```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


# Load the CSV files
calls_df = pd.read_csv('callsf0d4f5a.csv')
cust_df = pd.read_csv('customers2afd6ea.csv')
reas_df = pd.read_csv('reason18315ff.csv')
sent_df = pd.read_csv('sentiment_statisticscc1e57a.csv')
test_df = pd.read_csv('testbc7185d.csv')


calls_df.head()
```

	call_id	customer_id	agent_id	call_start_datetime	agent_assigned_datetime	call_end_datetime	call_transcript
0	4667960400	2033123310	963118	7/31/2024 23:56	8/1/2024 0:03	8/1/2024 0:34	\n\nAgent: Thank you for calling United Airlin...
1	1122072124	8186702651	519057	8/1/2024 0:03	8/1/2024 0:06	8/1/2024 0:18	\n\nAgent: Thank you for calling United Airlin...
2	6834291559	2416856629	158319	7/31/2024 23:59	8/1/2024 0:07	8/1/2024 0:26	\n\nAgent: Thank you for calling United Airlin...

Next steps: Generate code with calls_df | View recommended plots | New interactive sheet

```
cust_df.head()
```

	customer_id	customer_name	elite_level_code
0	2033123310	Matthew Foster	4.0
1	8186702651	Tammy Walters	NaN
2	2416856629	Jeffery Dixon	NaN
3	1154544516	David Wilkins	2.0
4	5214456437	Elizabeth Daniels	0.0

Next steps: Generate code with cust_df | View recommended plots | New interactive sheet

```
cust_df.fillna(np.median(cust_df["elite_level_code"]),inplace=True)


cust_df['elite_level_code'] = pd.to_numeric(cust_df['elite_level_code'], errors='coerce')

# Calculate the median of the elite_level_code column
median_value = np.median(cust_df['elite_level_code'])

# Fill NaN values in the elite_level_code column with the median
cust_df['elite_level_code'].fillna(median_value, inplace=True)

# Check the updated DataFrame
print(cust_df['elite_level_code'].isnull().sum())  # This should show 0 if all NaN values are filled
```

```
25767
<ipython-input-155-874d66024d59>:7: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assigr
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  cust_df['elite_level_code'].fillna(median_value, inplace=True)
```

```
reas_df.head()
```

	call_id	primary_call_reason
0	4667960400	Voluntary Cancel
1	1122072124	Booking
2	6834291559	IRROPS
3	2266439882	Upgrade
4	1211603231	Seating

Next steps: Generate code with reas_df | View recommended plots | New interactive sheet

```
import re


import numpy as np

# Dictionary for known duplicate mappings
duplicate_mappings = {
    "Postflight": "Post Flight",  # Correcting typo
    "Checkin": "Check In",
    "Products Services": "Products And Services",
    "Postflight": "Post Flight",
    # Add more mappings here if other duplicates are identified
}

# Function to clean and normalize text with uppercase first letters
def text_cleaning(text):
    if isinstance(text, str):  # Check if the value is a string
        text = text.strip()  # Remove leading/trailing whitespaces
        text = text.lower()  # Convert to lowercase
        text = re.sub(r'[-]', '', text)  # Remove hyphens
        text = re.sub(r'[^a-z0-9\s]', '', text)  # Remove special characters
        text = re.sub(r'\s+', ' ', text)  # Replace multiple spaces with a single space
        text = text.strip()  # Return cleaned text with leading/trailing spaces removed
        text = text.title()  # Convert first letter of each word to uppercase

        # Apply known duplicate mappings
        if text in duplicate_mappings:
            text = duplicate_mappings[text]

        return text
    else:
        return np.nan  # Return NaN for non-string values


# Apply the cleaning function
reas_df['primary_call_reason'] = reas_df['primary_call_reason'].apply(text_cleaning)


# Check the results
print(reas_df['primary_call_reason'].unique())
```

```
['Voluntary Cancel' 'Booking' 'Irrops' 'Upgrade' 'Seating' 'Mileage Plus'
 'Checkout' 'Voluntary Change' 'Post Flight' 'Check In' 'Other Topics'
 'Communications' 'Schedule Change' 'Products And Services'
 'Digital Support' 'Disability' 'Unaccompanied Minor' 'Baggage'
 'Traveler Updates' 'Etc']
```

```
sent_df.head()
```

	call_id	agent_id	agent_tone	customer_tone	average_sentiment	silence_percent_average
0	4667960400	963118	neutral	angry	-0.04	0.39
1	1122072124	519057	calm	neutral	0.02	0.35
2	6834291559	158319	neutral	polite	-0.13	0.32
3	2266439882	488324	neutral	frustrated	-0.20	0.20
4	1211603231	721730	neutral	polite	-0.05	0.35

Next steps: Generate code with sent_df | View recommended plots | New interactive sheet

```
test_df.head()
```

	call_id
0	7732610078
1	2400299738
2	6533095063
3	7774450920
4	9214147168

Next steps: Generate code with test_df | View recommended plots | New interactive sheet

```
calls_df.isnull().sum()
```

	0
call_id	0
customer_id	0
agent_id	0
call_start_datetime	0
agent_assigned_datetime	0
call_end_datetime	0
call_transcript	0

```
cust_df.isnull().sum()
```

	0
customer_id	0
customer_name	0
elite_level_code	25767

```
reas_df.isnull().sum()
```

	0
call_id	0
primary_call_reason	0

```
sent_df.isnull().sum()
```

	0
call_id	0
agent_id	0
agent_tone	217
customer_tone	0
average_sentiment	109
silence_percent_average	0

```
test_df.head()
```

	call_id
0	7732610078
1	2400299738
2	6533095063
3	7774450920
4	9214147168

Next steps: Generate code with test_df | View recommended plots | New interactive sheet

```
test_df.isnull().sum()
```

	0
call_id	0

```
calls_df.duplicated().sum()
```

```
0
```

```
calls_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 71810 entries, 0 to 71809
Data columns (total 7 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   call_id                  71810 non-null  int64 
 1   customer_id              71810 non-null  int64 
 2   agent_id                 71810 non-null  int64 
 3   call_start_datetime      71810 non-null  object
 4   agent_assigned_datetime  71810 non-null  object
 5   call_end_datetime        71810 non-null  object
 6   call_transcript          71810 non-null  object
dtypes: int64(3), object(4)
memory usage: 3.8+ MB
```

Object to Datetime

```
# Convert date columns to datetime format
calls_df['call_start_datetime'] = pd.to_datetime(calls_df['call_start_datetime'])
calls_df['call_end_datetime'] = pd.to_datetime(calls_df['call_end_datetime'])
calls_df['agent_assigned_datetime'] = pd.to_datetime(calls_df['agent_assigned_datetime'])
print(calls_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 71810 entries, 0 to 71809
Data columns (total 7 columns):
 #   Column                   Non-Null Count  Dtype         
---  ------                   --------------  -----         
 0   call_id                  71810 non-null  int64         
 1   customer_id              71810 non-null  int64         
 2   agent_id                 71810 non-null  int64         
 3   call_start_datetime      71810 non-null  datetime64[ns]
 4   agent_assigned_datetime  71810 non-null  datetime64[ns]
 5   call_end_datetime        71810 non-null  datetime64[ns]
 6   call_transcript          71810 non-null  object        
dtypes: datetime64[ns](3), int64(3), object(1)
memory usage: 3.8+ MB
None
```

```
calls_df.describe()
```

	call_id	customer_id	agent_id	call_start_datetime	agent_assigned_datetime	call_end_datetime
count	7.181000e+04	7.181000e+04	71810.000000	71810	71810	71810
mean	4.993574e+09	5.004334e+09	564768.278039	2024-08-16 10:42:34.023116544	2024-08-16 10:49:51.090655744	2024-08-16 11:01:28.139256320
min	1.316420e+05	1.197800e+04	102574.000000	2024-07-31 23:56:00	2024-08-01 00:03:00	2024-08-01 00:17:00
25%	2.480013e+09	2.514618e+09	347606.000000	2024-08-09 10:58:30	2024-08-09 11:07:15	2024-08-09 11:19:30
50%	4.989448e+09	4.999664e+09	591778.000000	2024-08-17 09:02:00	2024-08-17 09:08:00	2024-08-17 09:20:00
75%	7.493629e+09	7.509126e+09	786323.000000	2024-08-24 14:17:00	2024-08-24 14:24:45	2024-08-24 14:36:00
max	9.999806e+09	9.999935e+09	993862.000000	2024-08-31 23:55:00	2024-08-31 23:59:00	2024-09-01 01:33:00

```
# Check for duplicate rows
duplicates = calls_df.duplicated().sum()
print(f'Duplicate rows: {duplicates}')

# Drop duplicates if any
calls_data = calls_df.drop_duplicates()
```

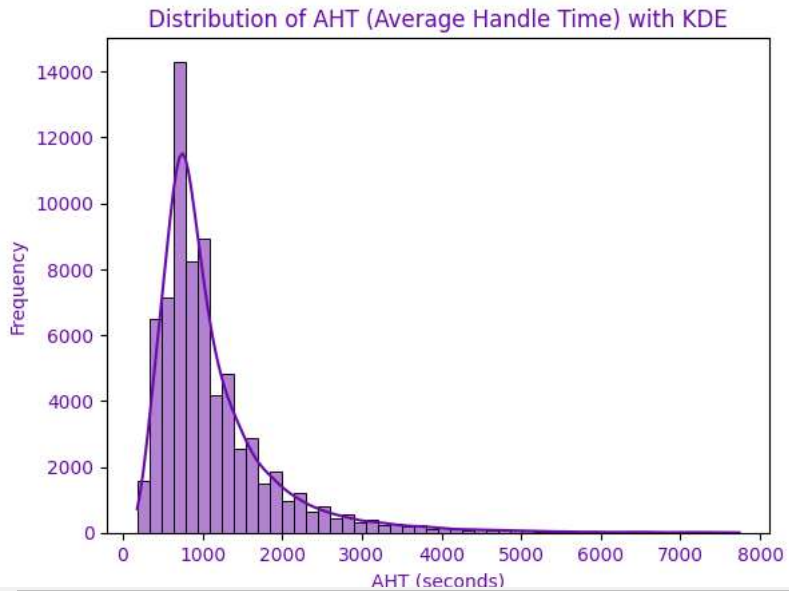
```
Duplicate rows: 0
```

```
# Calculate AHT(Average Handle Time) and AST(Average Speed to answer)
calls_data['AHT'] = (calls_data['call_end_datetime'] - calls_data['call_start_datetime']).dt.total_seconds()
calls_data['AST'] = (calls_data['agent_assigned_datetime'] - calls_data['call_start_datetime']).dt.total_seconds()

# Display the new columns
print(calls_data[['AHT', 'AST']].head())
```

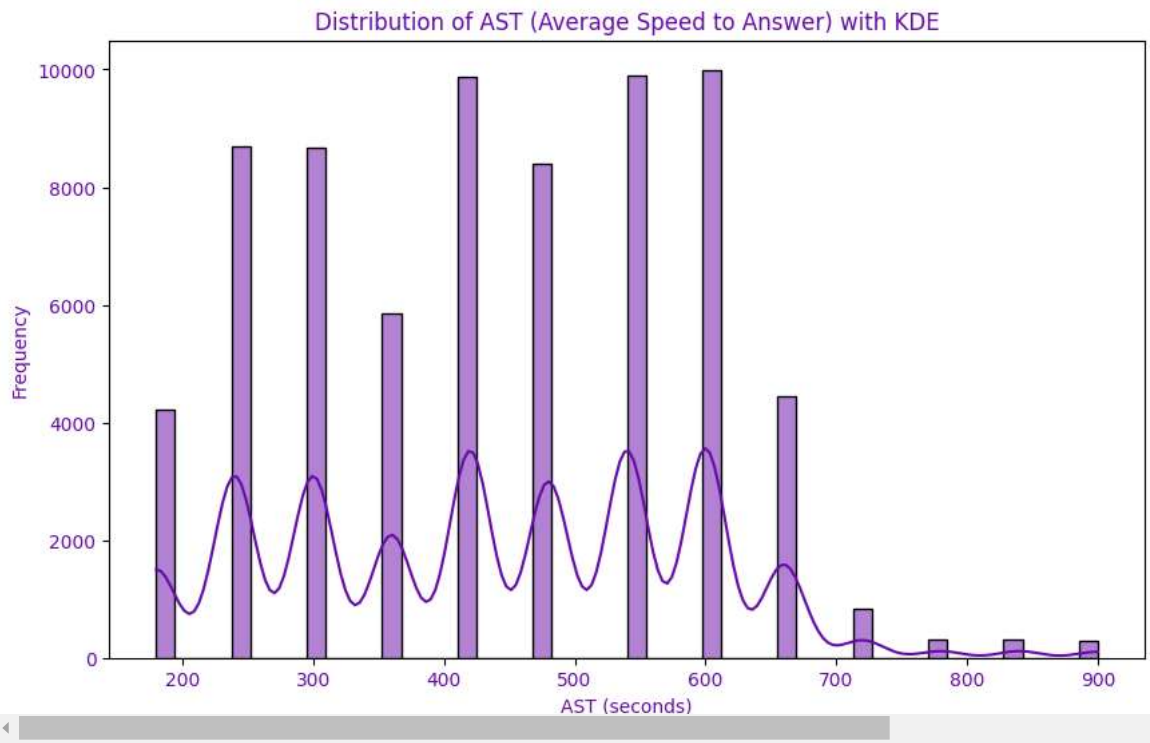
```
      AHT    AST
0  2280.0  420.0
1   900.0  180.0
2  1620.0  480.0
3   720.0  300.0
4  1140.0  600.0
```

Univariate Analysis

```
import seaborn as sns
import matplotlib.pyplot as plt

# Updated colors to match your vibrant presentation
plot_color = "#6A0DAD"  # Amethyst Purple (matches your presentation's tone)

# Histogram with KDE for AHT
sns.histplot(calls_data['AHT'].dropna(), bins=50, color=plot_color, kde=True, line_kws={"color": "#FF6347"})  # Using Tomato for KDE line
plt.title('Distribution of AHT (Average Handle Time) with KDE', color=plot_color)
plt.xlabel('AHT (seconds)', color=plot_color)
plt.ylabel('Frequency', color=plot_color)
plt.xticks(color=plot_color)
plt.yticks(color=plot_color)
plt.show()
```

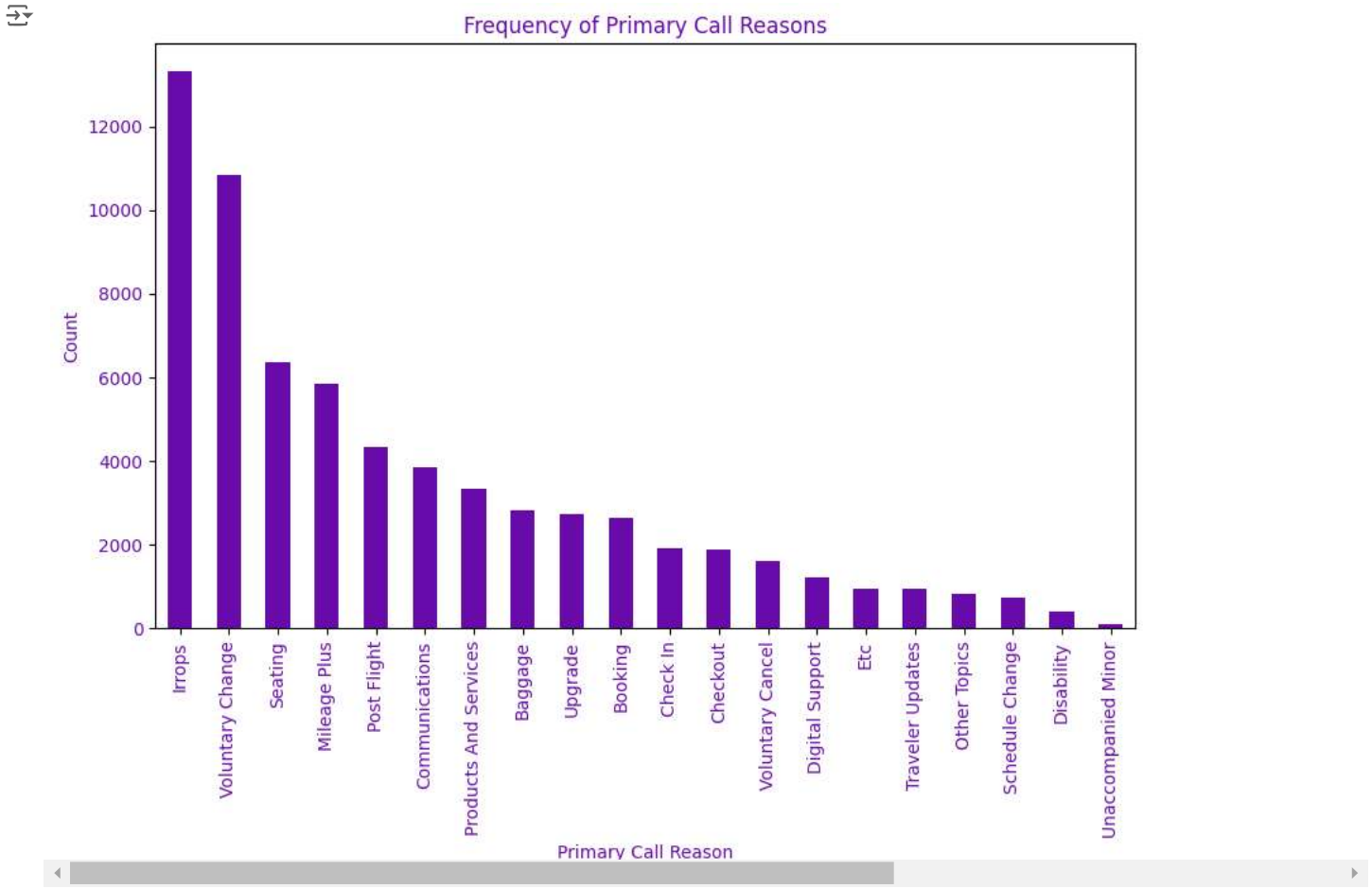



```
import seaborn as sns
import matplotlib.pyplot as plt

# Updated colors to match your vibrant presentation
plot_color = "#6A0DAD"  # Amethyst Purple (matches your presentation's tone)

# Plotting the distribution of AST with KDE
plt.figure(figsize=(10, 6))
sns.histplot(calls_data['AST'].dropna(), bins=50, color=plot_color, kde=True, line_kws={"color": "#FF6347"})  # Using Tomato for KDE line
plt.title('Distribution of AST (Average Speed to Answer) with KDE', color=plot_color)
plt.xlabel('AST (seconds)', color=plot_color)
plt.ylabel('Frequency', color=plot_color)
plt.xticks(color=plot_color)
plt.yticks(color=plot_color)
plt.show()
```



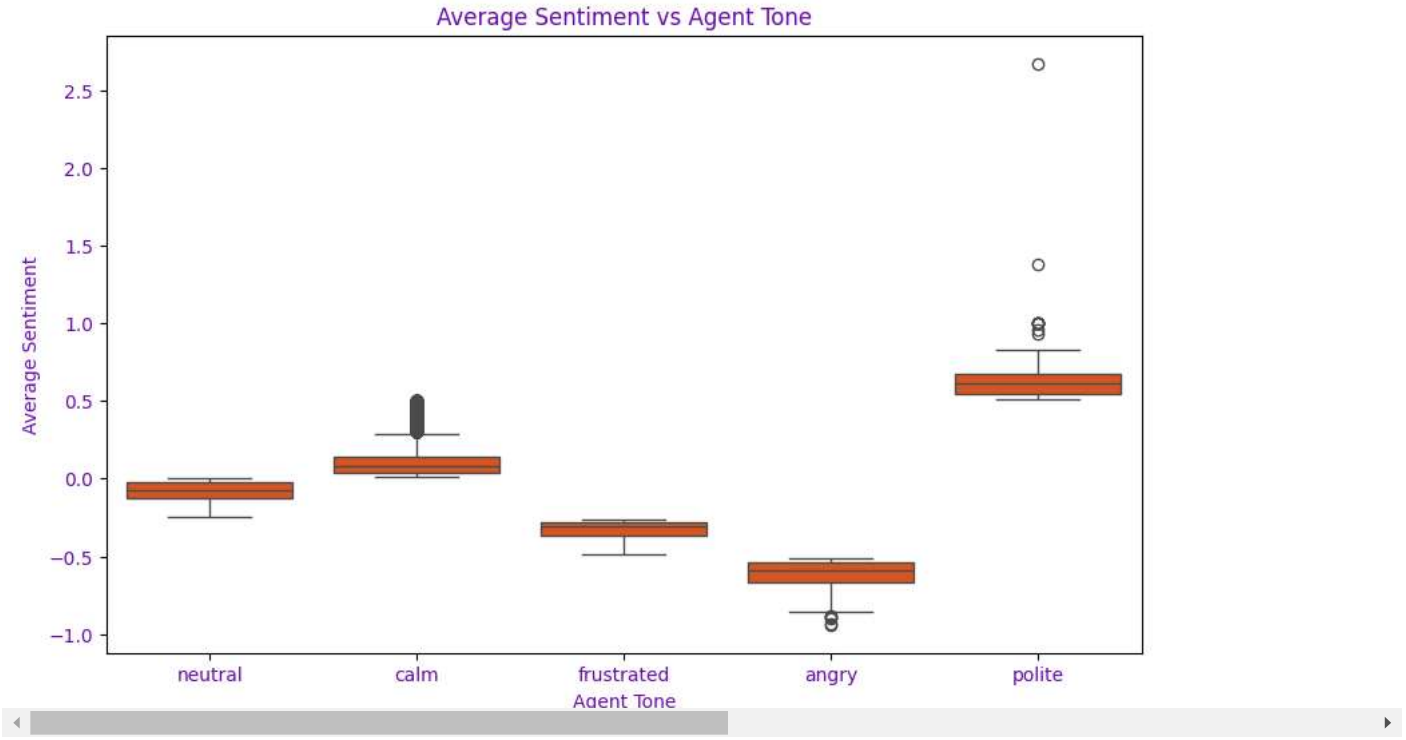

```
import matplotlib.pyplot as plt

# Updated colors to match your vibrant presentation
bar_color = "#6A0DAD"  # Amethyst Purple (to match the presentation)
```

```
# Bar chart for primary call reasons
reas_df['primary_call_reason'].value_counts().plot(kind='bar', figsize=(10, 6), color=bar_color)
plt.title('Frequency of Primary Call Reasons', color=bar_color)
plt.xlabel('Primary Call Reason', color=bar_color)
plt.ylabel('Count', color=bar_color)
plt.xticks(color=bar_color)
plt.yticks(color=bar_color)
plt.show()
```




```
import matplotlib.pyplot as plt
import seaborn as sns

# Updated color for the plot elements
box_color = "#FF4500"  # Orange Red for the boxplot outline to contrast with your presentation's vibrant style
title_color = "#6A0DAD"  # Amethyst Purple for the title
label_color = "#6A0DAD"  # Amethyst Purple for the labels

# Average sentiment analysis
plt.figure(figsize=(10, 6))
sns.boxplot(x='agent_tone', y='average_sentiment', data=sent_df, color=box_color, saturation=0.7)
plt.title('Average Sentiment vs Agent Tone', color=title_color)
plt.xlabel('Agent Tone', color=label_color)
plt.ylabel('Average Sentiment', color=label_color)
plt.xticks(color=label_color)
plt.yticks(color=label_color)
plt.show()
```

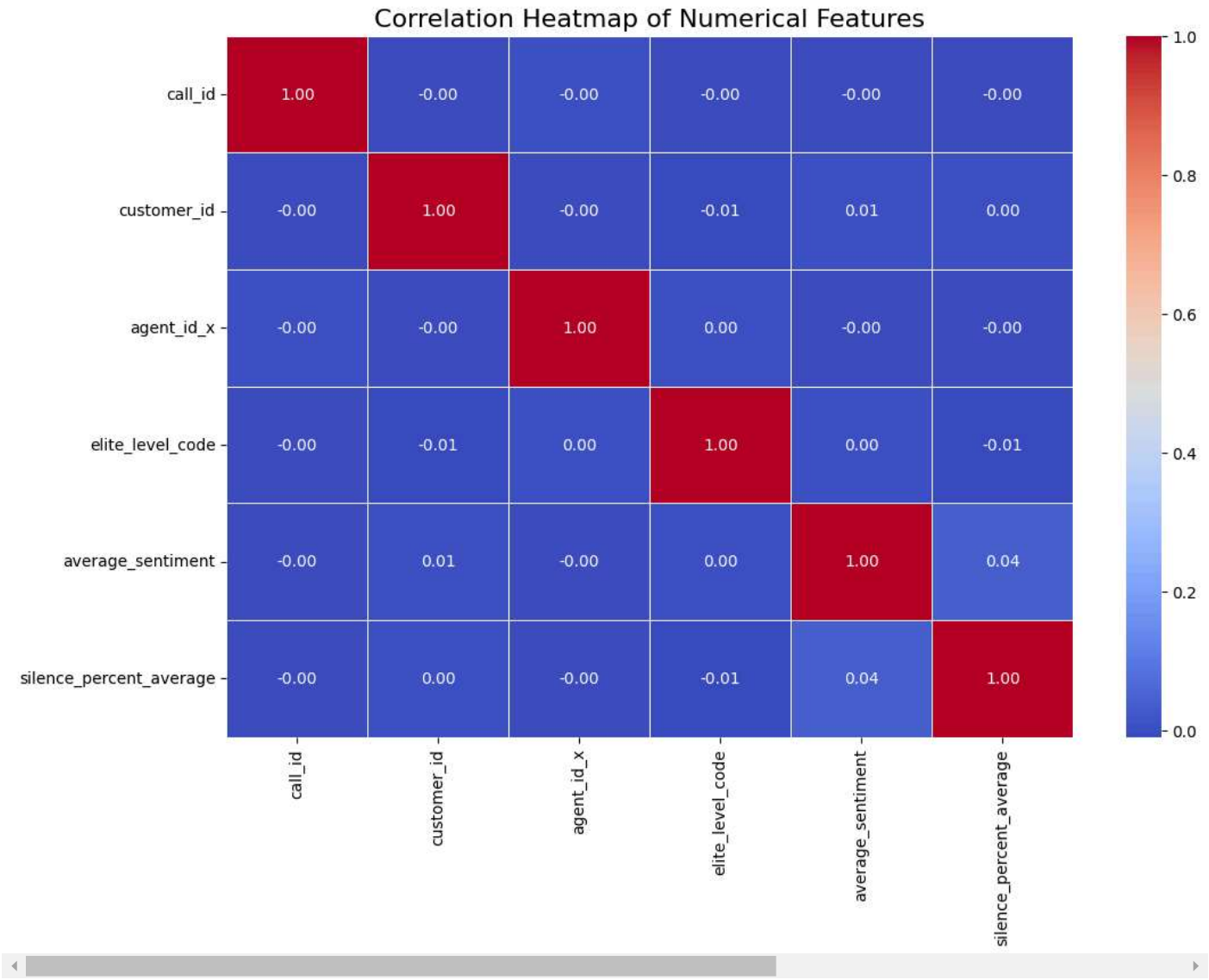
```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:640: FutureWarning: SeriesGroupBy.grouper is deprecated and will be remov
  positions = grouped.grouper.result_index.to_numpy(dtype=float)
```




```
# Merging calls_df with other DataFrames
merged_df = calls_df.merge(cust_df, on='customer_id', how='left') \
                    .merge(reas_df, on='call_id', how='left') \
                    .merge(sent_df, on='call_id', how='left')

# Make sure the AHT column is present
print(merged_df.columns)
```

```
Index(['call_id', 'customer_id', 'agent_id_x', 'call_start_datetime',
       'agent_assigned_datetime', 'call_end_datetime', 'call_transcript',
       'customer_name', 'elite_level_code', 'primary_call_reason',
       'agent_id_y', 'agent_tone', 'customer_tone', 'average_sentiment',
       'silence_percent_average'],
      dtype='object')
```

```
# Drop one of the duplicate agent_id columns (e.g., agent_id_y)
merged_df_cleaned = merged_df.drop(columns=['agent_id_y'])

# Step 1: Filter only numerical columns
numerical_cols = merged_df_cleaned.select_dtypes(include=['float64', 'int64'])

# Step 2: Handle any missing values (if necessary)
numerical_cols = numerical_cols.fillna(0)

# Step 3: Calculate the correlation matrix
corr_matrix = numerical_cols.corr()

# Step 4: Plot the heatmap
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)

# Add title
plt.title('Correlation Heatmap of Numerical Features', size=16)

# Show the plot
plt.show()
```




```
merged_df.shape
```

```
(71810, 15)
```

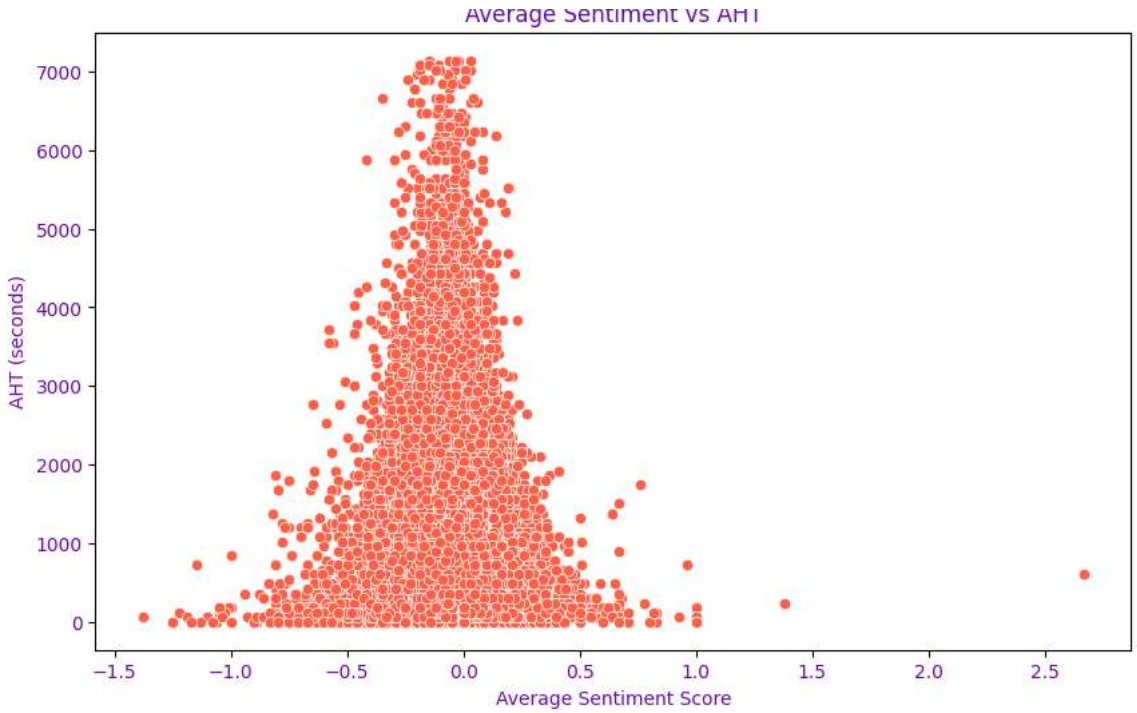
```
# Ensure that the relevant columns are in datetime format
merged_df['agent_assigned_datetime'] = pd.to_datetime(merged_df['agent_assigned_datetime'])
merged_df['call_end_datetime'] = pd.to_datetime(merged_df['call_end_datetime'])

# Calculate AHT in seconds
merged_df['AHT'] = (merged_df['call_end_datetime'] - merged_df['agent_assigned_datetime']).dt.total_seconds()
```

```
# Group by primary call reason and calculate mean AHT
call_reason_aht = merged_df.groupby('primary_call_reason')['AHT'].mean().sort_values(ascending=False)
print(call_reason_aht)
```

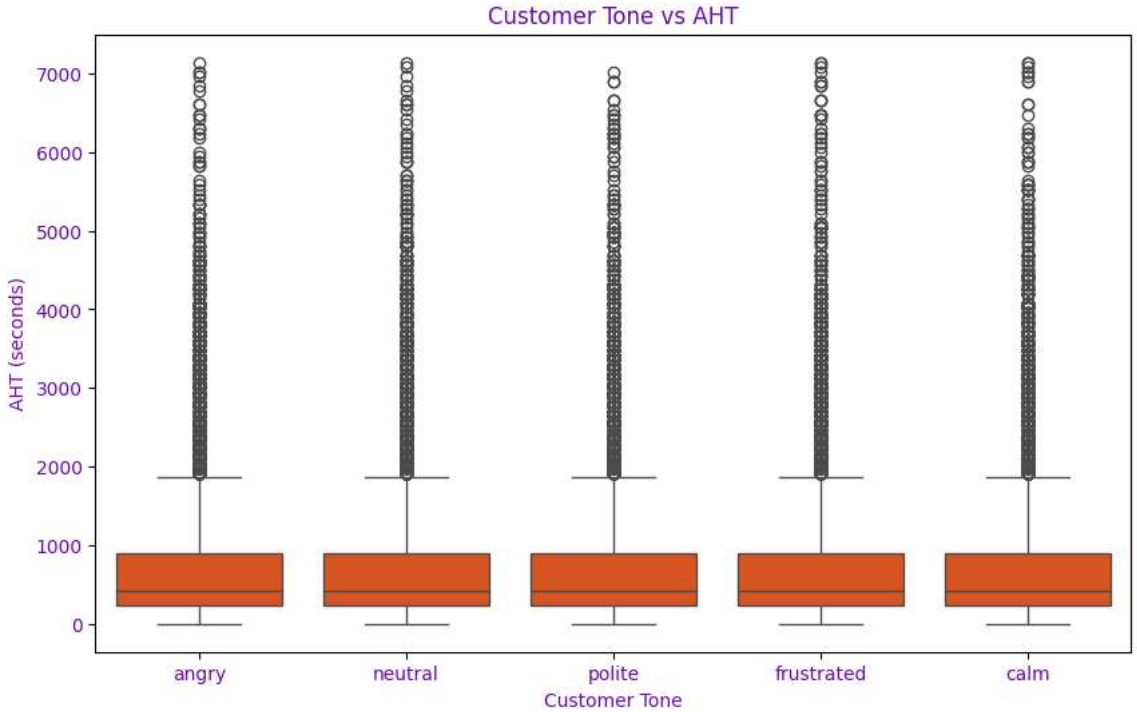
```
primary_call_reason
Checkout                 1016.853814
Mileage Plus              995.573406
Etc                       962.899160
Post Flight               932.896074
Communications            826.718750
Irrops                    785.116069
Products And Services     746.560624
Voluntary Cancel          721.866833
Voluntary Change          639.153761
Upgrade                   632.344777
Check In                  574.128151
```

```
Unaccompanied Minor       519.230769
Schedule Change           490.013680
Seating                   474.994501
Booking                   427.736064
Traveler Updates          393.233725
Digital Support           372.293878
Other Topics              350.097800
Baggage                   333.644068
Disability                292.109181
Name: AHT, dtype: float64
```

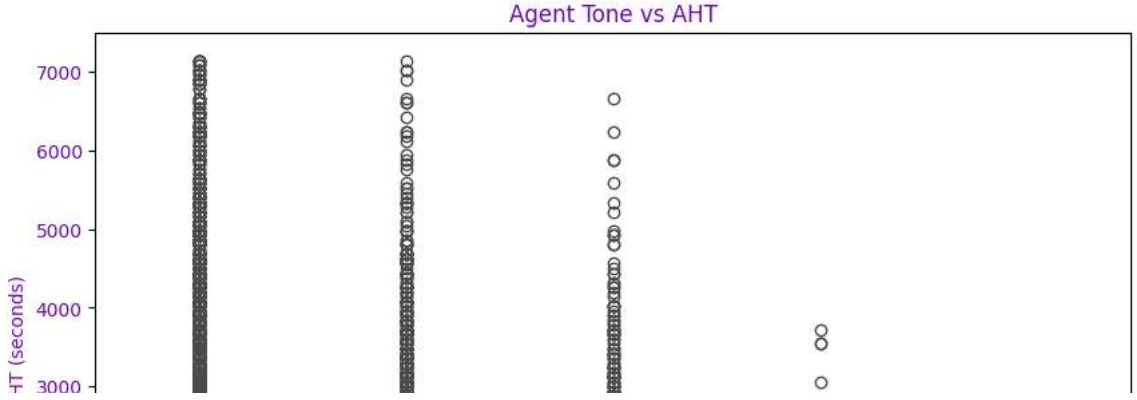
```
import seaborn as sns
import matplotlib.pyplot as plt

# Define colors based on the presentation palette
scatter_color = "#FF6347"  # Tomato for scatter plot points
box_color = "#FF4500"       # Orange Red for boxplot outlines
title_color = "#6A0DAD"     # Amethyst Purple for titles
label_color = "#6A0DAD"     # Amethyst Purple for labels

# Scatterplot: Average Sentiment vs AHT
plt.figure(figsize=(10, 6))
sns.scatterplot(x=merged_df['average_sentiment'], y=merged_df['AHT'], color=scatter_color)
plt.title('Average Sentiment vs AHT', color=title_color)
plt.xlabel('Average Sentiment Score', color=label_color)
plt.ylabel('AHT (seconds)', color=label_color)
plt.xticks(color=label_color)
plt.yticks(color=label_color)
plt.show()

# Boxplot: Customer Tone vs AHT
plt.figure(figsize=(10, 6))
sns.boxplot(x=merged_df['customer_tone'], y=merged_df['AHT'], color=box_color, saturation=0.7)
plt.title('Customer Tone vs AHT', color=title_color)
plt.xlabel('Customer Tone', color=label_color)
plt.ylabel('AHT (seconds)', color=label_color)
plt.xticks(color=label_color)
plt.yticks(color=label_color)
plt.show()

# Boxplot: Agent Tone vs AHT
plt.figure(figsize=(10, 6))
sns.boxplot(x=merged_df['agent_tone'], y=merged_df['AHT'], color=box_color, saturation=0.7)
plt.title('Agent Tone vs AHT', color=title_color)
plt.xlabel('Agent Tone', color=label_color)
plt.ylabel('AHT (seconds)', color=label_color)
plt.xticks(color=label_color)
plt.yticks(color=label_color)
plt.show()
```

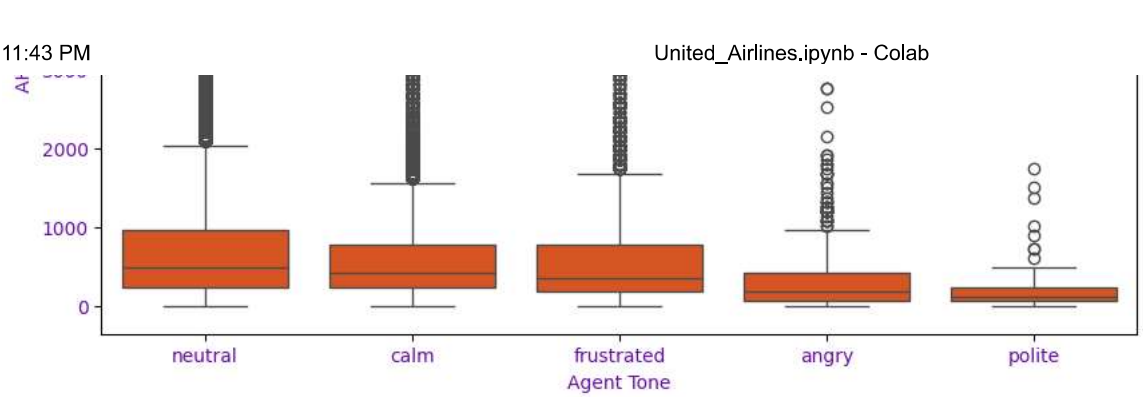



```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:640: FutureWarning: SeriesGroupBy.grouper is deprecated and will be re
  positions = grouped.grouper.result_index.to_numpy(dtype=float)
```




```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:640: FutureWarning: SeriesGroupBy.grouper is deprecated and will be re
  positions = grouped.grouper.result_index.to_numpy(dtype=float)
```






```
# Convert categorical values (tones) to numeric codes
merged_df['customer_tone_numeric'] = merged_df['customer_tone'].astype('category').cat.codes
merged_df['agent_tone_numeric'] = merged_df['agent_tone'].astype('category').cat.codes

# Check if conversion is successful
print(merged_df[['customer_tone', 'customer_tone_numeric', 'agent_tone', 'agent_tone_numeric']].head())
```

```
  customer_tone  customer_tone_numeric agent_tone  agent_tone_numeric
0         angry                      0    neutral                   3
1       neutral                      3       calm                   1
2        polite                      4    neutral                   3
3    frustrated                      2    neutral                   3
4        polite                      4    neutral                   3
```

```
# Correlation between AHT and sentiment, with numeric tone values
correlation = merged_df[['AHT', 'average_sentiment', 'customer_tone_numeric', 'agent_tone_numeric']].corr()
print(correlation)
```

```
                           AHT  average_sentiment  customer_tone_numeric  \
AHT                   1.000000          -0.076065              -0.000667
average_sentiment    -0.076065           1.000000              -0.005221
customer_tone_numeric -0.000667          -0.005221               1.000000
agent_tone_numeric     0.081953          -0.459861              -0.002208

                       agent_tone_numeric
AHT                              0.081953
average_sentiment               -0.459861
customer_tone_numeric           -0.002208
agent_tone_numeric               1.000000
```

```
# Group by customer tone and calculate mean AHT
customer_tone_aht = merged_df.groupby('customer_tone')['AHT'].mean().sort_values(ascending=False)
print("Mean AHT by Customer Tone:\n", customer_tone_aht)



# Group by agent tone and calculate mean AHT
agent_tone_aht = merged_df.groupby('agent_tone')['AHT'].mean().sort_values(ascending=False)
print("Mean AHT by Agent Tone:\n", agent_tone_aht)
```

```
Mean AHT by Customer Tone:
 customer_tone
neutral       707.625312
calm          699.604098
angry         695.530261
frustrated    692.738764
polite        689.659731
Name: AHT, dtype: float64
Mean AHT by Agent Tone:
 agent_tone
neutral       750.276439
calm          626.499696
frustrated    617.068404
angry         394.395887
polite        220.000000
Name: AHT, dtype: float64
```

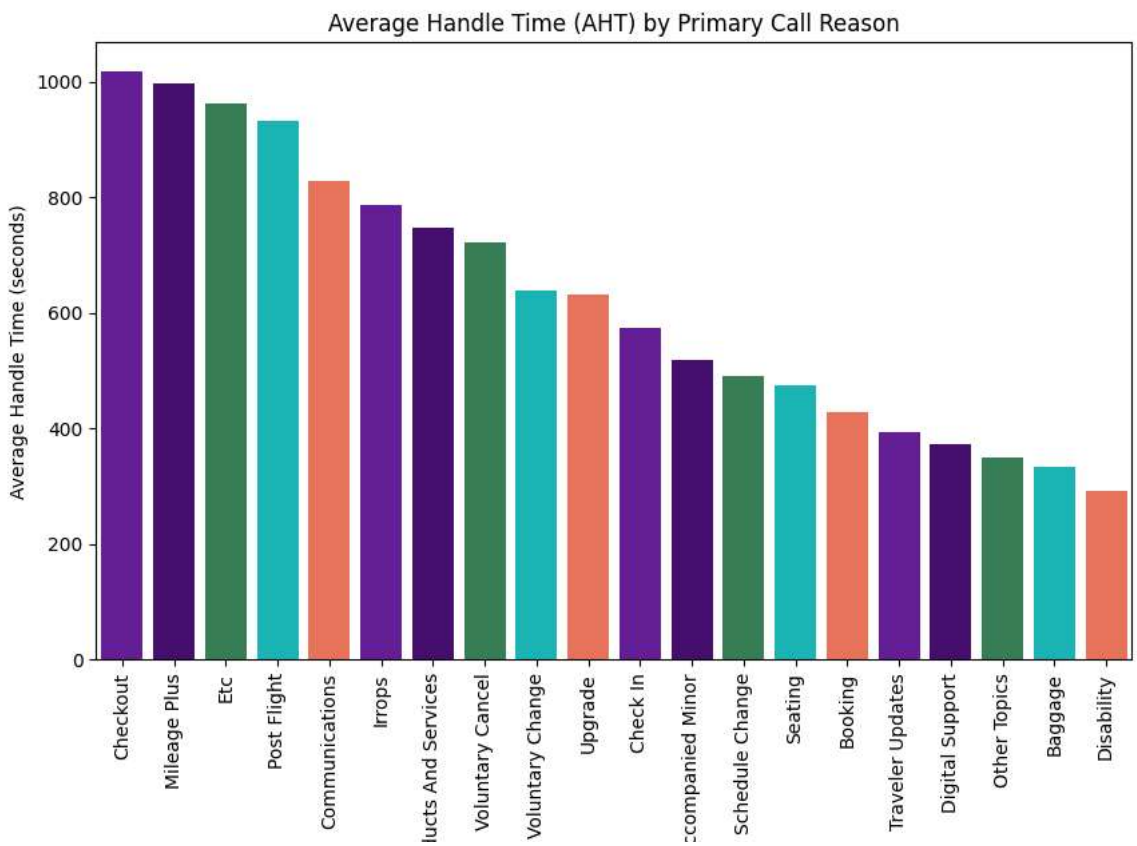
```
# Group by primary_call_reason and calculate mean AHT
call_reason_aht = merged_df.groupby('primary_call_reason')['AHT'].mean().sort_values(ascending=False)
```

```
print("Mean AHT by Primary Call Reason:\n", call_reason_aht)
```

```
Mean AHT by Primary Call Reason:
 primary_call_reason
Checkout                 1016.853814
Mileage Plus              995.573406
Etc                       962.899160
Post Flight               932.896074
Communications            826.718750
Irrops                    785.116069
Products And Services     746.560624
Voluntary Cancel          721.866833
Voluntary Change          639.153761
Upgrade                   632.344777
Check In                  574.128151
Unaccompanied Minor       519.230769
Schedule Change           490.013680
Seating                   474.994501
Booking                   427.736064
Traveler Updates          393.233725
Digital Support           372.293878
Other Topics              350.097800
Baggage                   333.644068
Disability                292.109181
Name: AHT, dtype: float64
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Barplot for AHT by call reason
plt.figure(figsize=(10, 6))
sns.barplot(x=call_reason_aht.index, y=call_reason_aht.values, palette=["#6A0DAD", "#4B0082", "#2E8B57", "#00CED1", "#FF6347"])
plt.title('Average Handle Time (AHT) by Primary Call Reason')
plt.xticks(rotation=90)
plt.xlabel('Primary Call Reason')
plt.ylabel('Average Handle Time (seconds)')
plt.show()
```

```
<ipython-input-189-62c8dea0091f>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `leg

  sns.barplot(x=call_reason_aht.index, y=call_reason_aht.values, palette=["#6A0DAD", "#4B0082", "#2E8B57", "#00CED1", "#FF6347"])
<ipython-input-189-62c8dea0091f>:6: UserWarning:
The palette list has fewer values (5) than needed (20) and will cycle, which may produce an uninterpretable plot.
  sns.barplot(x=call_reason_aht.index, y=call_reason_aht.values, palette=["#6A0DAD", "#4B0082", "#2E8B57", "#00CED1", "#FF6347"])
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need t
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need t
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need t
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need t
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need t
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need t
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need t
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need t
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need t
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need t
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need t
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need t
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need t
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need t
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need t
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need t
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need t
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need t
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need t
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need t
  data_subset = grouped_data.get_group(pd_key)
```



Prod Unar

Primary Call Reason

```
# Group by primary_call_reason and calculate mean AHT
call_reason_aht = merged_df.groupby('primary_call_reason')['AHT'].mean().sort_values(ascending=False)

# Display the call reason with the longest AHT
longest_aht_reason = call_reason_aht.head(1)
print("Call Reason with Longest AHT:\n", longest_aht_reason)
```

```
Call Reason with Longest AHT:
 primary_call_reason
Checkout    1016.853814
Name: AHT, dtype: float64
```

```
# Analyze average silence percentage (Hold Time) and AHT correlation
hold_time_correlation = merged_df[['silence_percent_average', 'AHT']].corr()
print("Correlation between Hold Time and AHT:\n", hold_time_correlation)

# Identify agents with the longest hold times
hold_time_agents = merged_df.groupby('agent_id_x')['silence_percent_average'].mean().sort_values(ascending=False).head(10)
print("Agents with the Longest Hold Times:\n", hold_time_agents)
```

```
Correlation between Hold Time and AHT:
                           silence_percent_average       AHT
silence_percent_average                 1.000000  0.406883
AHT                                     0.406883  1.000000
Agents with the Longest Hold Times:
 agent_id_x
980156    0.560000
974978    0.533333
958516    0.447500
255256    0.443333
391553    0.433333
783441    0.433333
140146    0.420000
737543    0.410000
901274    0.405455
107876    0.405000
Name: silence_percent_average, dtype: float64
```

```
# Group by agent_id and calculate mean AHT
agent_aht = merged_df.groupby('agent_id_x')['AHT'].mean().sort_values(ascending=False)

# Display the agent with the highest AHT
struggling_agent = agent_aht.head(1)
print("Agent with the Highest AHT:\n", struggling_agent)
```

```
Agent with the Highest AHT:
 agent_id_x
102574    3600.0
Name: AHT, dtype: float64
```

```
# Analyze sentiment and silence percentage for the struggling agent
struggling_agent_id = struggling_agent.index[0]  # Get the ID of the agent with the highest AHT

# Filter data for that agent
struggling_agent_data = merged_df[merged_df['agent_id_x'] == struggling_agent_id]

# Average sentiment and silence percentage
agent_sentiment = struggling_agent_data['average_sentiment'].mean()
agent_silence = struggling_agent_data['silence_percent_average'].mean()

print(f"Average Sentiment for Agent {struggling_agent_id}: {agent_sentiment}")
print(f"Average Silence Percentage for Agent {struggling_agent_id}: {agent_silence}")
```

```
Average Sentiment for Agent 102574: -0.075
Average Silence Percentage for Agent 102574: 0.26
```

```
# Calculate average hold time (silence percent) for each call reason
hold_time_per_reason = merged_df.groupby('primary_call_reason')['silence_percent_average'].mean().sort_values(ascending=False)
```

```
print("Hold Time by Call Reason:\n", hold_time_per_reason)
```

```
Hold Time by Call Reason:
 primary_call_reason
Irrops                  0.309404
Communications          0.303578
Post Flight             0.297804
Checkout                0.292309
Voluntary Change        0.289599
Etc                     0.287563
Digital Support         0.285298
Traveler Updates        0.284717
Voluntary Cancel        0.281767
Upgrade                 0.277023
Products And Services   0.276759
Mileage Plus            0.275038
Other Topics            0.272726
Schedule Change         0.267223
Seating                 0.264613
Check In                0.262736
Booking                 0.260948
Baggage                 0.250508
Disability              0.248635
Unaccompanied Minor     0.242500
Name: silence_percent_average, dtype: float64
```

```
# Correlation between hold time (silence percentage) and AHT
hold_time_aht_correlation = merged_df[['silence_percent_average', 'AHT']].corr()
print("Correlation between Hold Time and AHT:\n", hold_time_aht_correlation)
```

```
Correlation between Hold Time and AHT:
                          silence_percent_average       AHT
silence_percent_average                 1.000000  0.406883
AHT                                     0.406883  1.000000
```

```
# Find agents with the highest average hold time
hold_time_agents = merged_df.groupby('agent_id_x')['silence_percent_average'].mean().sort_values(ascending=False)
print("Agents with the Longest Hold Times:\n", hold_time_agents.head(5))
```

```
Agents with the Longest Hold Times:
 agent_id_x
980156    0.560000
974978    0.533333
958516    0.447500
255256    0.443333
391553    0.433333
Name: silence_percent_average, dtype: float64
```

```
# Calculate average silence time for each call reason
silence_per_reason = merged_df.groupby('primary_call_reason')['silence_percent_average'].mean().sort_values(ascending=False)

print("Average Silence Time by Call Reason:\n", silence_per_reason)
```

```
Average Silence Time by Call Reason:
 primary_call_reason
Irrops                  0.309404
Communications          0.303578
Post Flight             0.297804
Checkout                0.292309
Voluntary Change        0.289599
Etc                     0.287563
Digital Support         0.285298
Traveler Updates        0.284717
Voluntary Cancel        0.281767
Upgrade                 0.277023
Products And Services   0.276759
Mileage Plus            0.275038
Other Topics            0.272726
Schedule Change         0.267223
Seating                 0.264613
Check In                0.262736
Booking                 0.260948
Baggage                 0.250508
Disability              0.248635
Unaccompanied Minor     0.242500
Name: silence_percent_average, dtype: float64
```

```
# Correlation between silence and AHT
silence_aht_corr = merged_df[['silence_percent_average', 'AHT']].corr()
print("Correlation between Silence Time and AHT:\n", silence_aht_corr)
```

```
Correlation between Silence Time and AHT:
                           silence_percent_average       AHT
silence_percent_average                 1.000000  0.406883
AHT                                     0.406883  1.000000
```

```
# Identify agents with the highest average silence time
silence_per_agent = merged_df.groupby('agent_id_x')['silence_percent_average'].mean().sort_values(ascending=False)

print("Agents with Highest Silence Time:\n", silence_per_agent.head(5))
```

```
Agents with Highest Silence Time:
 agent_id_x
980156    0.560000
974978    0.533333
958516    0.447500
255256    0.443333
391553    0.433333
Name: silence_percent_average, dtype: float64
```

```
# Calculate silence for different complex call types
complex_calls_silence = merged_df[merged_df['primary_call_reason'].isin(['Loyalty Inquiry', 'Flight Disruption'])]

complex_calls_silence_avg = complex_calls_silence.groupby('primary_call_reason')['silence_percent_average'].mean()
print("Silence Time for Complex Calls:\n", complex_calls_silence_avg)
```

```
Silence Time for Complex Calls:
 Series([], Name: silence_percent_average, dtype: float64)
```

```
# Determine agent performance by call reason
agent_performance = merged_df.groupby(['agent_id_x', 'primary_call_reason'])['AHT'].mean().sort_values(ascending=False)
print(agent_performance)
```

```
agent_id_x  primary_call_reason
102574      Post Flight            6900.0
121149      Check In               6420.0
136065      Etc                    6300.0
917294      Etc                    5580.0
255256      Booking                5400.0
                                    ...
616988      Other Topics              0.0
292344      Other Topics              0.0
901722      Disability                0.0
541395      Checkout                  0.0
306996      Check In                  0.0
Name: AHT, Length: 6020, dtype: float64
```

```
# Analyzing customer tone for high-priority routing
priority_calls = merged_df[merged_df['customer_tone'] == 'frustrated'].groupby('primary_call_reason')['AHT'].mean().sort_values(ascending=Fa
print("Call Routing Improvement based on Sentiment:\n", priority_calls)
```

```
Call Routing Improvement based on Sentiment:
 primary_call_reason
Etc                     1055.368421
Checkout                 976.675749
Mileage Plus             912.539550
Post Flight              899.284863
Communications           842.813793
Irrops                   782.237288
Products And Services    755.539773
Upgrade                  668.203125
Voluntary Cancel         667.302632
Voluntary Change         658.856877
Check In                 548.307692
Schedule Change          515.384615
Seating                  484.589905
Unaccompanied Minor      482.608696
Booking                  434.308617
Traveler Updates         384.000000
```

```
Digital Support           359.478261
Baggage                   341.184669
Other Topics              313.255814
Disability                303.370787
Name: AHT, dtype: float64
```

```
# Common call reasons that could be moved to self-service
common_self_service_issues = merged_df.groupby('primary_call_reason')['AHT'].mean().sort_values(ascending=False).head(5)
print("Common Issues for Self-Service Improvement:\n", common_self_service_issues)
```

```
Common Issues for Self-Service Improvement:
 primary_call_reason
Checkout          1016.853814
Mileage Plus       995.573406
Etc                962.899160
Post Flight        932.896074
Communications     826.718750
Name: AHT, dtype: float64
```

```
import pandas as pd

# Assuming merged_df has already been created and AHT calculated

# Calculate the average AHT for each agent (using agent_id_x or agent_id_y as appropriate)
agent_aht = merged_df.groupby('agent_id_x')['AHT'].mean().sort_values(ascending=True)

# Identify the best agent
best_agent_id = agent_aht.index[0]  # The agent with the lowest AHT
best_agent_aht = agent_aht.iloc[0]  # Their corresponding AHT

# Output the results
print(f"The best agent is: {best_agent_id} with an average AHT of {best_agent_aht} seconds.")
```

```
The best agent is: 547592 with an average AHT of 180.0 seconds.
```

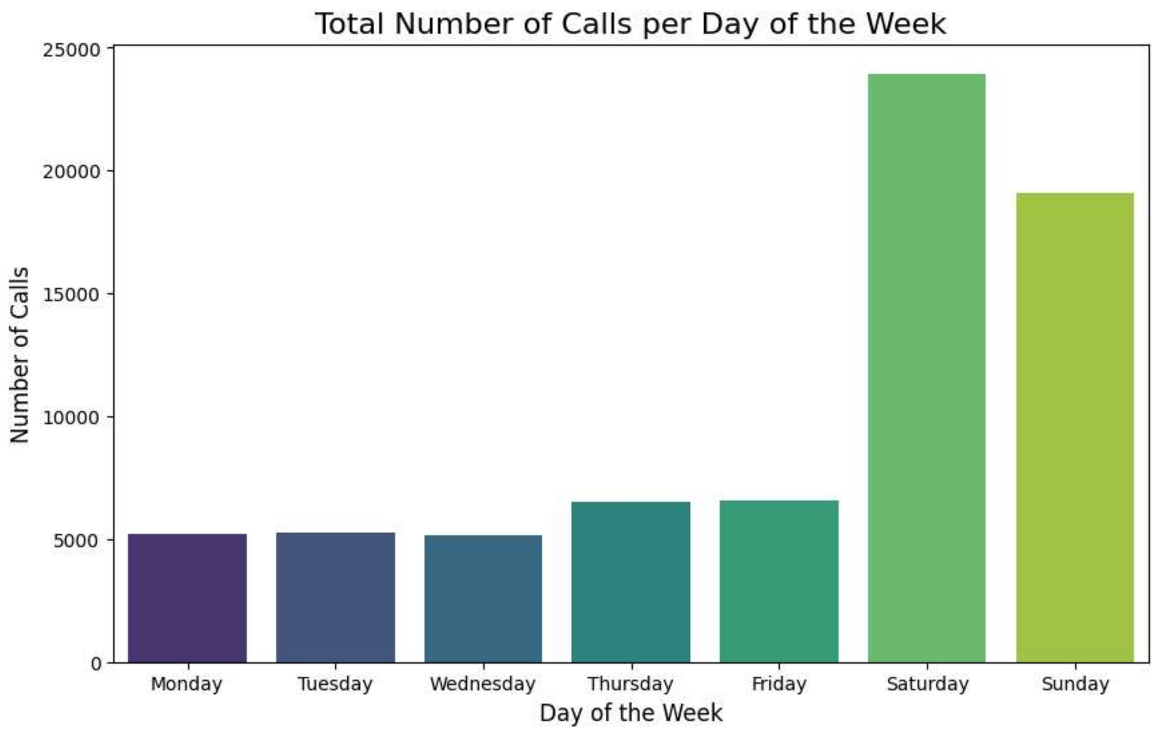
```
# Count total calls handled by each agent
agent_call_counts = merged_df['agent_id_x'].value_counts()

# Combine with average AHT for better insights
agent_summary = pd.DataFrame({
    'Average AHT': agent_aht,
    'Total Calls': agent_call_counts
}).fillna(0)  # Fill NaN values for agents with no calls

# Sort to identify the best agent based on AHT and total calls
print(agent_summary.sort_values(by='Average AHT'))
```

```
             Average AHT  Total Calls
agent_id_x
547592       180.000000            1
616988       285.000000            4
161354       360.000000            2
229129       373.333333            9
676262       390.000000            8
...                 ...          ...
558705      1980.000000            2
140146      2620.000000            3
255256      2620.000000            3
506130      2880.000000            1
102574      3600.000000            2

[383 rows x 2 columns]
```

```
# Step 1: Convert call_start_datetime to the day of the week

merged_df['call_start_datetime'] = pd.to_datetime(merged_df['call_start_datetime'])
merged_df['day_of_week'] = merged_df['call_start_datetime'].dt.day_name()

# Step 2: Count the total number of calls per day of the week
calls_per_day = merged_df.groupby('day_of_week').size().reset_index(name='call_count')

# Step 3: Sort the days of the week in proper order
ordered_days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
calls_per_day['day_of_week'] = pd.Categorical(calls_per_day['day_of_week'], categories=ordered_days, ordered=True)
```

```
calls_per_day = calls_per_day.sort_values('day_of_week')

# Step 4: Plot the bar chart
plt.figure(figsize=(10, 6))
sns.barplot(x='day_of_week', y='call_count', data=calls_per_day, palette='viridis')

# Add labels and title
plt.title('Total Number of Calls per Day of the Week', size=16)
plt.xlabel('Day of the Week', size=12)
plt.ylabel('Number of Calls', size=12)

# Show the plot
plt.show()
```

```
<ipython-input-206-c5a13f826bf9>:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legenc

  sns.barplot(x='day_of_week', y='call_count', data=calls_per_day, palette='viridis')
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need to p
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need to p
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need to p
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need to p
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need to p
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need to p
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a length-1 list-like, you will need to p
  data_subset = grouped_data.get_group(pd_key)
```



```
# Example mapping for days of the week
day_mapping = {
    'Monday': 1,
    'Tuesday': 2,
    'Wednesday': 3,
    'Thursday': 4,
    'Friday': 5,
    'Saturday': 6,
    'Sunday': 7
}

# Apply the mapping to the day_of_the_week column
merged_df['day_of_the_week_numeric'] = merged_df['day_of_week'].map(day_mapping)


# Convert the relevant columns to datetime if they aren't already
merged_df['call_start_datetime'] = pd.to_datetime(merged_df['call_start_datetime'])
merged_df['agent_assigned_datetime'] = pd.to_datetime(merged_df['agent_assigned_datetime'])
merged_df['call_end_datetime'] = pd.to_datetime(merged_df['call_end_datetime'])

# Create the first new column: Time between call start and agent assigned
merged_df['call_start_to_agent_assigned'] = merged_df['agent_assigned_datetime'] - merged_df['call_start_datetime']

# Create the second new column: Time between agent assigned and call end
merged_df['agent_assigned_to_call_end'] = merged_df['call_end_datetime'] - merged_df['agent_assigned_datetime']

# Convert timedelta to seconds for easier interpretation (optional)
merged_df['call_start_to_agent_assigned'] = merged_df['call_start_to_agent_assigned'].dt.total_seconds()
merged_df['agent_assigned_to_call_end'] = merged_df['agent_assigned_to_call_end'].dt.total_seconds()

# Display the updated DataFrame
merged_df[['call_start_datetime', 'agent_assigned_datetime', 'call_end_datetime',
           'call_start_to_agent_assigned', 'agent_assigned_to_call_end']].head()
```

	call_start_datetime	agent_assigned_datetime	call_end_datetime	call_start_to_agent_assigned	agent_assigned_to_call_end
0	2024-07-31 23:56:00	2024-08-01 00:03:00	2024-08-01 00:34:00	420.0	1860.0
1	2024-08-01 00:03:00	2024-08-01 00:06:00	2024-08-01 00:18:00	180.0	720.0
2	2024-07-31 23:59:00	2024-08-01 00:07:00	2024-08-01 00:26:00	480.0	1140.0
3	2024-08-01 00:05:00	2024-08-01 00:10:00	2024-08-01 00:17:00	300.0	420.0
4	2024-08-01 00:04:00	2024-08-01 00:14:00	2024-08-01 00:23:00	600.0	540.0

```
merged_df.columns
```

```
Index(['call_id', 'customer_id', 'agent_id_x', 'call_start_datetime',
       'agent_assigned_datetime', 'call_end_datetime', 'call_transcript',
       'customer_name', 'elite_level_code', 'primary_call_reason',
       'agent_id_y', 'agent_tone', 'customer_tone', 'average_sentiment',
       'silence_percent_average', 'AHT', 'customer_tone_numeric',
       'agent_tone_numeric', 'day_of_week', 'day_of_the_week_numeric',
       'call_start_to_agent_assigned', 'agent_assigned_to_call_end'],
      dtype='object')
```

```
merged_df['elite_level_code'] = pd.to_numeric(merged_df['elite_level_code'], errors='coerce')
# Calculate the median of the elite_level_code column
median_value = np.median(merged_df['elite_level_code'])

# Fill NaN values in the elite_level_code column with the median
merged_df['elite_level_code'] = merged_df['elite_level_code'].fillna(median_value)


merged_df.isna().sum()
```

	0
call_id	0
customer_id	0
agent_id_x	0
call_start_datetime	0
agent_assigned_datetime	0
call_end_datetime	0
call_transcript	0
customer_name	0
elite_level_code	25767
primary_call_reason	5157
agent_id_y	0
agent_tone	217
customer_tone	0
average_sentiment	109
silence_percent_average	0
AHT	0
customer_tone_numeric	0
agent_tone_numeric	0
day_of_week	0
day_of_the_week_numeric	0
call_start_to_agent_assigned	0
agent_assigned_to_call_end	0

dtype: int64

```
merged_df.dropna(inplace=True)
```

```
!pip install scikit-learn==1.3.0  # Install scikit-learn if you haven't already
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
Requirement already satisfied: scikit-learn==1.3.0 in /usr/local/lib/python3.10/dist-packages (1.3.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.3.0) (1.26.4)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.3.0) (1.13.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.3.0) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.3.0) (3.5.0)
```

```
vectorizer = TfidfVectorizer(max_features=10, stop_words='english') # Adjust max_features as needed
```

```
transcript_features = vectorizer.fit_transform(merged_df['call_transcript'].fillna('')).toarray()
```

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# Select features and target variable
features = ['call_id', 'customer_id', 'agent_id_x',
        'elite_level_code',
       'agent_id_y', 'average_sentiment',
       'silence_percent_average', 'AHT', 'customer_tone_numeric',
       'agent_tone_numeric', 'call_start_to_agent_assigned',
       'agent_assigned_to_call_end','day_of_the_week_numeric']
target = 'primary_call_reason'
X = merged_df[features]
y = merged_df[target]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Assuming X_train and X_test are your existing feature matrices
X_train_with_transcript = np.concatenate([X_train, transcript_features[merged_df.index.isin(X_train.index)]], axis=1)
X_test_with_transcript = np.concatenate([X_test, transcript_features[merged_df.index.isin(X_test.index)]], axis=1)


# Create and train the model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_with_transcript, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
from sklearn.metrics import accuracy_score, classification_report

# Make predictions on the test set
y_pred = rf_model.predict(X_test_with_transcript)

# Evaluate the model using classification metrics
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Classification Report:\n{report}")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
Accuracy: 0.39584311883513384
Classification Report:
                      precision    recall  f1-score   support

             Baggage       0.36      0.07      0.12       332
             Booking       0.29      0.25      0.27       307
            Check In       0.00      0.00      0.00       225
            Checkout       0.83      0.53      0.65       208
      Communications       0.39      0.29      0.33       478
     Digital Support       0.00      0.00      0.00       168
          Disability       0.00      0.00      0.00        60
                 Etc       0.25      0.01      0.01       184
              Irrops       0.38      0.74      0.51      1638
        Mileage Plus       0.45      0.45      0.45      1188
        Other Topics       0.00      0.00      0.00        91
         Post Flight       0.42      0.37      0.40       508
Products And Services       0.12      0.00      0.01       414
      Schedule Change       0.00      0.00      0.00        84
             Seating       0.40      0.73      0.51       730
    Traveler Updates       0.77      0.27      0.40       110
  Unaccompanied Minor       0.00      0.00      0.00        11
             Upgrade       0.14      0.01      0.01       336
    Voluntary Cancel       0.00      0.00      0.00       215
    Voluntary Change       0.35      0.41      0.38      1229

            accuracy                           0.40      8516
           macro avg       0.26      0.21      0.20      8516
        weighted avg       0.34      0.40      0.34      8516

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
```

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Select features and target variable (same as before)
features = ['call_id', 'customer_id', 'agent_id_x',
        'elite_level_code',
       'agent_id_y', 'average_sentiment',
       'silence_percent_average', 'AHT', 'customer_tone_numeric',
       'agent_tone_numeric', 'call_start_to_agent_assigned',
       'agent_assigned_to_call_end','day_of_the_week_numeric']
target = 'primary_call_reason'
X = merged_df[features]
y = merged_df[target]

# Split data into training and testing sets (same as before)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```