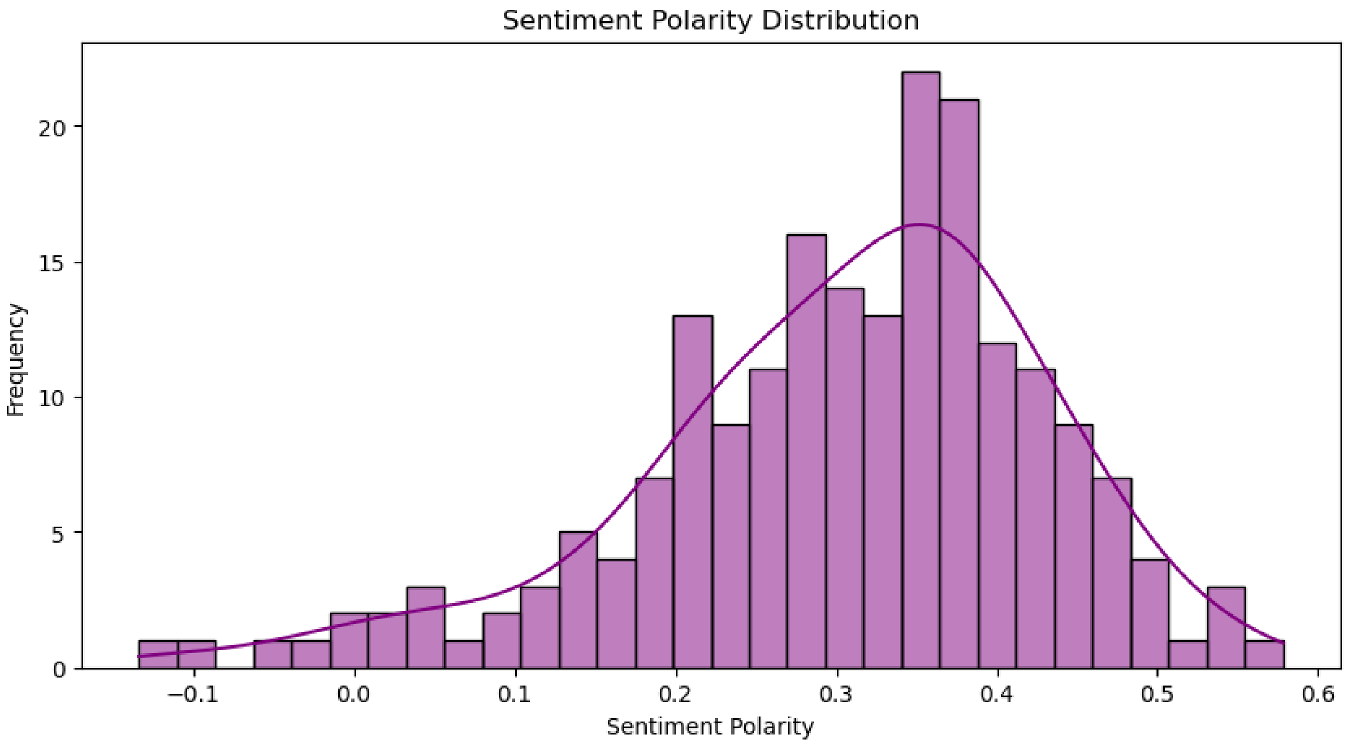
In [3]:
```
df = pd.DataFrame(customer_texts)
df.head()
```

Out[3]:

	file_name	customer_text
0	transcript_38.txt	Hi, I'm calling to get a case pre-authorized f...
1	transcript_10.txt	Hi, I'm calling to schedule an appointment wit...
2	transcript_138.txt	Hi, I'm calling to get a case pre-authorized f...
3	transcript_104.txt	Hi, I'm calling to get a case pre-authorized. ...
4	transcript_110.txt	Hi, I'm calling about a denied claim I receive...

Sentiment Distribution (Primary Analysis)

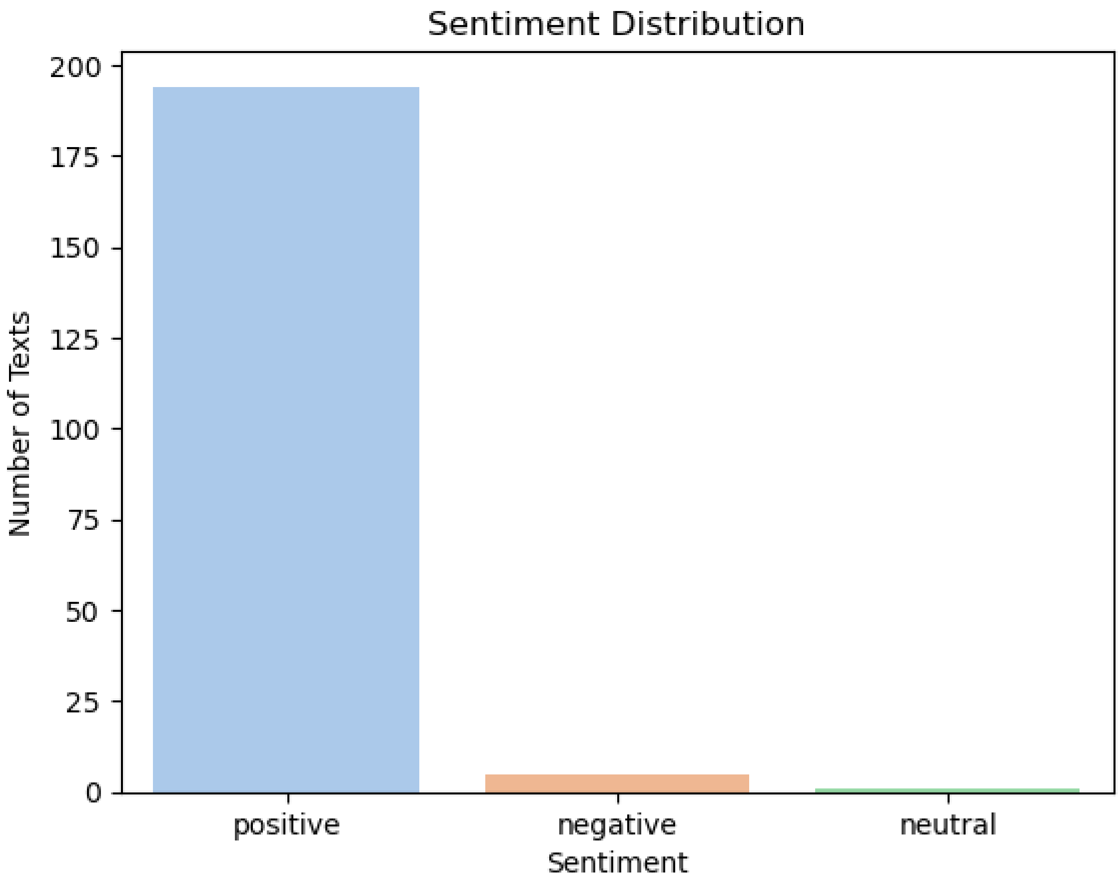
In [4]:
```
from textblob import TextBlob
df['sentiment'] = df['customer_text'].apply(lambda x: TextBlob(x).sentiment.polarity)

plt.figure(figsize=(10, 5))
sns.histplot(df['sentiment'], kde=True, bins=30, color="purple")
plt.title("Sentiment Polarity Distribution")
plt.xlabel("Sentiment Polarity")
plt.ylabel("Frequency")
plt.show()
```



Basic Sentiment Classification

In [5]:
```python
from textblob import TextBlob
import seaborn as sns
import matplotlib.pyplot as plt

def classify_sentiment(text):
    polarity = TextBlob(text).sentiment.polarity
    if polarity > 0:
        return "positive"
    elif polarity < 0:
        return "negative"
    else:
        return "neutral"

df['sentiment'] = df['customer_text'].apply(classify_sentiment)

print(df['sentiment'].value_counts())

sns.countplot(data=df, x='sentiment', palette='pastel')
plt.title("Sentiment Distribution")
plt.xlabel("Sentiment")
plt.ylabel("Number of Texts")
plt.show()
```

```
positive    194
negative      5
neutral       1
Name: sentiment, dtype: int64
```



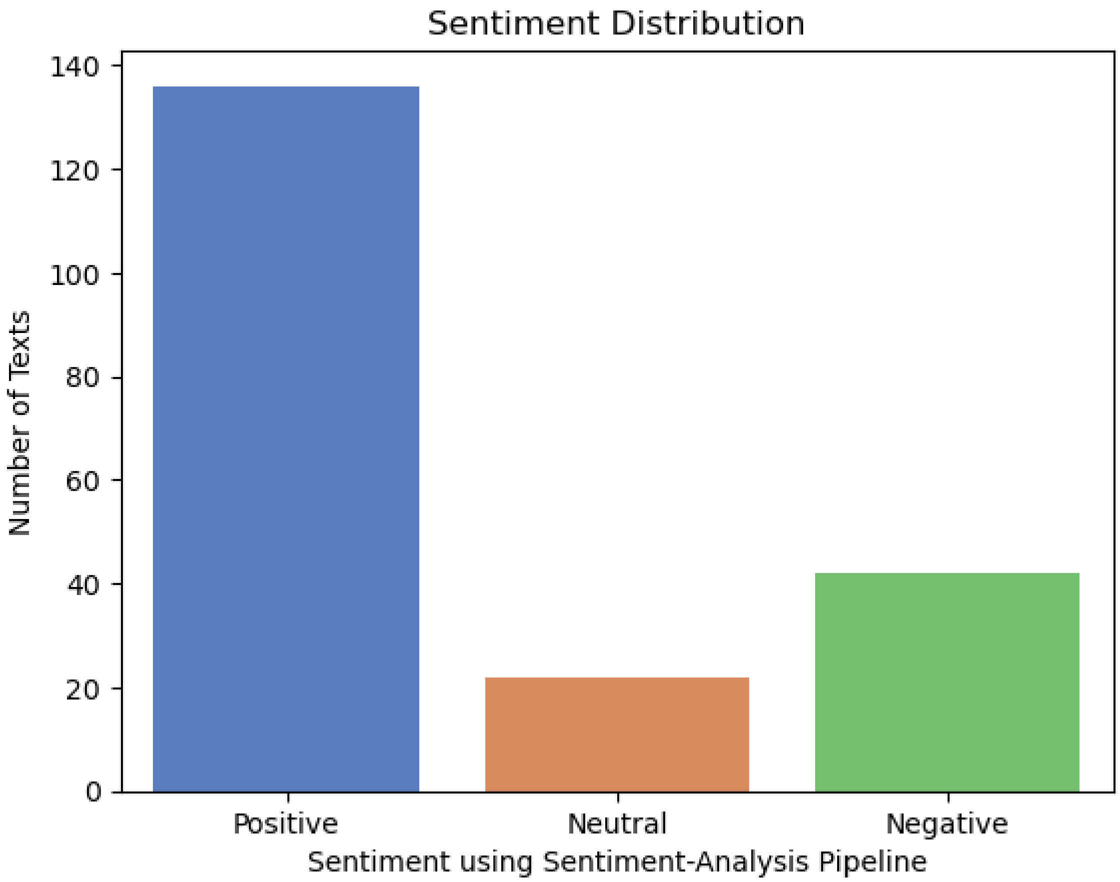
In [6]:
```python
df_check=df
```

Extended Sentiment Classification - using transformer package : 'roberta-base-sentiment'

In [7]:
```python
from transformers import pipeline

# Loaded pre-trained roberta sentiment analysis pipeline
sentiment_analyzer1 = pipeline("sentiment-analysis",model="cardiffnlp/twitter-roberta
```

```python
# Classify sentiment for each text
df['sentiment1'] = df['customer_text'].apply(lambda x: sentiment_analyzer1(x)[0]['lab

# # Display sentiment distribution
# print(df['sentiment1'].value_counts())


# Mapping dictionary for sentiment labels
sentiment_mapping = {
    "LABEL_0": "Negative",
    "LABEL_1": "Neutral",
    "LABEL_2": "Positive"
}

# Replace labels in the 'sentiment1' column
df['sentiment1'] = df['sentiment1'].replace(sentiment_mapping)

# Verify the changes
print(df['sentiment1'].value_counts())

# Visualize sentiment distribution
sns.countplot(data=df, x='sentiment1', palette='muted')
plt.title("Sentiment Distribution")
plt.xlabel("Sentiment using Sentiment-Analysis Pipeline")
plt.ylabel("Number of Texts")
plt.show()
```

```
Xformers is not installed correctly. If you want to use memory_efficient_attention to
accelerate training use the following command to install Xformers
pip install xformers.
Positive    136
Negative     42
Neutral      22
Name: sentiment1, dtype: int64
```



Using OPENAI for Outcome Determination

In [8]:

```python
# Initialize the OpenAI client
client = OpenAI(
    api_key="sk-proj-0GchwbS6onUdrRwrfooz8YAzVhkB5hK9e2yPaP3FhAj0FYI6Z2bQH8wZBqOVXsjt

def determine_outcome(text):
    """
    Determines the call outcome as either 'issue resolved' or 'follow-up action neede
    """
    try:
        messages = [
            {
                "role": "system",
                "content": "You are a helpful assistant that determines the outcome o
            },
            {
                "role": "user",
                "content": f"Based on the following conversation, determine if the is
            }
        ]
        response = client.chat.completions.create(
            model="gpt-3.5-turbo",
            messages=messages,
            temperature=0
        )
        response_message = response.choices[0].message.content.strip()
        return response_message
    except Exception as e:
        print(f"Error in outcome determination: {e}")
        return "Error"


# Enable progress bar for pandas
tqdm.pandas()

# Load the customer text data
df = pd.read_csv("customer_texts.csv")
df['customer_text'] = df['customer_text'].fillna('')

# Apply outcome determination with tqdm
df['determine_outcome'] = [
    determine_outcome(text) for text in tqdm(df['customer_text'], desc="Determining O
]

df_outcome=df

## Determine call outcome (issue resolved, follow-up action needed)

# Load the zero-shot classification pipeline
zero_shot_classifier = pipeline("zero-shot-classification", model="facebook/bart-larg

# Define possible outcomes
labels = ["issue resolved", "follow-up action needed"]

# Enable tqdm for pandas
tqdm.pandas()

df_outcome['determine_outcome'] = df_outcome['determine_outcome'].fillna('')  # Ensur

df_outcome['call_outcome'] = df_outcome['determine_outcome'].progress_apply(
    lambda x: zero_shot_classifier(x, candidate_labels=labels)['labels'][0]
)

print(df_outcome['call_outcome'].value_counts())
```

```
df_outcome=df_outcome[['file_name', 'customer_text','determine_outcome', 'call_outcom
df_outcome
```

```
huggingface/tokenizers: The current process just got forked, after parallelism has alr
eady been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
	- Avoid using `tokenizers` before the fork if possible
	- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | fals
e)
Determining Outcome: 100%|████████████████████| 200/200 [02:55<00:00,  1.14it/s]
100%|█████████████████████████████████████████| 200/200 [07:27<00:00,  2.24s/it]
issue resolved           163
follow-up action needed   37
Name: call_outcome, dtype: int64
```

Out[8]:

	file_name	customer_text	determine_outcome	call_outcome
0	transcript_38.txt	Hi, I'm calling to get a case pre-authorized f...	Based on the conversation provided, it seems l...	issue resolved
1	transcript_10.txt	Hi, I'm calling to schedule an appointment wit...	Based on the conversation provided, the issue ...	issue resolved
2	transcript_138.txt	Hi, I'm calling to get a case pre-authorized f...	Based on the conversation provided, it seems t...	issue resolved
3	transcript_104.txt	Hi, I'm calling to get a case pre-authorized. ...	Based on the conversation provided, it seems l...	issue resolved
4	transcript_110.txt	Hi, I'm calling about a denied claim I receive...	Based on the conversation provided, it seems t...	follow-up action needed
...	...	...	...	...
195	transcript_135.txt	Hi, I'm calling about a denied claim I receive...	Based on the conversation provided, it seems l...	issue resolved
196	transcript_121.txt	Hi, I'm calling to get a case pre-authorized f...	Based on the conversation provided, it seems l...	issue resolved
197	transcript_0.txt	Hi, I'm calling to get a case pre-authorized. ...	Based on the conversation provided, it appears...	issue resolved
198	transcript_35.txt	Hi, I'm having trouble logging in to my online...	Based on the conversation provided, it seems t...	follow-up action needed
199	transcript_21.txt	Hi, I'm calling about issues I'm having with m...	Based on the conversation provided, it seems t...	follow-up action needed

200 rows × 4 columns

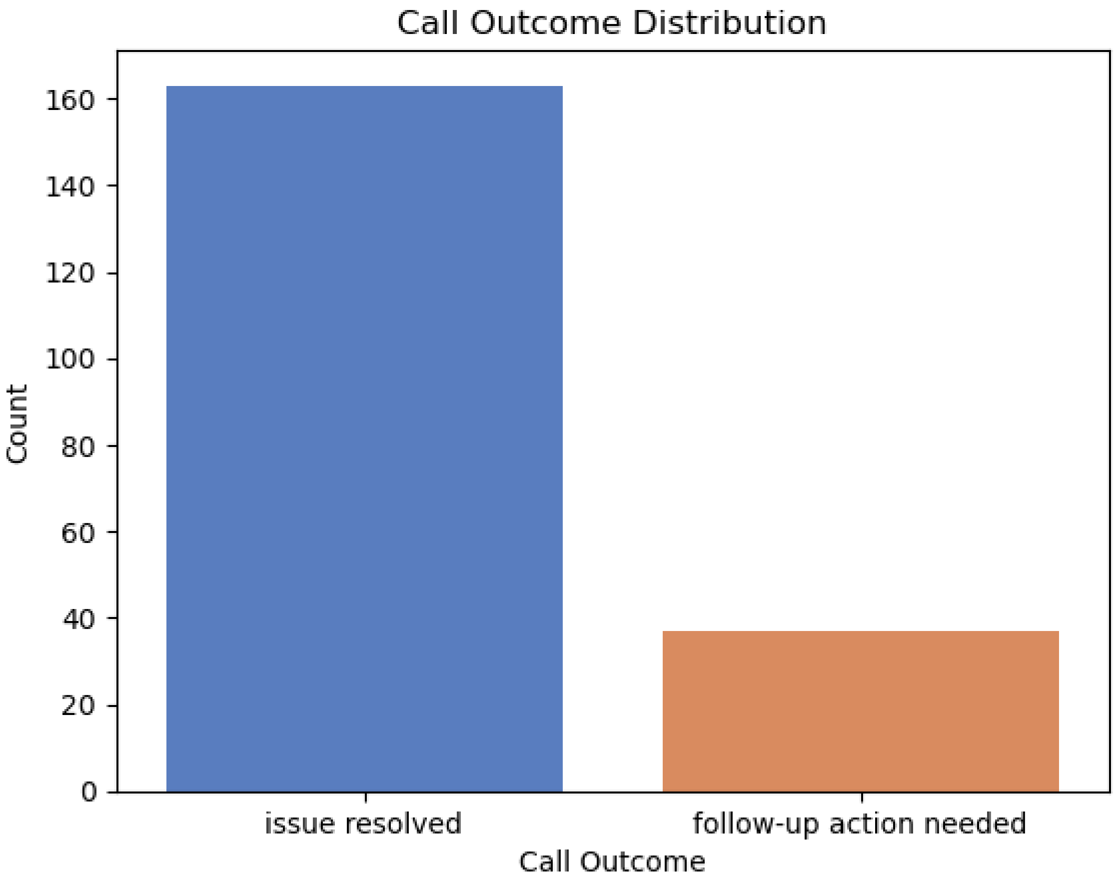
Performance monitoring:

- Monitoring the performance of an unsupervised model, especially one using a zero-shot classification pipeline, can be challenging due to the absence of ground truth labels.
- However, there are techniques and strategies to evaluate and monitor the model's performance:

1. Compare Output Distribution Against Expectations

In [9]:

```
sns.countplot(data=df_outcome, x='call_outcome', palette='muted')
plt.title("Call Outcome Distribution")
plt.xlabel("Call Outcome")
plt.ylabel("Count")
plt.show()
```



Outcome: The ouput looks quite legitimate as 3/4th of the result has been resolved whole 1/4th of them needs to be followed up. This is quite tantamount to company standards as anything more than 25% of issues, if that needed to be followed up shall bring broader questions as how the agents are performing as they are unable to solve the issues of the clients.

In [ ]:

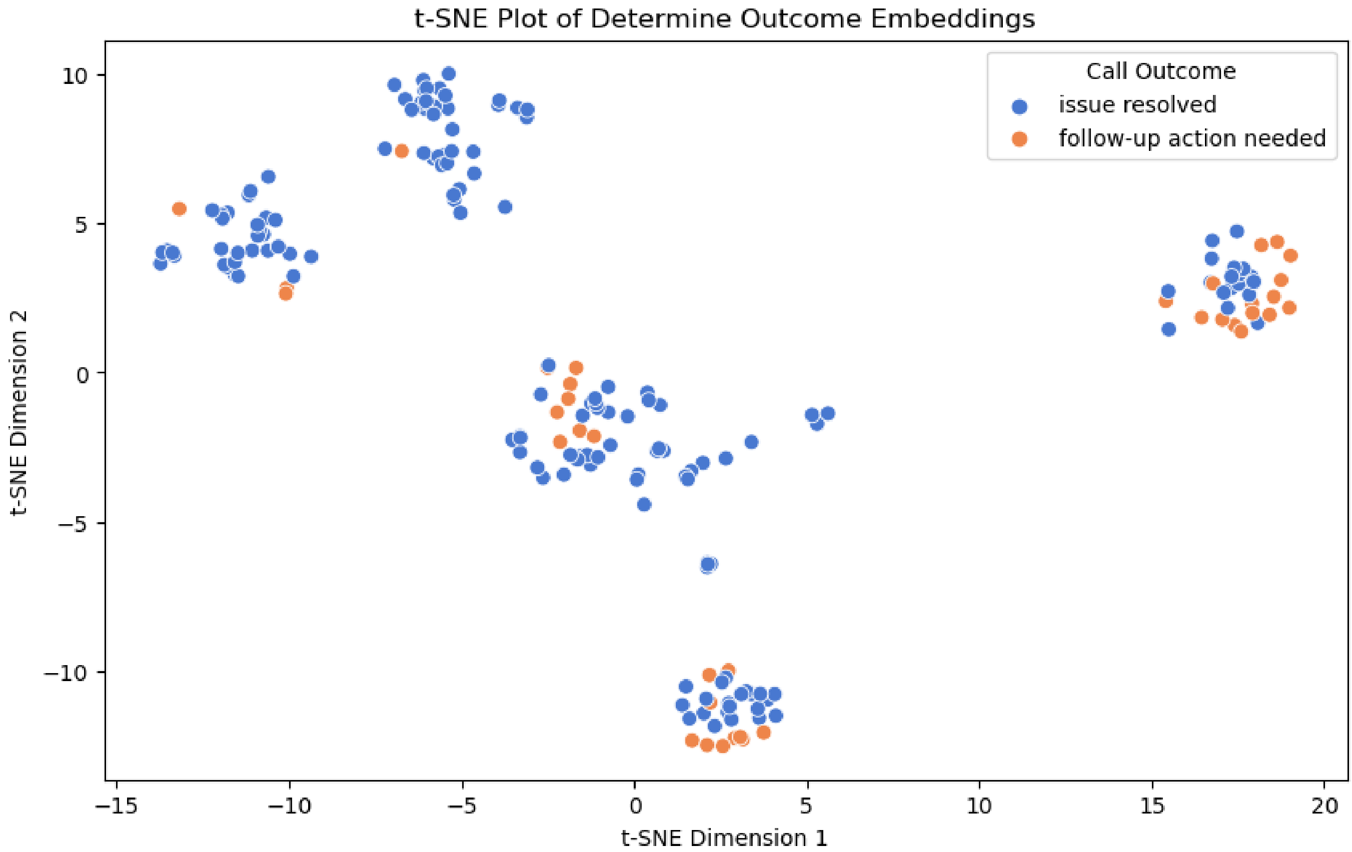
```

```

In [10]:

```
from sklearn.manifold import TSNE
from sentence_transformers import SentenceTransformer
import seaborn as sns
import matplotlib.pyplot as plt

# Initialize the sentence transformer model
embedding_model = SentenceTransformer('all-MiniLM-L6-v2')

# Generate embeddings for the determine_outcome column
determine_outcome_texts = df_outcome['determine_outcome'].tolist()
embeddings = embedding_model.encode(determine_outcome_texts)

# Apply t-SNE for dimensionality reduction
tsne = TSNE(n_components=2, random_state=42, perplexity=30, n_iter=1000, metric='cosi
reduced_embeddings = tsne.fit_transform(embeddings)

# Add t-SNE results to the DataFrame
df_outcome['tsne_x'] = reduced_embeddings[:, 0]
df_outcome['tsne_y'] = reduced_embeddings[:, 1]

# Plot t-SNE results with call_outcome as hue
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_outcome, x='tsne_x', y='tsne_y', hue='call_outcome', palette=
plt.title("t-SNE Plot of Determine Outcome Embeddings")
plt.xlabel("t-SNE Dimension 1")
plt.ylabel("t-SNE Dimension 2")
```

```
plt.legend(title="Call Outcome", loc='best')
plt.show()
```

```
/Users/soumyadasgupta/anaconda3/lib/python3.11/site-packages/sklearn/manifold/_t_sne.p
y:1162: FutureWarning: 'n_iter' was renamed to 'max_iter' in version 1.5 and will be r
emoved in 1.7.
  warnings.warn(
```



In [11]:
```
from sklearn.metrics import silhouette_score

# Calculate silhouette score
score = silhouette_score(embeddings, df_outcome['call_outcome'].map({'issue resolved'
                                                                     'follow-up actio
print(f"Silhouette Score: {score}")
```

```
Silhouette Score: 0.0369996577501297
```

- Silhouette Score: 0.033 - Range: The silhouette score ranges from -1 to 1.

  1 indicates that the samples are well clustered and distinct from other clusters. 0 indicates that the samples are on or near the decision boundary between clusters. -1 indicates that the samples are incorrectly clustered.

  A score close to 0 (like 0.033) suggests that the points are not well separated into distinct clusters. The clusters might be overlapping, or the distinction between the two categories is weak. In this case, the silhouette score indicates that the clusters for issue resolved and follow-up action needed are not well-formed, meaning the model is having difficulty distinguishing between these two outcomes. This can imply that: The features used to generate the embeddings (in this case, the determine_outcome text) are not distinct enough to separate the categories effectively. The clustering model may be producing weak clusters due to overlapping characteristics between the two outcomes.

In [12]:
```
from sklearn.cluster import KMeans
import numpy as np

# Apply KMeans clustering
```

```
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans_labels = kmeans.fit_predict(embeddings)

# Calculate cluster purity
def cluster_purity(labels, predicted_labels):
    return np.mean(labels == predicted_labels)

purity = cluster_purity(df_outcome['call_outcome'].map({'issue resolved': 1, 'follow-
print(f"Cluster Purity: {purity}")
```

```
Cluster Purity: 0.79
```

```
    - Cluster Purity (0.73):

    Problem: While 73% purity is decent, there’s still room for
    improvement. It means that the model's classification isn't perfect,
    and some instances are misclassified.
    Action: This could be improved by incorporating additional features,
    more diverse training data, or exploring different clustering
    techniques.
```

In [ ]:

```

```

EDA

In [13]:

```
df = pd.read_csv("customer_texts.csv")
df.info();
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   file_name      200 non-null    object
 1   customer_text  200 non-null    object
dtypes: object(2)
memory usage: 3.3+ KB
```

Descriptive Statistics

In [14]:

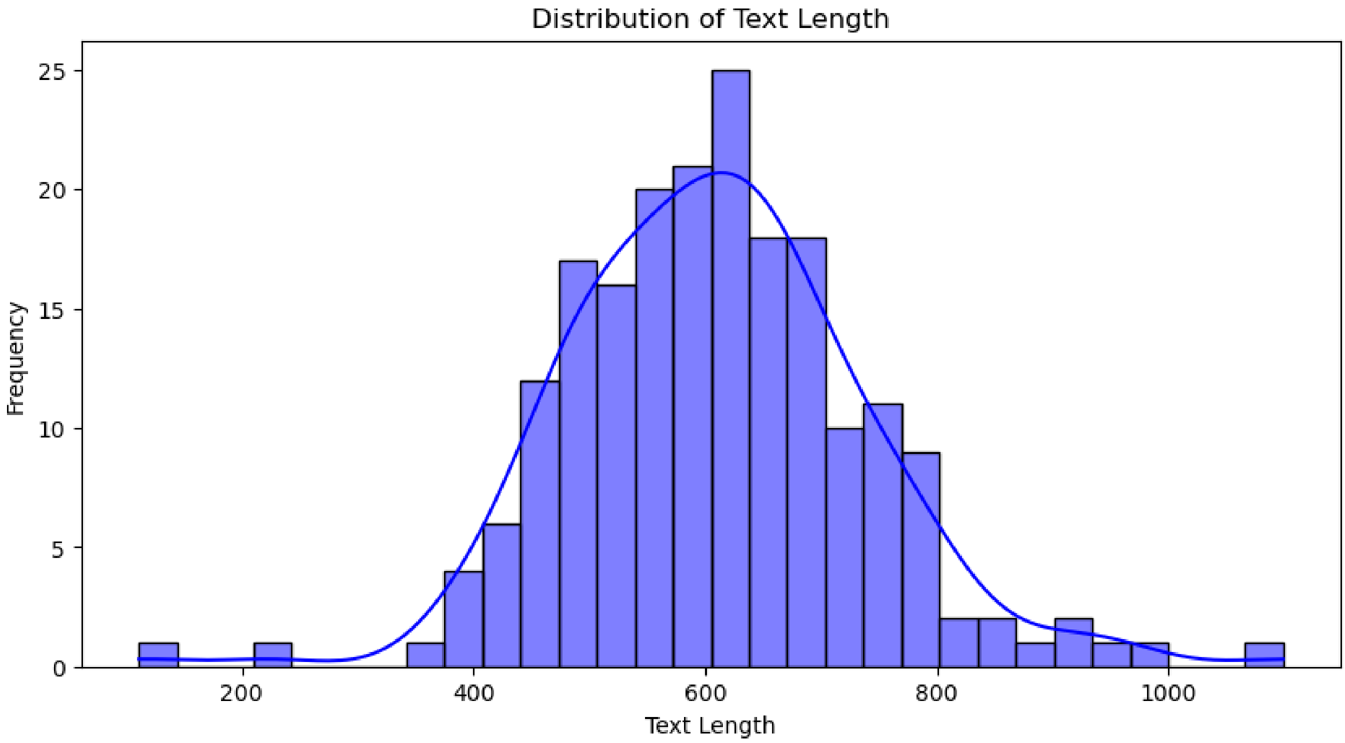
```
# Length of each customer text
df['text_length'] = df['customer_text'].apply(len)

# Number of words in each customer text
df['word_count'] = df['customer_text'].apply(lambda x: len(x.split()))

# Display summary statistics
print(df[['text_length', 'word_count']].describe())
```

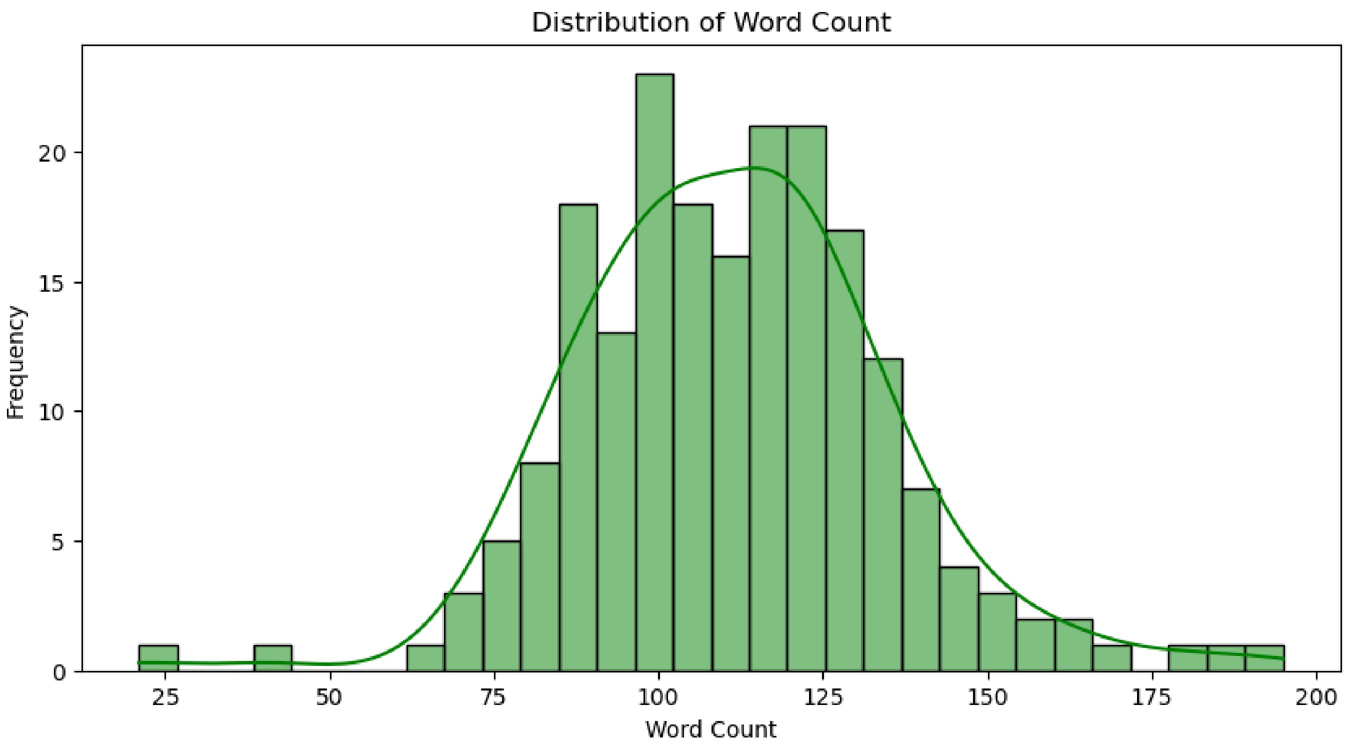
```
       text_length  word_count
count   200.000000  200.000000
mean    609.260000  111.435000
std     128.548223   23.542238
min     111.000000   21.000000
25%     515.500000   96.500000
50%     606.500000  111.000000
75%     684.000000  126.000000
max    1099.000000  195.000000
```

In [15]:

```
# Plot text length distribution
plt.figure(figsize=(10, 5))
sns.histplot(df['text_length'], kde=True, bins=30, color="blue")
```

```
plt.title("Distribution of Text Length")
plt.xlabel("Text Length")
plt.ylabel("Frequency")
plt.show()

# Plot word count distribution
plt.figure(figsize=(10, 5))
sns.histplot(df['word_count'], kde=True, bins=30, color="green")
plt.title("Distribution of Word Count")
plt.xlabel("Word Count")
plt.ylabel("Frequency")
plt.show()
```





In [16]:

```
from wordcloud import WordCloud

# Combine all customer texts into one large string
all_text = " ".join(df['customer_text'])

# Generate a word cloud
wordcloud = WordCloud(width=800, height=400, background_color="white").generate(all_t
```

```python
# Display the word cloud
plt.figure(figsize=(20, 15))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.title("Word Cloud of Customer Texts")
plt.show()
```

Identifying Common Issues (Keyword Extraction)

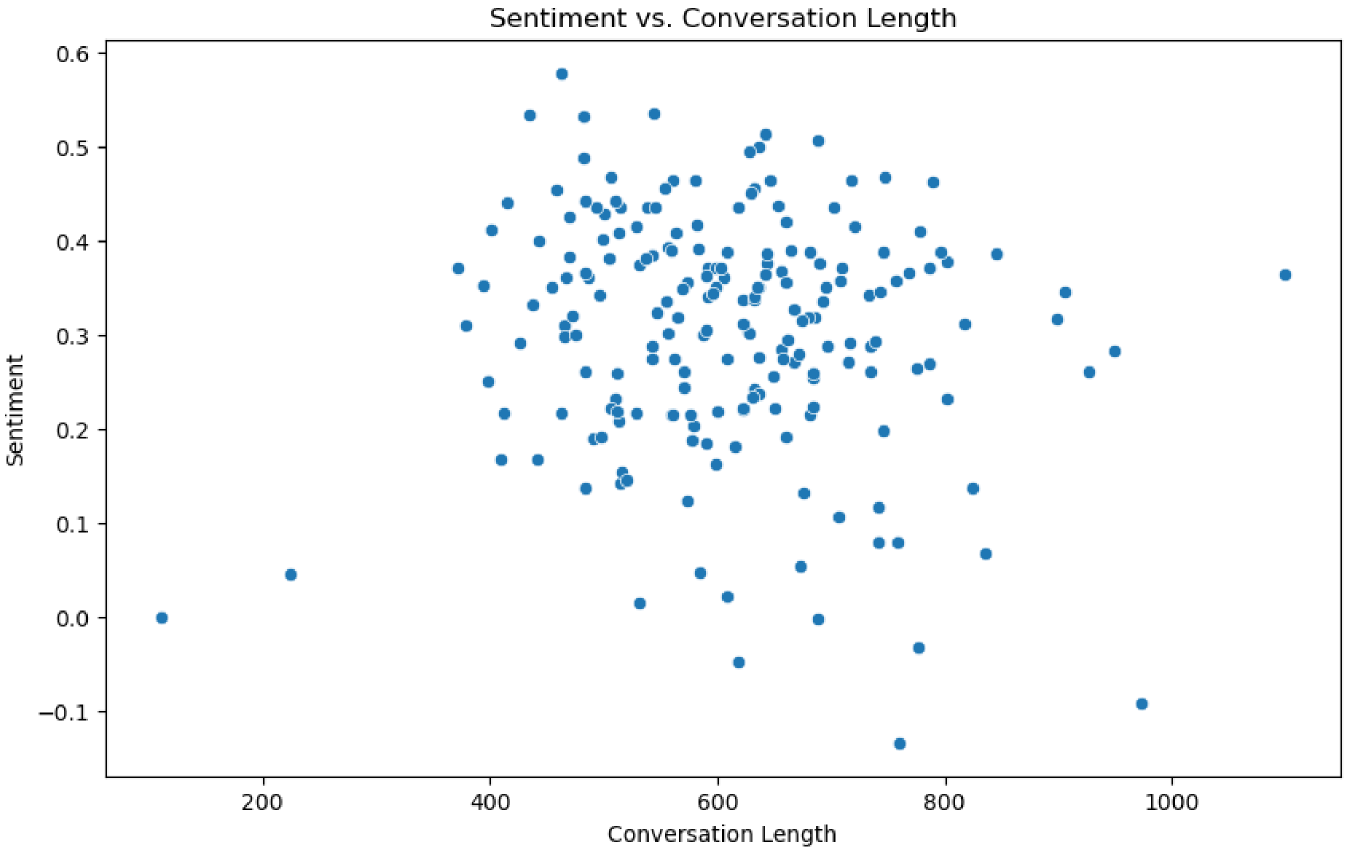
In [17]:
```python
# Using CountVectorizer to find frequently occurring words (excluding stopwords)
vectorizer = CountVectorizer(stop_words='english', max_features=10)
X = vectorizer.fit_transform(df['customer_text'])

# Convert the result to a DataFrame and display the top words
word_freq = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names_out())
word_freq = word_freq.sum(axis=0).sort_values(ascending=False)
word_freq.head(10)
```

Out[17]:
```
thank      467
okay       341
help       240
hi         199
policy     185
sounds     185
ve         173
service    171
id         163
member     163
dtype: int64
```

- CountVectorizer is used to extract and count words, excluding common stopwords like "the", "and", etc.
- The most frequent words give insights into the primary topics and concerns of customers. This helps in identifying recurring issues, which could be valuable for improving customer service.

In [18]:
```python
from textblob import TextBlob

# Function to calculate sentiment polarity
def get_sentiment(text):
```

```python
    return TextBlob(text).sentiment.polarity

# Apply sentiment analysis
df['sentiment'] = df['customer_text'].apply(get_sentiment)

# Plot sentiment vs. text length
plt.figure(figsize=(10, 6))
sns.scatterplot(x=df['text_length'], y=df['sentiment'])
plt.title('Sentiment vs. Conversation Length')
plt.xlabel('Conversation Length')
plt.ylabel('Sentiment')
plt.show()
```



Explanation:

    -Sentiment analysis helps understand customer emotions and attitudes
    towards the service. A negative sentiment might indicate frustration,
    while a positive sentiment indicates satisfaction.

    -This information is critical in identifying areas for improvement in
    customer service or product offerings.

In [19]:
```python
# Search for key terms like "policy", "claim", "doctor", etc.
keywords = ['policy', 'claim', 'doctor', 'procedure', 'authorization', 'surgery', 'th
keyword_counts = {}

for word in keywords:
    keyword_counts[word] = df['customer_text'].str.contains(word, case=False).sum()

# Display keyword counts
pd.DataFrame.from_dict(keyword_counts, orient='index', columns=['Count'])
```

Out[19]:

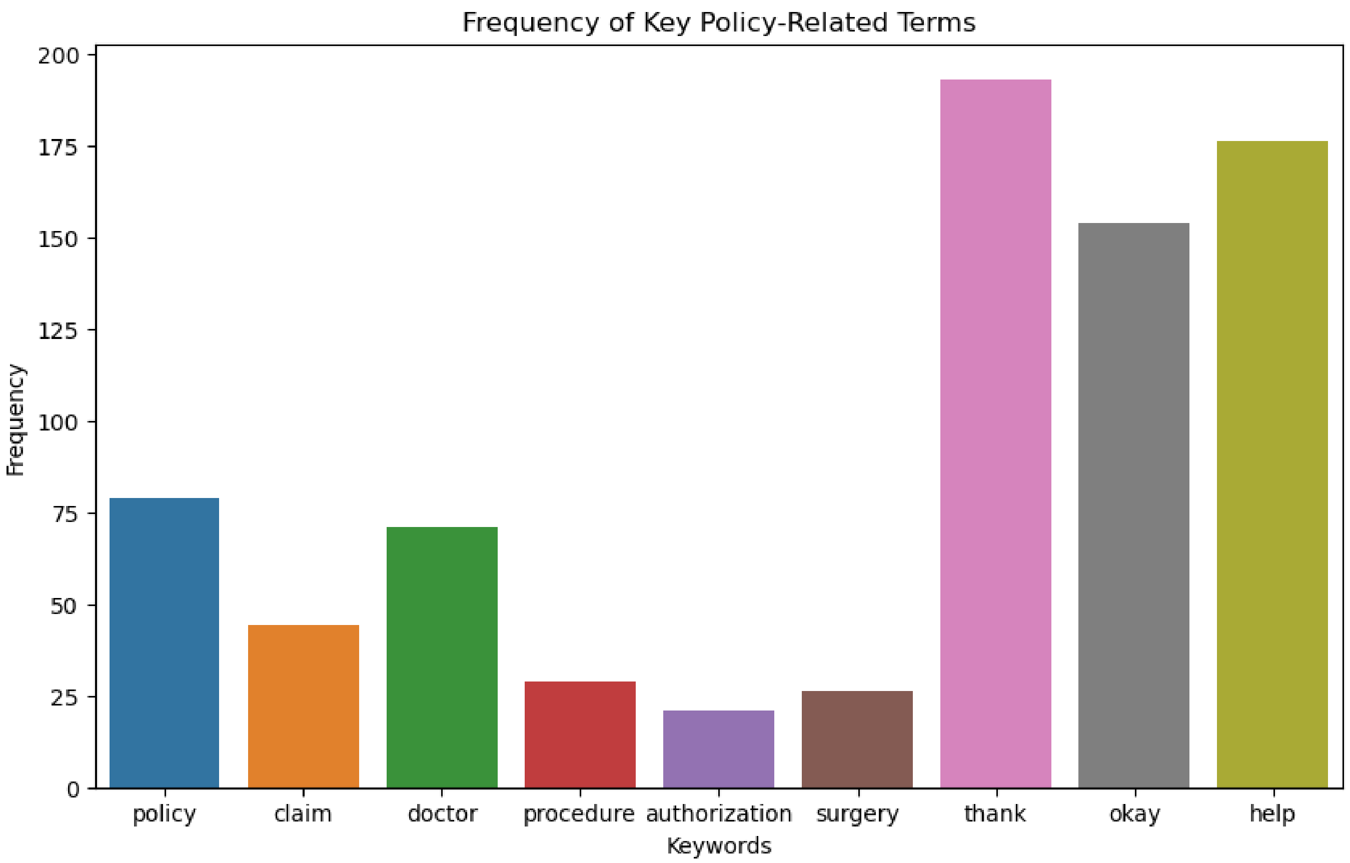
	Count
policy	79
claim	44
doctor	71
procedure	29
authorization	21
surgery	26
thank	193
okay	154
help	176

Explanation:

```
-Identifying references to specific terms like “policy”, “claim”, and
“doctor” will help the company understand which issues are most common
in customer conversations.

-Tracking the frequency of these terms can guide the company in
focusing on improving certain areas (e.g., claims processing or policy
coverage).
```

In [20]:

```python
# Bar plot of keyword counts
plt.figure(figsize=(10, 6))
sns.barplot(x=list(keyword_counts.keys()), y=list(keyword_counts.values()))
plt.title('Frequency of Key Policy-Related Terms')
plt.xlabel('Keywords')
plt.ylabel('Frequency')
plt.show()
```



Explanation:

-A bar plot provides a clear visualization of the most common issues that customers are facing based on the occurrence of specific keywords.
-This allows for quick identification of areas requiring attention, such as a high number of policy-related issues.

In [21]:
```
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import TfidfVectorizer

# Using TF-IDF Vectorizer for topic modeling
vectorizer = TfidfVectorizer(stop_words='english', max_features=1000)
X = vectorizer.fit_transform(df['customer_text'])

# Latent Dirichlet Allocation (LDA) for topic modeling
lda = LatentDirichletAllocation(n_components=15, random_state=42)
lda.fit(X)

# Display the top words for each topic
for idx, topic in enumerate(lda.components_):
    print(f"Topic {idx + 1}:")
    print([vectorizer.get_feature_names_out()[i] for i in topic.argsort()[-10:]])
    print("\n")
```

```
Topic 1:
['located', 'change', 'surgeon', 'june', 'rectify', 'explains', 'explaining', 'claime
d', 'availability', 'acne']


Topic 2:
['saw', 'browser', 'reflected', 'sworn', 'pol987654', 'exists', 'didn', 'security', 'j
smith123', 'went']


Topic 3:
['ensure', 'login', 'upgraded', 'minor', 'exactly', 'status', 'accepting', 'cell', 'ty
pe', 'wonderful']


Topic 4:
['pains', 'happens', 'efficient', 'status', 'family', 'medicine', 'session', 'yesterda
y', 'goodbye', 'friday']


Topic 5:
['decide', 'reason', 'letting', 'believe', 'stated', 'pa', 'mention', 'quarter', 'conf
used', 'group']


Topic 6:
['thing', 'whispers', 'checking', 'sooner', 'signed', 'worked', 'didn', 'ah', 'able',
'work']


Topic 7:
['called', 'reason', 'platinum', 'set', 'error', 'specialties', 'thinking', 'dermatolo
gy', 'believe', 'availability']


Topic 8:
['received', 'cover', 'pre', 'okay', 'denied', 'service', 'new', 'claim', 'thank', 'po
licy']


Topic 9:
['called', 'mozilla', 'folder', 'spam', 'chronic', 'click', 'recognize', 'according',
'goodbye', 'allows']


Topic 10:
['03', 'later', 'overpayment', 'early', 'qualifications', 'share', 'maybe', 'thinkin
g', 'ahead', 'orthopedic']


Topic 11:
['thursday', 'pm', 'sounds', 'thank', 'like', 'dr', 'wednesday', 'specialist', 'appoin
tment', 'schedule']


Topic 12:
['logging', 'tried', 'having', 'account', 'okay', 've', 'email', 'pause', 'try', 'pass
word']


Topic 13:
['gold', 'included', 'selecting', 'remember', 'goodbye', 'beginning', 'line', 'xyz12
3', 'surgeon', 'year']
```

```
Topic 14:
['76700', 'balance', 'kim', 'incorrectly', '30', 'clm987654', '1234', '01', 'rachel',
'00']


Topic 15:
['recent', 'thank', 'policy', '20', 'doctor', 'thought', '50', 'charged', 'visit', 'co
pay']
```

Explanation:

```
-Latent Dirichlet Allocation (LDA) is a popular topic modeling
technique that groups words into topics based on their co-occurrence
in documents (customer conversations, in this case).
-This analysis helps identify the main topics or issues being
discussed, such as "claims," "policy," or "surgery."
-The results can guide customer support teams to focus on the most
frequent topics and issues.
```

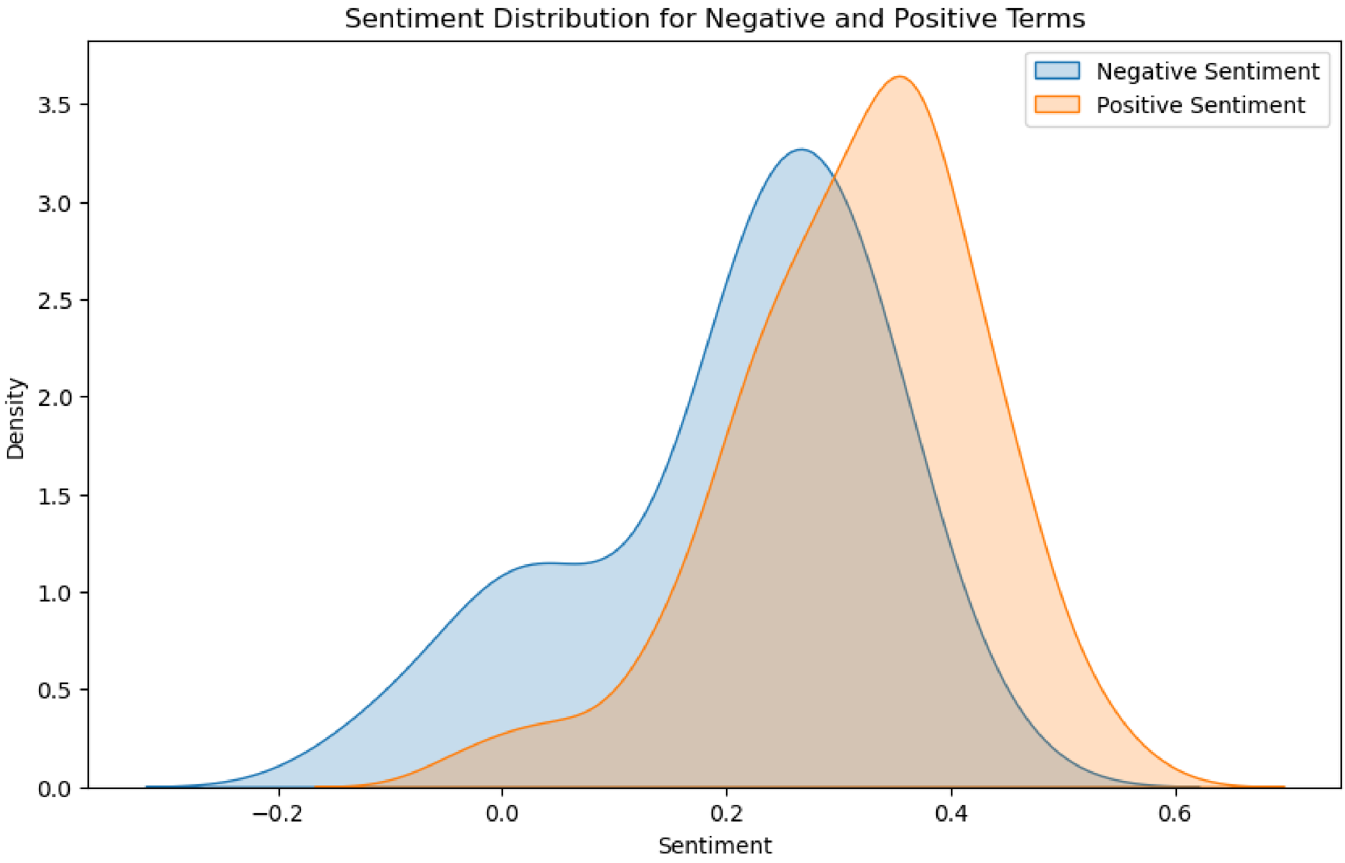
Example explanations of the result:

```
Topic 1:
Keywords: located, change, surgeon, june, rectify, explains, acne,
availability
Interpretation: This topic seems to be related to medical issues or
consultations, particularly concerning
surgeon availability, potential changes to appointments, and
rectifying problems related to specific
medical conditions like acne. The month of June might be relevant to
scheduling or policy changes.

Topic 3:
Keywords: ensure, login, upgraded, status, accepting, cell, type
Interpretation: This topic seems to focus on issues related to account
access and login problems,
potentially stemming from a recent upgrade or system changes. There
might also be issues with accepting
certain statuses or information during the login process. Keywords
like cell and type could point to mobile
app or system settings.

Topic 5:
Keywords: decide, reason, letting, believe, confused, group
Interpretation: This topic suggests decision-making around a service
or policy. Words like decide, reason,
and believe indicate a customer's thought process on a particular
issue, possibly related to group policies
or decisions that need to be made. Confused suggests customers
struggling to understand something in the
process.
```

In [22]:

```
# Define a list of negative and positive terms based on sentiment
negative_terms = ['denied', 'frustrated', 'upset', 'wrong', 'miscommunication']
positive_terms = ['approved', 'happy', 'thank', 'good', 'helpful']

# Count the presence of negative and positive terms
df['contains_negative'] = df['customer_text'].apply(lambda x: any(term in x.lower() f
df['contains_positive'] = df['customer_text'].apply(lambda x: any(term in x.lower() f

# Calculate sentiment for negative and positive term groups
```

```
negative_sentiment = df[df['contains_negative']]['sentiment']
positive_sentiment = df[df['contains_positive']]['sentiment']

# Visualize the sentiment of conversations with negative and positive terms
plt.figure(figsize=(10, 6))
sns.kdeplot(negative_sentiment, label='Negative Sentiment', fill=True)
sns.kdeplot(positive_sentiment, label='Positive Sentiment', fill=True)
plt.title('Sentiment Distribution for Negative and Positive Terms')
plt.xlabel('Sentiment')
plt.ylabel('Density')
plt.legend()
plt.show()
```



Explanation:

```
–We manually define a set of negative and positive terms based on what
we believe might evoke negative or positive sentiments in the customer
conversations.
–By identifying whether a conversation contains these terms, we can
check if their presence correlates with more negative or positive
sentiments.
–This analysis helps to understand the relationship between specific
keywords and customer emotions, which can be useful for improving
responses to common frustrations.
```

In [23]:
```
from scipy import stats

# Split the data into short and long conversations (e.g., median length as threshold)
median_length = df['text_length'].median()
short_conversations = df[df['text_length'] <= median_length]
long_conversations = df[df['text_length'] > median_length]

# Perform a t-test to check for significant difference in sentiment
t_stat, p_value = stats.ttest_ind(short_conversations['sentiment'], long_conversation

# Output the results
print(f"T-statistic: {t_stat:.2f}, P-value: {p_value:.2f}")
```

```python
# Interpret the p-value
if p_value < 0.05:
    print("There is a statistically significant difference between short and long con
else:
    print("No statistically significant difference between short and long conversatio
```

```
T-statistic: 1.37, P-value: 0.17
No statistically significant difference between short and long conversations' sentimen
ts.
```

Explanation:

```
-A t-test is used to determine whether the average sentiment differs
between short and long conversations.
-The p-value tells us whether the difference is statistically
significant.
-This analysis helps assess if longer conversations are more likely to
be negative (e.g., due to frustration with unresolved issues).
```

Conclusion

```
Topic Modeling (LDA): Uncovers hidden topics discussed in customer
conversations, helping identify main themes.
Sentiment & Keywords: Analyzes how the presence of specific terms
correlates with sentiment, revealing drivers of customer satisfaction
or frustration.
Temporal Trends: Visualizes customer conversation frequency over time,
helping identify patterns (e.g., peak complaint periods).
Correlation Between Text Length and Sentiment: Investigates if longer
conversations correlate with more positive or negative sentiments.
Statistical Significance (t-test): Compares the sentiment between
short and long conversations to see if the difference is significant.
```

In [ ]: