SKYHACK 2024

LEVERAGING DATA ANALYTICS FOR CALL CENTER OPTIMIZATION AND STRENGTHENING CUSTOMER RELATION WITH UNITED AIRLINES.

Optimizing Operations and Improving Call Center Processes to Enhance the Customer Experience. A Comprehensive Data-Driven Strategy for Streamlining Service Delivery and Empowering Agents to Deliver Efficient, High-Quality Support at Customers of United Airlines.

AAKHYAT BAGGA 2K21/EP/01

KAMAL KUMAR 2K21/EP/48

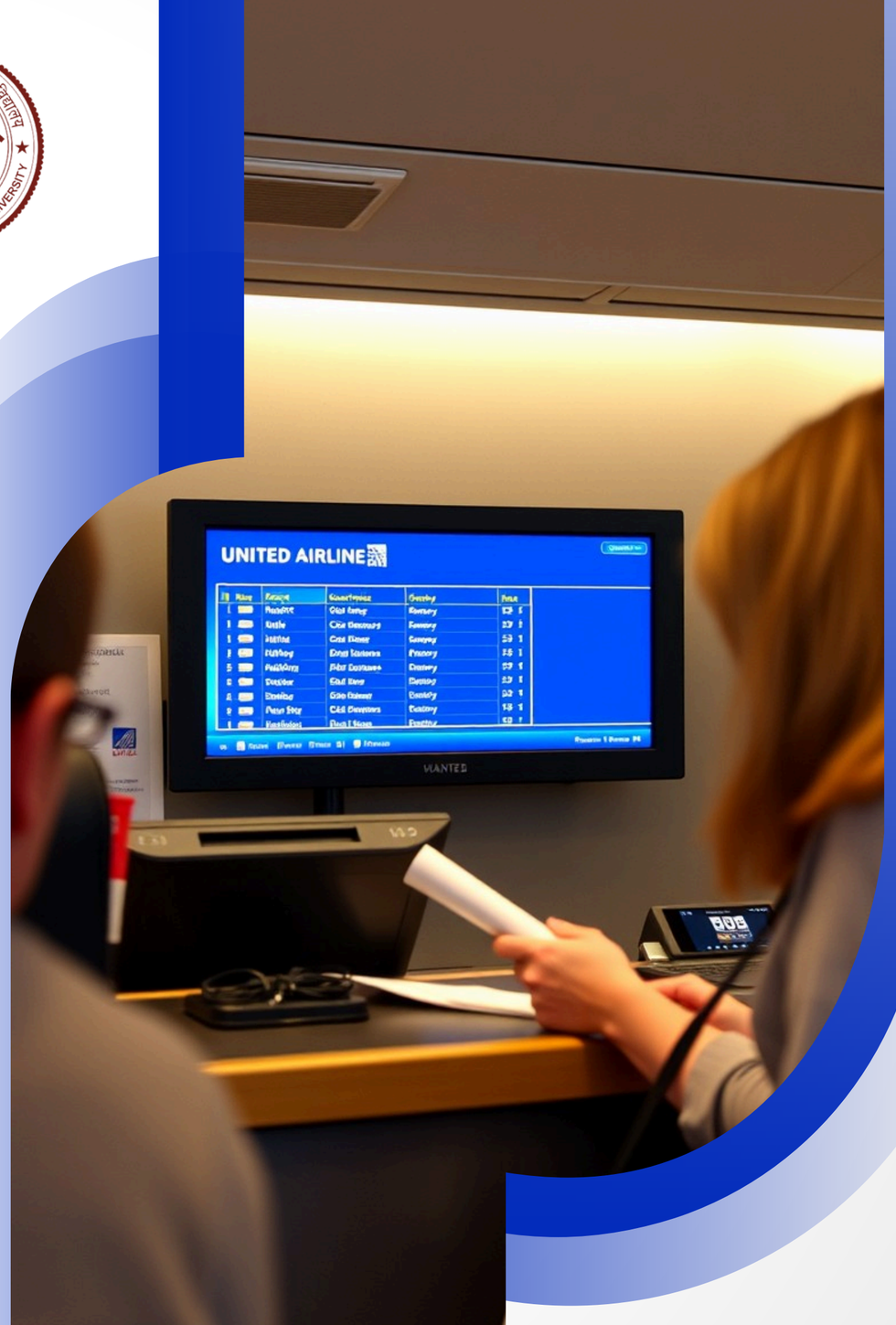


Table of Content

SKYHACK-2024

Problem Statement

United Airlines is committed to setting new standards in aviation. Providing excellent customer service is therefore of utmost importance. An important aspect of this service is the efficiency of our call centre. They play an important role in resolving customer queries and problems quickly and efficiently. However, key indicators such as Average Handling Time (AHT) and Average Answer Speed (AST) still face challenges in optimising them. appropriate Both of which directly affect operational efficiency and customer satisfaction

Elevating the Customer Journey

Aim

Project is to identify and address the factors contributing to the expansion of AHT and AST by analyzing our current call center data. You'll explore how variables such as agent performance, call patterns, and customer sentiment affect the duration of particularly high-demand calls. Our analysis focuses on identifying the key drivers of long-term AHT and AST... to quantify. Percentage difference in average handling time for most and least frequent call reasons.

Problem Statement

United Airlines is committed to providing excellent customer service. Their call centers play a key role in achieving this goal. However, they face challenges with long average handle times (AHT) and slow average response speeds (AST), which has a negative impact on operational efficiency and customer satisfaction .This is compounded by increased self-resolution issues for our agents, increase in their workload and hamper their ability to deal with the increasingly complex needs of the customers... This situation requires focus on two areas: Improving call center efficiency: We need to identify factors that contribute to longer AHT and AST times, especially during peak call periods. It analyzes agent performance, call patterns, and customer sentiment. To identify areas that need improvement Optimizing IVR for self-service: To enable customers to resolve common issues independently. We needed to improve our Interactive Voice Response (IVR) system. This requires analyzing call logs to identify recurring issues suitable for self-service and developing data-driven recommendations to improve ourselves. Select IVR and User Guide.

DELIVERABLES

1. To explore the factors contributing to extended call durations, such as agent performance, call types, and sentiment. Identify key drivers of long AHT and AST, especially during high volume call periods.
2. To analyse the transcripts and call reasons to identify granular reasons associated to recurring problems that could be resolved via self-service options in the IVR system and proposing specific improvements to the IVR options to effectively reduce agent intervention in these cases.
3. To analyze the dataset to uncover patterns that can assist in understanding and identifying these primary call reasons.

Data Description

1. **calls.csv**
   call_id , Agent assgined date time, Call Started date & time , Call ended Date & Time

2. **customers.csv**
   customer elite code and Customer ID.

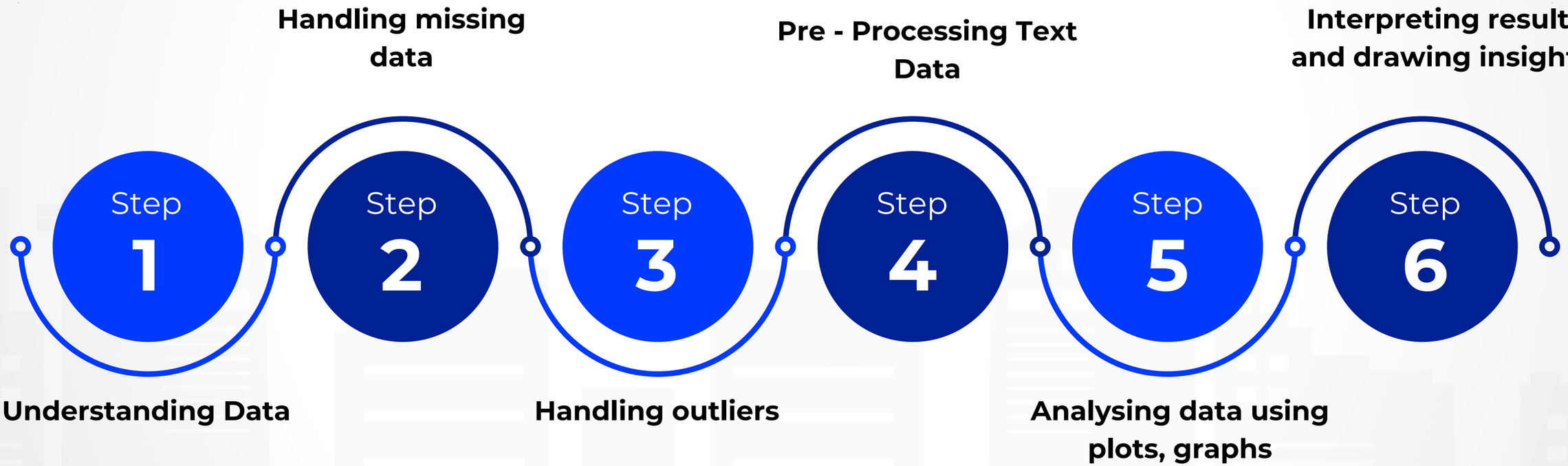
3. **reason.csv**
   call reasons and call ID.

4. **sentiment_statistics.csv**
   agent sentiment , customer sentiment , avg sentiment

Drive Link of Dataset provided.

Source Code

EXPLORATORY DATA ANALYSIS(EDA)



UNDERSTANDING DATA

FEATURES OF THE MERGED DATASET

1. SHAPE

```
[24] df.shape

(71810, 14)
```

2. INFORMATION

```
[25] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 71810 entries, 0 to 71809
Data columns (total 14 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
 7   customer_name           71810 non-null  object
 8   elite_level_code        46043 non-null  float64
 9   primary_call_reason     66653 non-null  object
 10  agent_tone              71593 non-null  object
 11  customer_tone           71810 non-null  object
 12  average_sentiment       71701 non-null  float64
 13  silence_percent_average 71810 non-null  float64
```

3.DESCRIPTION OF DATASET

```
[26] df.describe()
```

	call_id	customer_id	agent_id	elite_level_code	average_sentiment	silence_percent_average
count	7.181000e+04	7.181000e+04	71810.000000	46043.000000	71701.000000	71810.000000
mean	4.993574e+09	5.004334e+09	564768.278039	1.370871	-0.033519	0.285515
std	2.889673e+09	2.884255e+09	257532.362167	1.322297	0.143715	0.192031
min	1.316420e+05	1.197800e+04	102574.000000	0.000000	-1.380000	0.000000
25%	2.480013e+09	2.514618e+09	347606.000000	0.000000	-0.110000	0.130000
50%	4.989448e+09	4.999664e+09	591778.000000	1.000000	-0.020000	0.260000
75%	7.493629e+09	7.509126e+09	786323.000000	2.000000	0.050000	0.410000
max	9.999806e+09	9.999935e+09	993862.000000	5.000000	2.670000	0.980000

```
Number of numerical columns: 6
Number of categorical columns: 8
```

```
Number of distinct agent IDs: 383
Number of distinct customer IDs: 71810
Number of distinct call IDs: 71810
```

CHECKING FOR MISSING VALUES

HANDLING MISSING VALUES

```
1 print("Null Values")
2 print(df.isnull().sum())
```

```
Null Values
call_id                        0
primary_call_reason         5157
customer_id                    0
agent_id                       0
call_start_datetime            0
agent_assigned_datetime        0
call_end_datetime              0
call_transcript                0
customer_name                  0
elite_level_code           25767
agent_tone                   217
customer_tone                  0
average_sentiment            109
silence_percent_average        0
```

```
df.dropna(subset=['primary_call_reason', 'agent_tone', 'average_sentiment'], inplace=True)
```

```
1 print("Null Values")
2 print(df.isnull().sum())
```

```
Null Values
call_id                        0
primary_call_reason            0
customer_id                    0
agent_id                       0
call_start_datetime            0
agent_assigned_datetime        0
call_end_datetime              0
call_transcript                0
customer_name                  0
elite_level_code           23876
agent_tone                     0
customer_tone                  0
average_sentiment              0
silence_percent_average        0
```

Handling Outliers

```python
Q1 = df['call_duration'].quantile(0.25)
Q3 = df['call_duration'].quantile(0.75)
IQR = Q3 - Q1

# Define lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter out outliers
outliers = df[(df['call_duration'] >= lower_bound) & (df['call_duration'] <= upper_bound)]
df=df.drop(outliers.index,axis=0)

plt.figure(figsize=(10, 6))
sns.boxplot(x=df['call_duration'])
plt.title("Boxplot of Call Duration (in minutes)")
plt.show()
```

Figure 1: **Before**

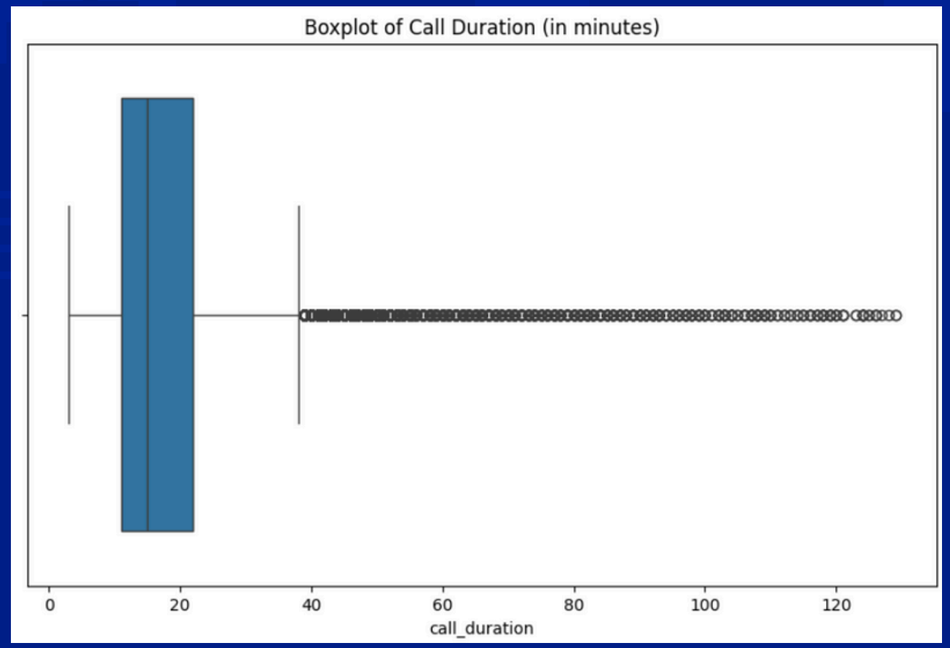
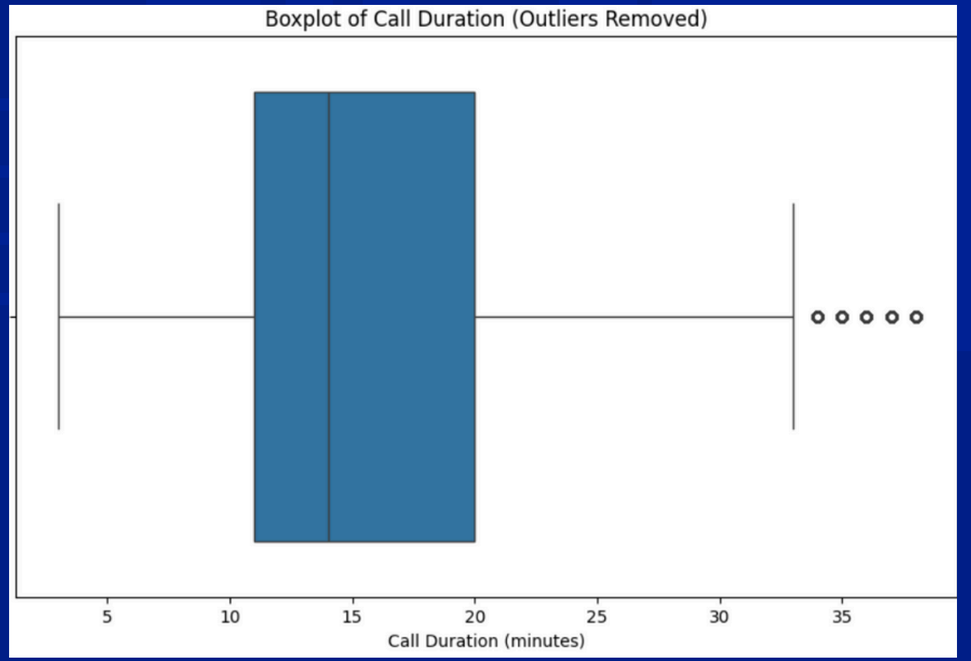


Figure 2: **After**



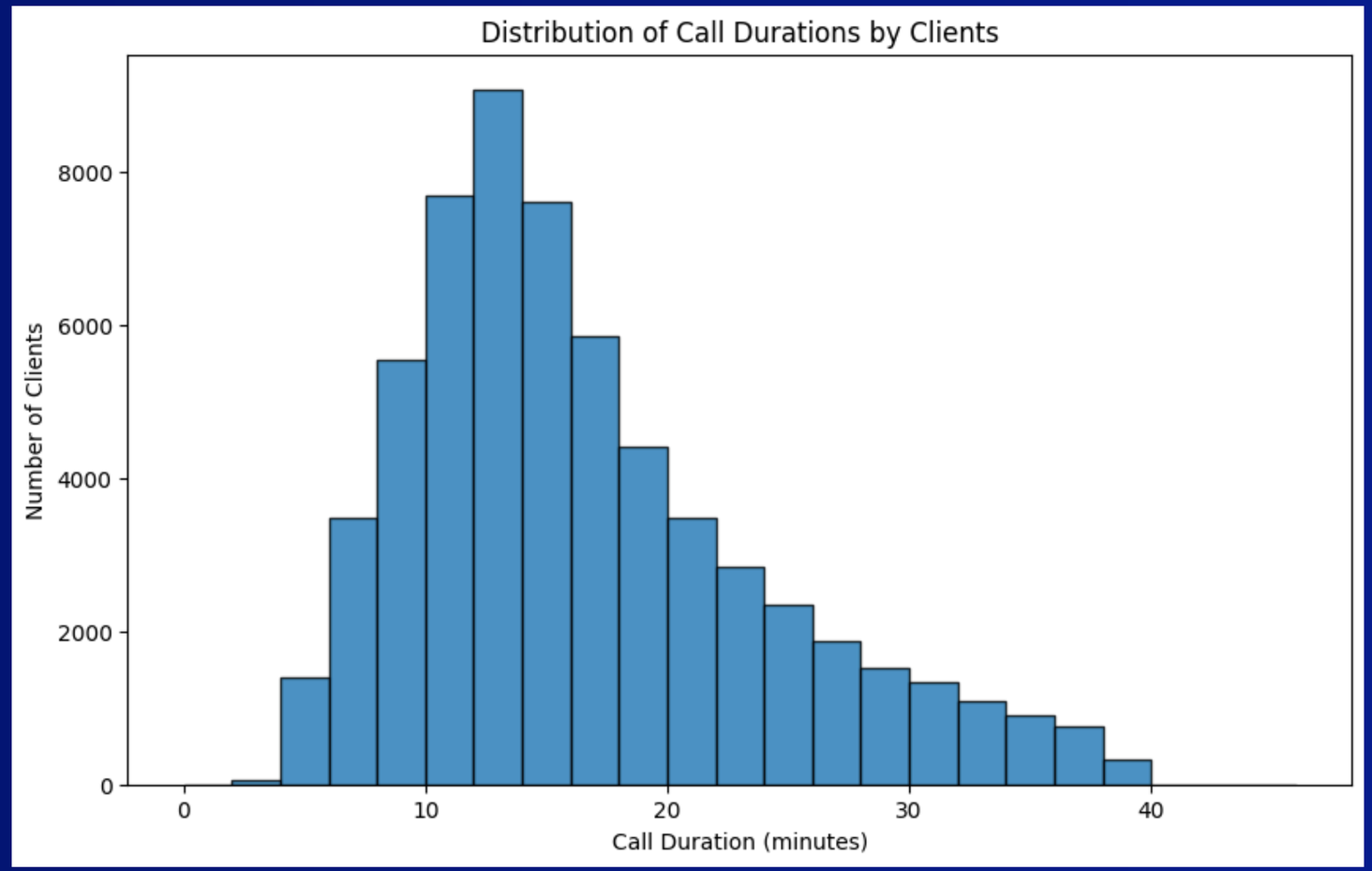
PRE PROCESSING OF CALL TRANSCRIPT

```python
def clean_transcript(text):
    text = text.replace('agent', '').replace('customer', '')
    return text.strip()

# Apply the function to the 'transcript' column
df['cleaned_transcript'] = df['cleaned_transcript'].apply(clean_transcript)

# Display the cleaned transcripts
df[['call_transcript','cleaned_transcript']].head()
```

```python
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
import re

# Download stopwords from nltk
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

# Function to clean the transcripts
def preprocess_transcript(text):
    # Remove special characters, numbers, and lower the text
    text = re.sub(r'[^a-zA-Z0-9]', ' ', text.lower())

    # Tokenize and remove stopwords
    tokens = text.split()
    tokens = [word for word in tokens if word not in stop_words]

    return ' '.join(tokens)

# Apply the preprocessing function to the 'transcript' column
df['cleaned_transcript'] = df['call_transcript'].apply(preprocess_transcript)
```

	call_transcript	cleaned_transcript
0	\n\nAgent: Thank you for calling United Airlin...	thank calling united airlines name jane may he...
1	\n\nAgent: Thank you for calling United Airlin...	thank calling united airlines name mark may he...
2	\n\nAgent: Thank you for calling United Airlin...	thank calling united airlines name steven may ...
3	\n\nAgent: Thank you for calling United Airlin...	thank calling united airlines name amy may hel...
4	\n\nAgent: Thank you for calling United Airlin...	thank calling united airlines service john sp...

Distribution of call duration by Clients



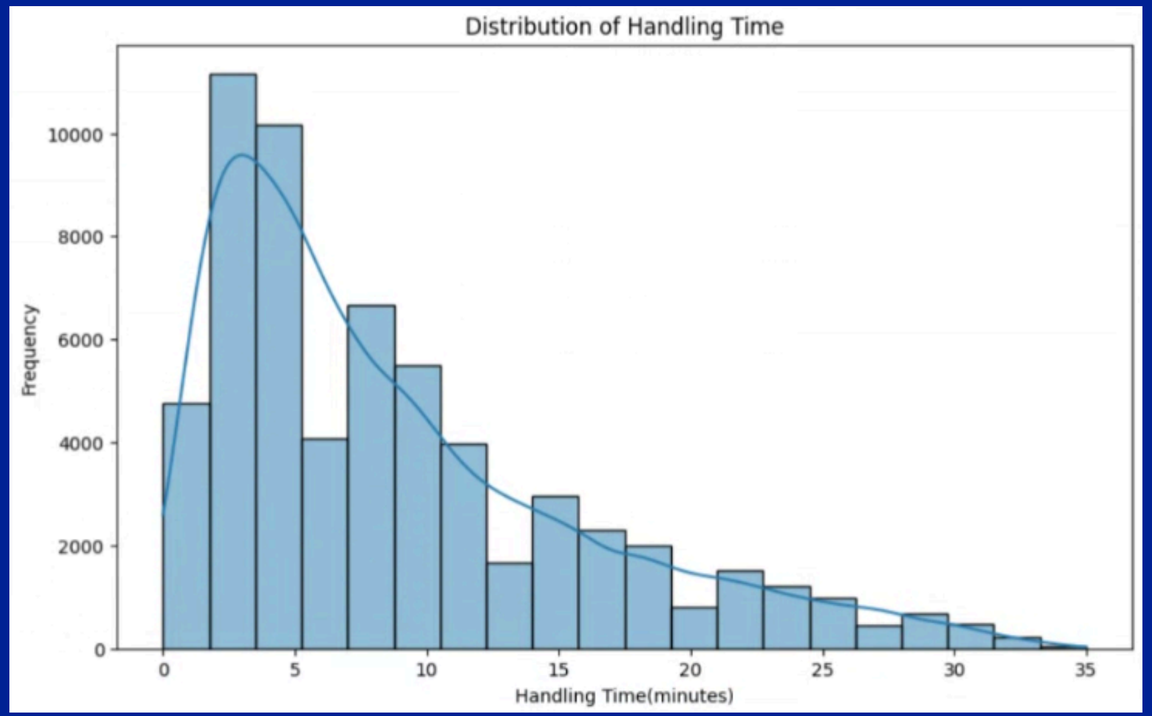
Distribution of Handling Time and Speed to Answer

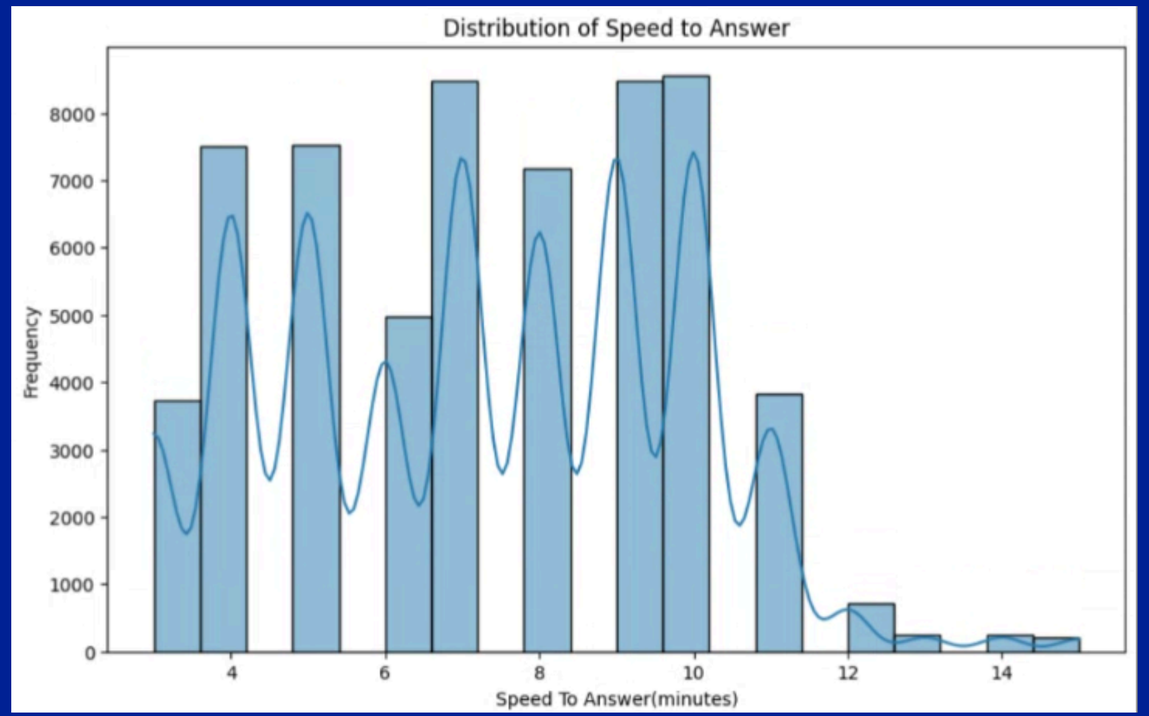
```
1 df[['agent_assigned_datetime','call_end_datetime','Handling_Time']].head()
```

	agent_assigned_datetime	call_end_datetime	Handling_Time
0	2024-08-04 17:55:00	2024-08-04 17:59:00	4.0
1	2024-08-18 07:45:00	2024-08-18 07:50:00	5.0
2	2024-08-10 12:08:00	2024-08-10 12:15:00	7.0
3	2024-08-11 11:25:00	2024-08-11 11:39:00	14.0
4	2024-08-06 16:09:00	2024-08-06 16:15:00	6.0

```
1 df[['call_start_datetime','agent_assigned_datetime','Speed_To_Answer']].head()
```

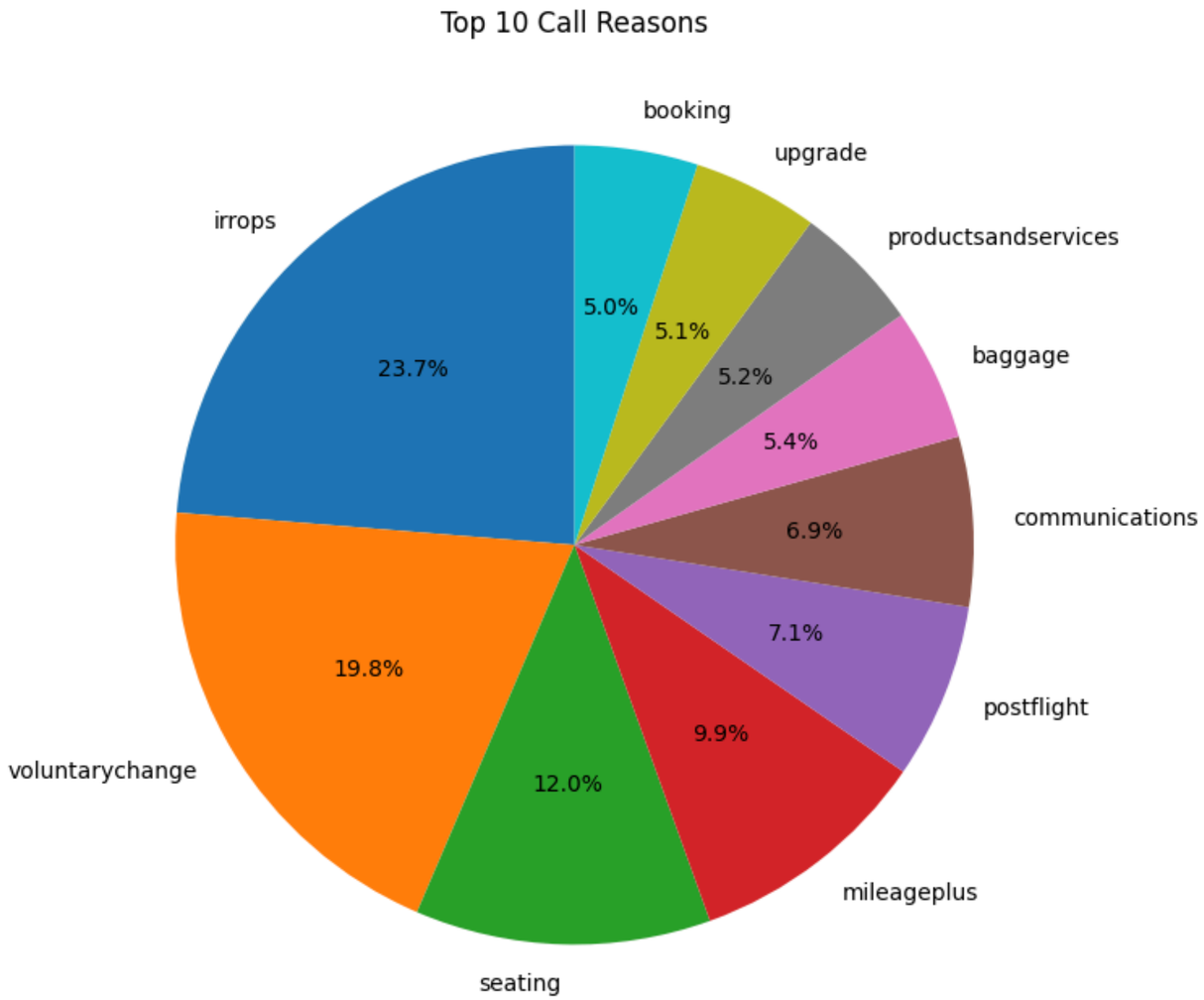
	call_start_datetime	agent_assigned_datetime	Speed_To_Answer
0	2024-08-04 17:46:00	2024-08-04 17:55:00	9.0
1	2024-08-18 07:41:00	2024-08-18 07:45:00	4.0
2	2024-08-10 11:59:00	2024-08-10 12:08:00	9.0
3	2024-08-11 11:15:00	2024-08-11 11:25:00	10.0
4	2024-08-06 15:58:00	2024-08-06 16:09:00	11.0





Analysis of Call Reasons

primary_call_reason	count
irrops	13311
voluntarychange	10848
seating	6365
mileageplus	5851
postflight	4330
communications	3840
productsandservices	2856
baggage	2832
upgrade	2738
booking	2637

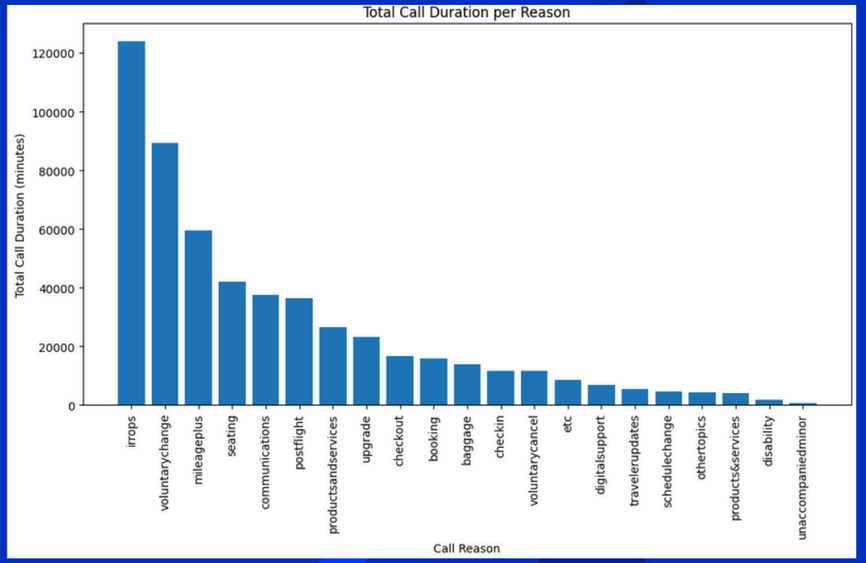


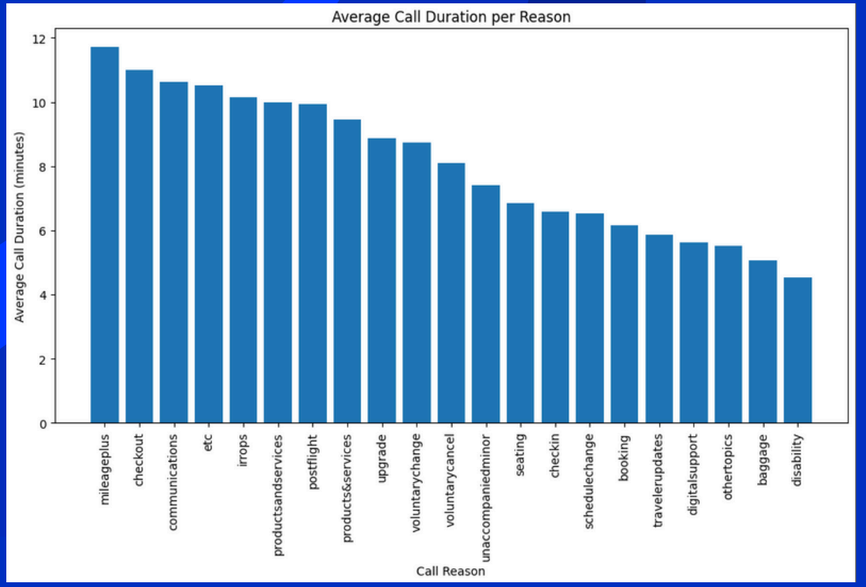
Analysis of AHT on the basis of Call Reasons

```python
# Calculate average call duration for each call reason
avg_duration_per_reason = df.groupby('primary_call_reason')['Handling_Time'].mean().sort_values(ascending=False)

# Calculate total call duration for each call reason
total_duration_per_reason = df.groupby('primary_call_reason')['Handling_Time'].sum().sort_values(ascending=False)

# Display the results
print("Average Call Duration per Reason:\n")
print(avg_duration_per_reason)

print("\nTotal Call Duration per Reason:\n")
print(total_duration_per_reason)
```

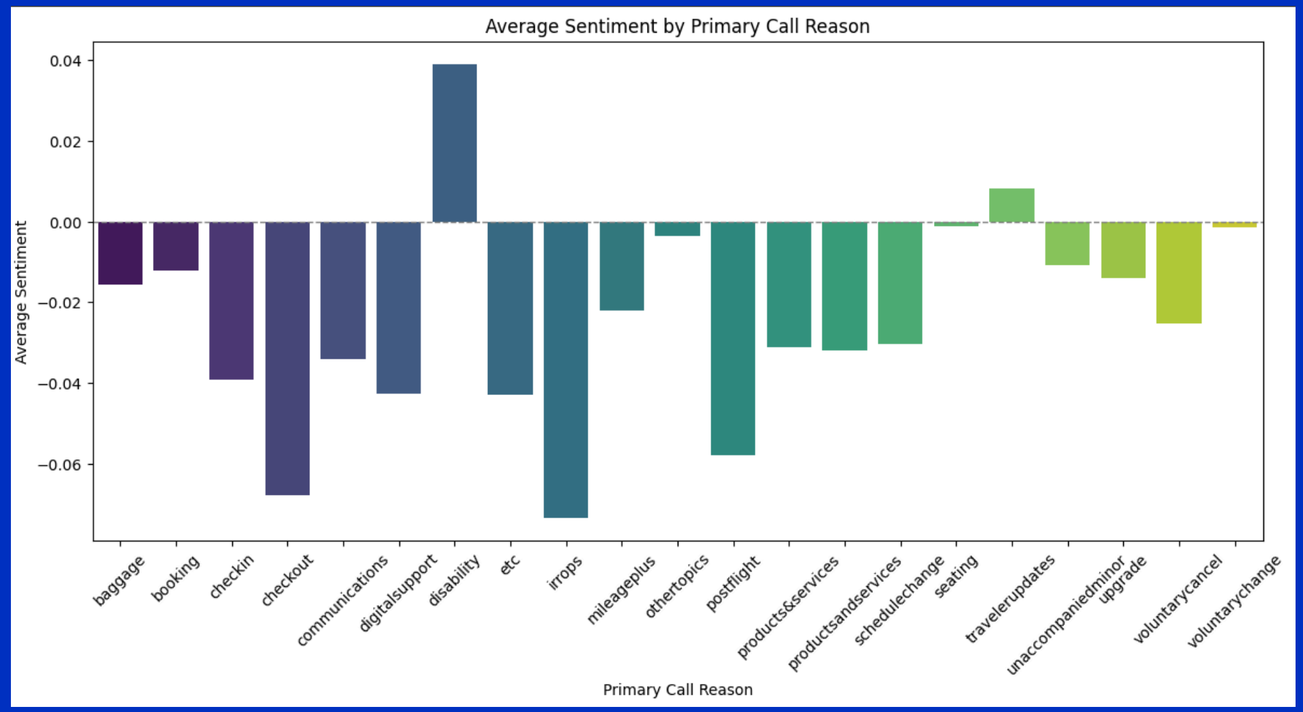
```
Total Call Duration per Reason:

primary_call_reason
irrops                123845.0
voluntarychange        89226.0
mileageplus            59621.0
seating                42194.0
communications         37559.0
postflight             36380.0
productsandservices    26605.0
upgrade                23247.0
checkout               16880.0
booking                15838.0
baggage                14041.0
checkin                11773.0
voluntarycancel        11732.0
etc                     8469.0
digitalsupport          6754.0
travelerupdates         5373.0
schedulechange          4577.0
othertopics             4464.0
products&services       4214.0
disability              1806.0
unaccompaniedminor       740.0
Name: Handling_Time, dtype: float64
```

```
Average Call Duration per Reason:

primary_call_reason
mileageplus           11.706460
checkout              11.003911
communications        10.621889
etc                   10.533582
irrops                10.144577
productsandservices    9.983114
postflight             9.937176
products&services      9.448430
upgrade                8.869515
voluntarychange        8.736512
voluntarycancel        8.113416
unaccompaniedminor     7.400000
seating                6.845230
checkin                6.584452
schedulechange         6.519943
booking                6.167445
travelerupdates        5.878556
digitalsupport         5.633028
othertopics            5.517923
baggage                5.076283
disability             4.537688
Name: Handling_Time, dtype: float64
```





Analysis of Customer Sentiments on Call Reasons




```
df['primary_call_reason'] = df['primary_call_reason'].astype('category')
plt.figure(figsize=(14, 6))
average_sentiment_per_reason = df.groupby('primary_call_reason')['average_sentiment'].mean().reset_index()
sns.barplot(x='primary_call_reason', y='average_sentiment', data=average_sentiment_per_reason, palette='viridis'
plt.title("Average Sentiment by Primary Call Reason")
plt.xlabel("Primary Call Reason")
plt.ylabel("Average Sentiment")
plt.xticks(rotation=45)
plt.axhline(0, color='grey', lw=1, ls='--')  # Line for neutral sentiment
plt.show()
```

DATA INSIGHTS FROM CALL REASONS

From the data analysis we can draw following conclusions about the data:

- The most common call reasons are **IRROPS (23.7%) , Voluntary Changes (19.8%) and Seating(12%).** These take up almost 50% of the calls received.
- **IRROPS**, **Voluntary Change**, **Mileage Plus**, and **Seating** require more time due to their complexity, such as rebooking flights, negotiating seat changes, or managing loyalty rewards.
- **Mileage Plus**, **Checkout**, and **Communications** have the longest average call durations, suggesting these topics are more complicated and time-consuming to resolve.
- Reasons like **Voluntary Cancellation Requests** are not among the most frequent requests however they are among the most time consuming requests.

RECOMMENDATIONS BASED ON DATA INSIGHTS

1. Subscription Tiers

Customers are categorized into four subscription levels: Gold, Silver, Bronze, and None. Each tier has a base priority score: Gold members start with the highest priority, followed by Silver, Bronze, and None.

2.Call Type Importance

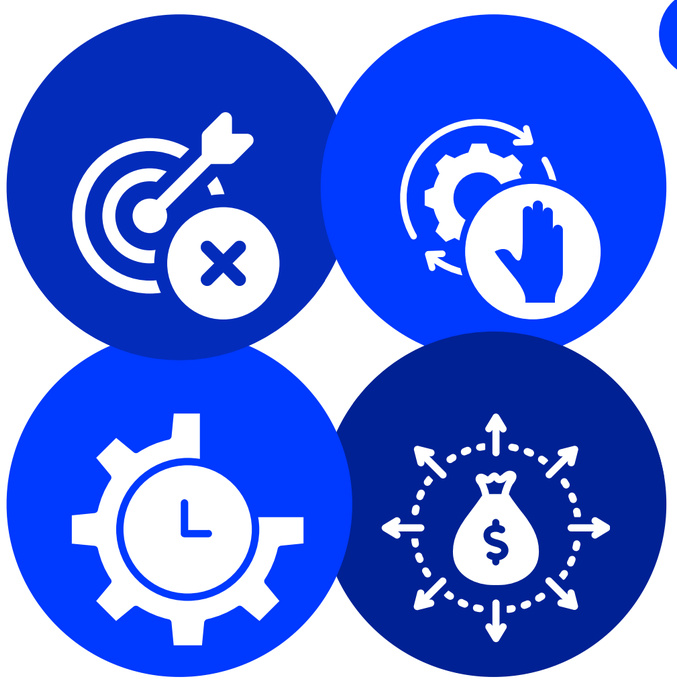
Different reasons for calling have assigned priority points. For example, calls about flight cancellations are the highest priority, followed by rescheduling, seat changes, and flight delays.

3. Total Priority Score Calculation

The total priority score for each call is determined by adding the base score from the subscription tier, the waiting time points, and the points assigned to the reason for the call.

4.Dynamic Prioritization

This system ensures that high-tier members are prioritized, while also taking into account how long customers have been waiting and the urgency of their inquiries. This helps improve the overall efficiency of customer service.



5. Enhance Self-Service

Offer automated tools via apps or websites for frequent inquiries like seating, loyalty program management, and voluntary changes, allowing customers to resolve issues on their own.

6. Automate Simple IRROPS Solutions

Use automation for rebooking and compensation during irregular operations to reduce both call volume and agent handling time.

7.Proactive Communication

Use proactive outreach (SMS, email, or app notifications) to keep loyalty members informed during flight changes or disruptions, reducing the need for calls.

AGENT PERFORMANCE ANALYSIS:

We performed data analysis on agents data using following parameter:

- **Average duration of call for each agent**: If the average call duration for an agent is large then we can conclude that the agent is not efficient enough. We analyze this by categorizing each agent on daily traffic he faces

- **Total no of customers each agent caters to**: We identify the no of customers each agent handles. If the no of Customers are unequally distributed it will affect speed to respond time, as it would indicate at a given moment some agents are more busier than other due to which more customers will be in waiting queue

- **Sentiment of agent during the call**: Based on the agent tone ie calm angry frustrated etc we will identify performance of agent as agent tone affects the overall handling time. This also effects the customer satisfaction and can be used as a metric to improve

Analysing AHT as per agent

1 Average_Handle_Time

	agent_id	Handling_Time
0	102574	5.000000
1	103651	9.042802
2	106000	9.107527
3	107876	13.000000
4	112609	9.000000
...	...	...
377	981776	8.498054
378	981779	8.730964
379	982131	9.243478
380	992521	11.131579
381	993862	8.703125

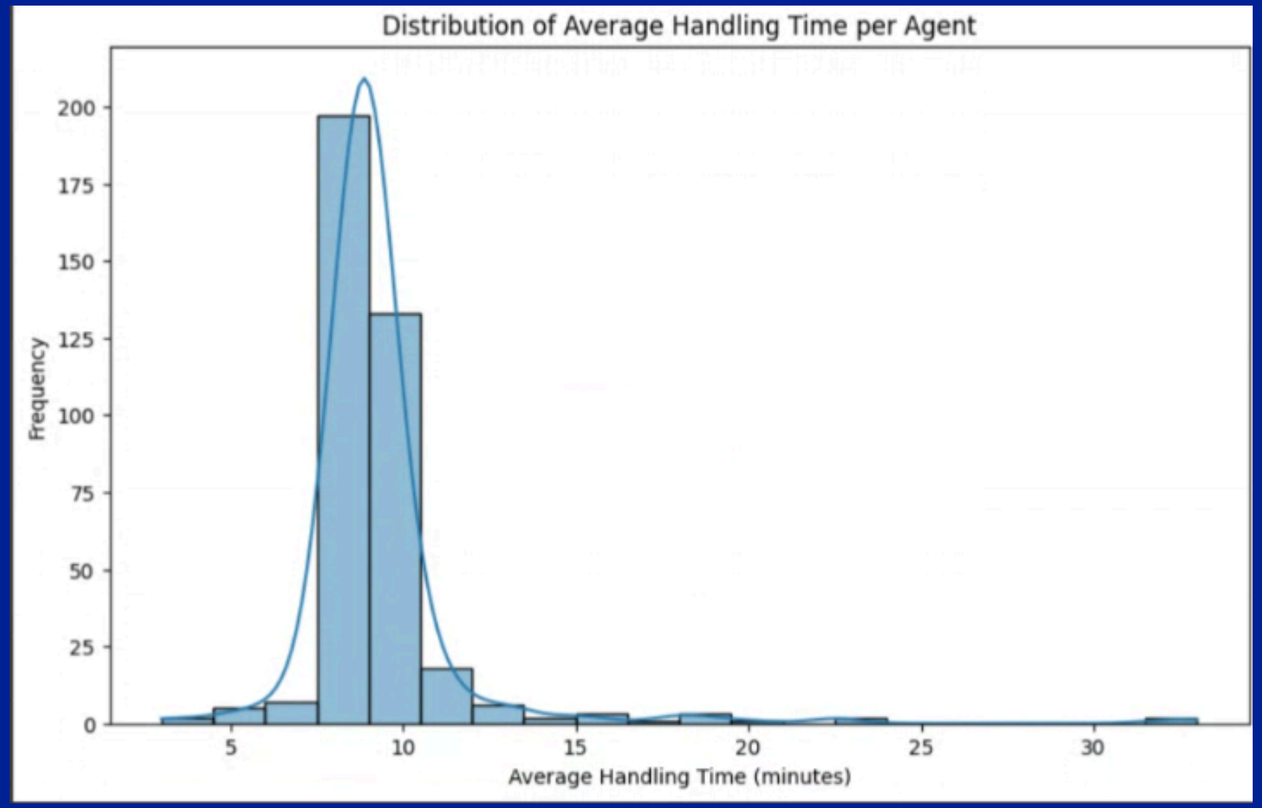
382 rows × 2 columns

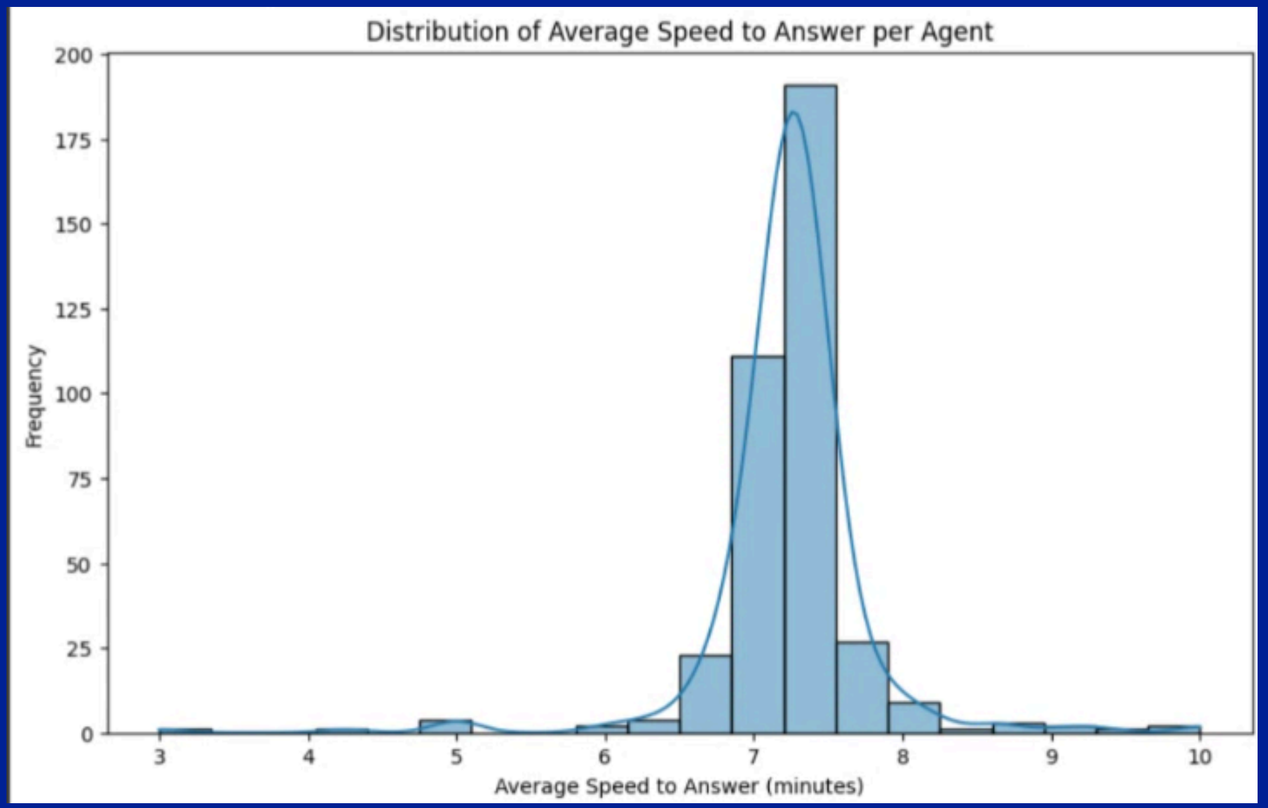
Analysing AST as per agent

1 AST

	agent_id	Speed_To_Answer
0	102574	7.000000
1	103651	6.964981
2	106000	7.354839
3	107876	7.200000
4	112609	7.714286
...	...	...
377	981776	7.077821
378	981779	7.248731
379	982131	7.356522
380	992521	7.368421
381	993862	7.292969

382 rows × 2 columns



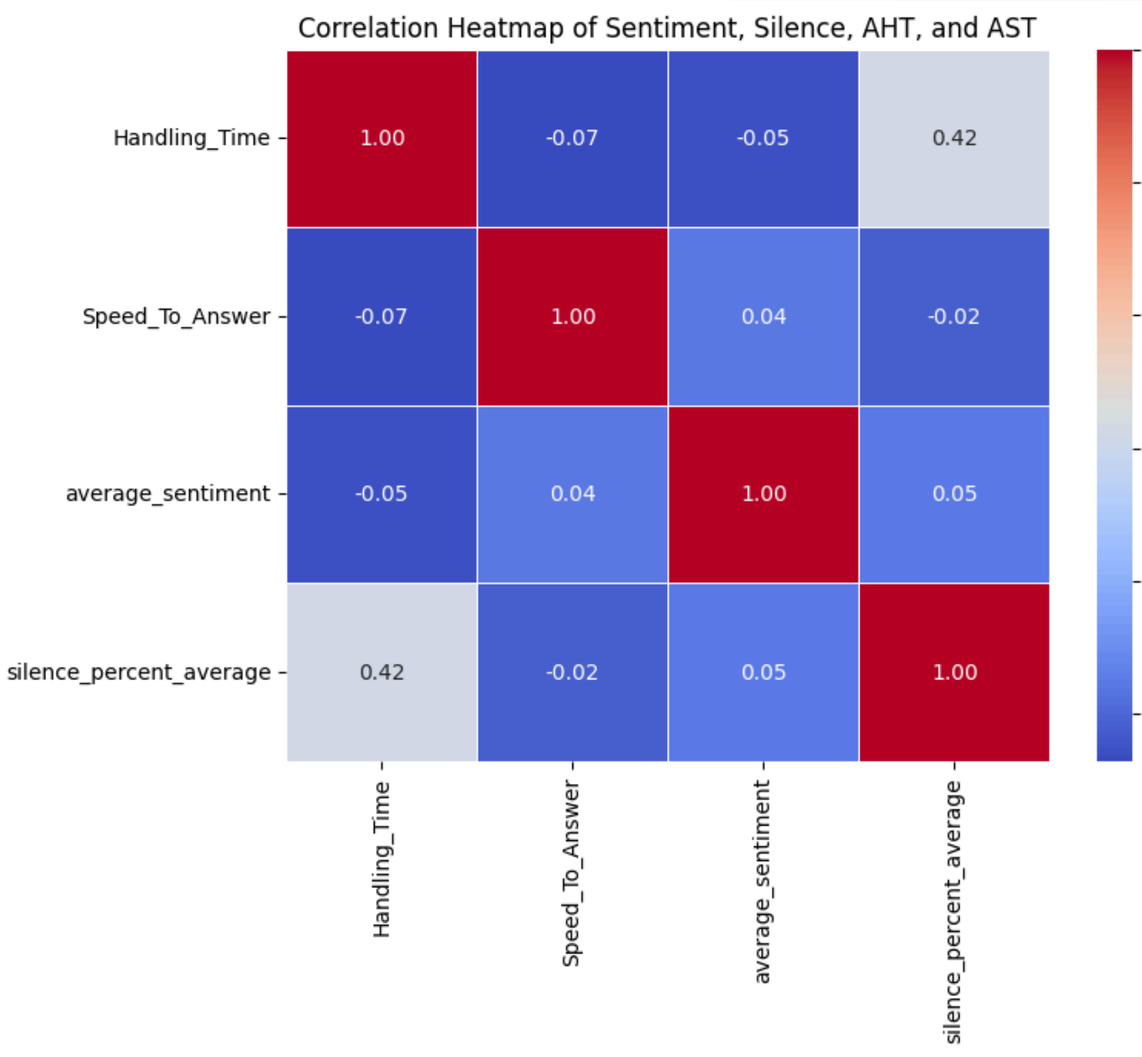


Analysis of agent sentiment and how it affects AHT,AST and effect on customer satisfaction

Impact of Agent Tone on Average Handling Time

```
[37]
agent_emotion_impact = df.groupby('agent_tone').agg(
    average_aht=('Handling_Time', 'mean'),
    call_count=('call_id', 'count')
).reset_index()

# Display the result
agent_emotion_impact
```

	agent_tone	average_aht	call_count
0	angry	5.343085	376
1	calm	8.649578	23446
2	frustrated	7.814654	3453
3	neutral	9.117784	39063
4	polite	3.666667	87



COMPREHENSIVE ANALYSIS OF AGENT TONE IMPACT ON KEY PERFORMANCE METRICS LIKE HANDLING TIME,SPEED TO TIME AVG. SENTIMENT

```python
agent_emotion_impact = df.groupby('agent_tone').agg(
    average_aht=('Handling_Time', 'mean'),
    average_ast=('Speed_To_Answer', 'mean'),
    sentiment_score=('average_sentiment', 'mean'),
    silence_avg=('silence_percent_average', 'mean')
).reset_index()


# Display the result
agent_emotion_impact
```

	agent_tone	average_aht	average_ast	sentiment_score	silence_avg
0	angry	5.343085	7.050532	-0.616915	0.183723
1	calm	8.649578	7.380491	0.103344	0.281470
2	frustrated	7.814654	7.022879	-0.330802	0.238430
3	neutral	9.117784	7.213629	-0.079731	0.271331
4	polite	3.666667	7.689655	0.673793	0.256552

```python
sentiment_counts = sentiment_data.groupby(['agent_id', 'agent_tone']).size().unstack(fill_value=0)

avg_sentiment_per_agent = sentiment_data.groupby('agent_id')['average_sentiment'].mean()


agent_sentiment_analysis = pd.merge(sentiment_counts, avg_sentiment_per_agent, on='agent_id')
agent_sentiment_analysis
```

agent_id	angry	calm	frustrated	neutral	polite	average_sentiment
102574	0	1	0	1	0	-0.075000
103651	3	102	13	180	0	-0.027953
106000	0	36	3	74	0	-0.032124
107876	0	1	0	5	0	-0.065000
112609	0	5	0	6	0	-0.020909
...	...	...	...	...	...	...
981776	1	109	20	178	1	-0.026861
981779	1	68	12	145	1	-0.034714
982131	2	84	15	168	0	-0.038815
992521	0	9	2	34	1	-0.016957
993862	3	106	16	172	0	-0.035488

CATEGORISING AGENTS INTO GOOD AND BAD AGENTS BASED ON AGENT TONE AND AVERAGE SENTIMENT

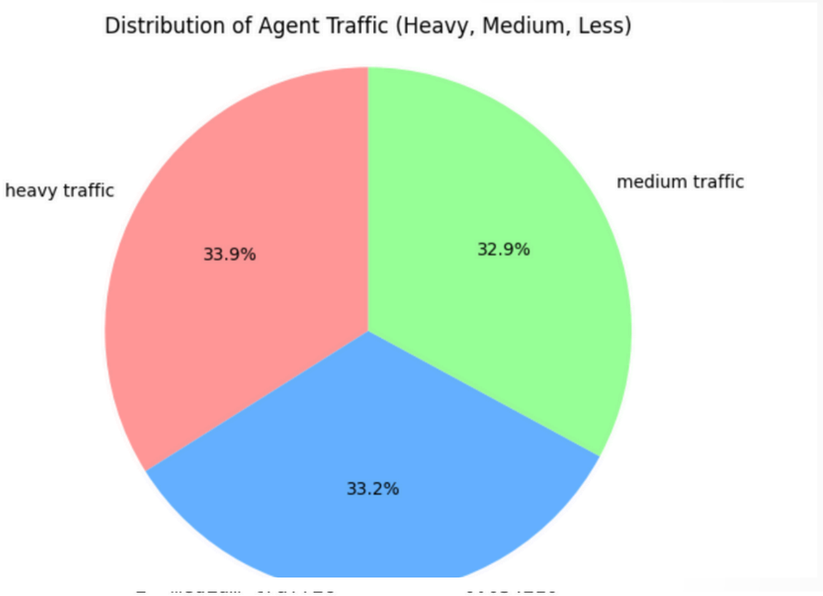
```
agent_sentiment_analysis['best_agent_score'] = agent_sentiment_analysis['calm'] + agent_sentiment_analysis['po


agent_sentiment_analysis['worst_agent_score'] = agent_sentiment_analysis['angry'] - agent_sentiment_analysis['

best_agents = agent_sentiment_analysis.sort_values(by='best_agent_score', ascending=False).head(10)
worst_agents = agent_sentiment_analysis.sort_values(by='worst_agent_score', ascending=False).head(10)


print("Best Agents:")
print(best_agents)

print("\nWorst Agents:")
print(worst_agents)
```

```
Best Agents:
          angry  calm  frustrated  neutral  polite  average_sentiment  \
agent_id
784648        4   172          22      216       0          -0.023188
251287        5   167          19      294       0          -0.036543
757349        2   166          25      267       0          -0.031978
542034        3   158          18      263       0          -0.033747
758927        2   154          17      207       1          -0.021732
524049        1   154          22      225       0          -0.032655
931734        2   153          31      250       1          -0.043333
594957        3   153          33      289       0          -0.048938
390819        0   153          19      208       0          -0.013500
537208        5   152          20      230       0          -0.031499

          best_agent_score  worst_agent_score
agent_id
784648                 172           4.023188
251287                 167           5.036543
757349                 166           2.031978
542034                 158           3.033747
758927                 155           2.021732
524049                 154           1.032655
931734                 154           2.043333
594957                 153           3.048938
390819                 153           0.013500
537208                 152           5.031499
```

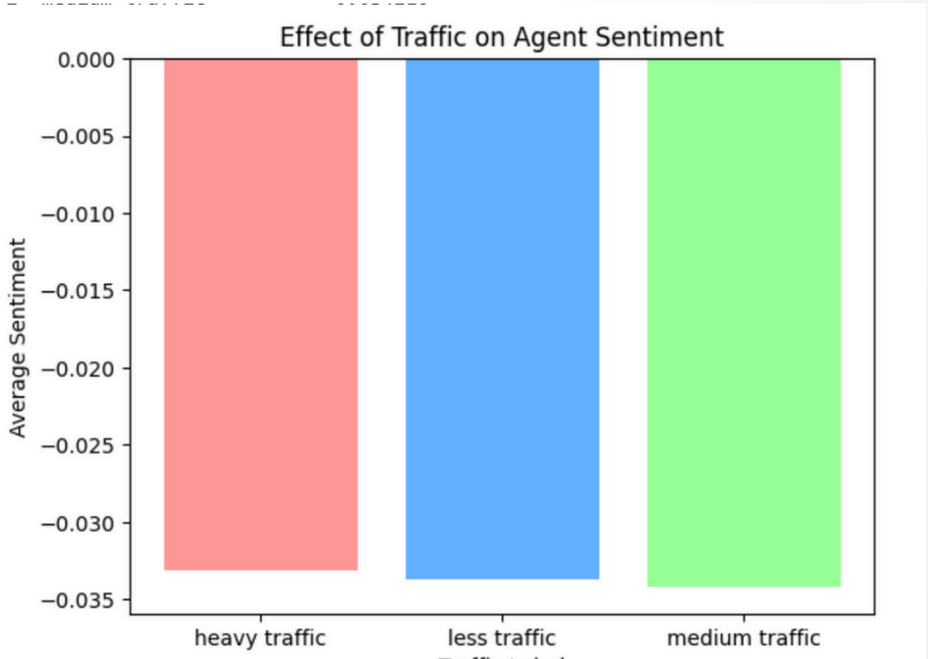
```
Worst Agents:
          angry  calm  frustrated  neutral  polite  average_sentiment  \
agent_id
600666        6   147          25      239       0          -0.041727
215457        6   131          17      236       1          -0.035751
607742        6   150          24      230       1          -0.033504
265196        5    69          16      149       1          -0.060417
266799        5    87          19      148       0          -0.040927
251287        5   167          19      294       0          -0.036543
149315        5   131          16      272       1          -0.035188
537208        5   152          20      230       0          -0.031499
311571        4    24           7       50       0          -0.072941
510903        4    35           9       66       0          -0.063684

          best_agent_score  worst_agent_score
agent_id
600666                 147           6.041727
215457                 132           6.035751
607742                 151           6.033504
265196                  70           5.060417
266799                  87           5.040927
251287                 167           5.036543
149315                 132           5.035188
537208                 152           5.031499
311571                  24           4.072941
510903                  35           4.063684
```

ANALYSIS OF DAILY TRAFFIC FACED BY AN AGENT AND ITS IMPACT ON AVG. SENTIMENT

```python
low_threshold = agent_traffic['total_calls'].quantile(0.33)
high_threshold = agent_traffic['total_calls'].quantile(0.66)


def label_traffic(calls):
    if calls <= low_threshold:
        return 'less traffic'
    elif calls <= high_threshold:
        return 'medium traffic'
    else:
        return 'heavy traffic'


agent_traffic['traffic_label'] = agent_traffic['total_calls'].apply(label_traffic)

traffic_distribution = agent_traffic['traffic_label'].value_counts()

print(traffic_distribution)

plt.figure(figsize=(6, 6))
plt.pie(traffic_distribution, labels=traffic_distribution.index, autopct='%1.1f%%', startangle=90, colors=['#
plt.title('Distribution of Agent Traffic (Heavy, Medium, Less)')
plt.axis('equal')
```




```python
agent_sentiment_traffic = pd.merge(agent_traffic, sentiment_data, on='agent_id')


traffic_sentiment_analysis = agent_sentiment_traffic.groupby('traffic_label')['average_sentiment'].mean().rese


print(traffic_sentiment_analysis)


import matplotlib.pyplot as plt

plt.bar(traffic_sentiment_analysis['traffic_label'], traffic_sentiment_analysis['average_sentiment'], color=['
plt.title('Effect of Traffic on Agent Sentiment')
plt.xlabel('Traffic Label')
plt.ylabel('Average Sentiment')
plt.show()
```



INFERENCE ON AGENT PERFORMANCE AS DRIVER OF LONG AHT.

On analysing the data we make the following inferences:

- **Call Sentiment and Agent Performance:** The majority of calls had a neutral and calm tone, leading to a low average sentiment score. Negative emotions such as anger and frustration have a significant impact on Average Handling Time (AHT). The correlation matrix shows that agents who exhibit anger and frustration tend to have a higher AHT.
- **Agent Categorization:**To enhance customer satisfaction, agents were classified into two categories: Good Agents (efficient) and Bad Agents (inefficient) based on their performance metrics**.**
- **Call Duration as a Metric:** Another metric analyzed was the average call duration. Agents with higher call durations are categorized as inefficient, as longer calls often indicate performance issues.
- **Customer Distribution:** The analysis also revealed an unequal distribution of customers across agents, which influences Average Speed of Answer (ASA) and overall agent performance.
- **Influence of Customer Traffic on sentiments:** The customer traffic analysis showed that on higher customer traffic, Average sentiment of the call reduces

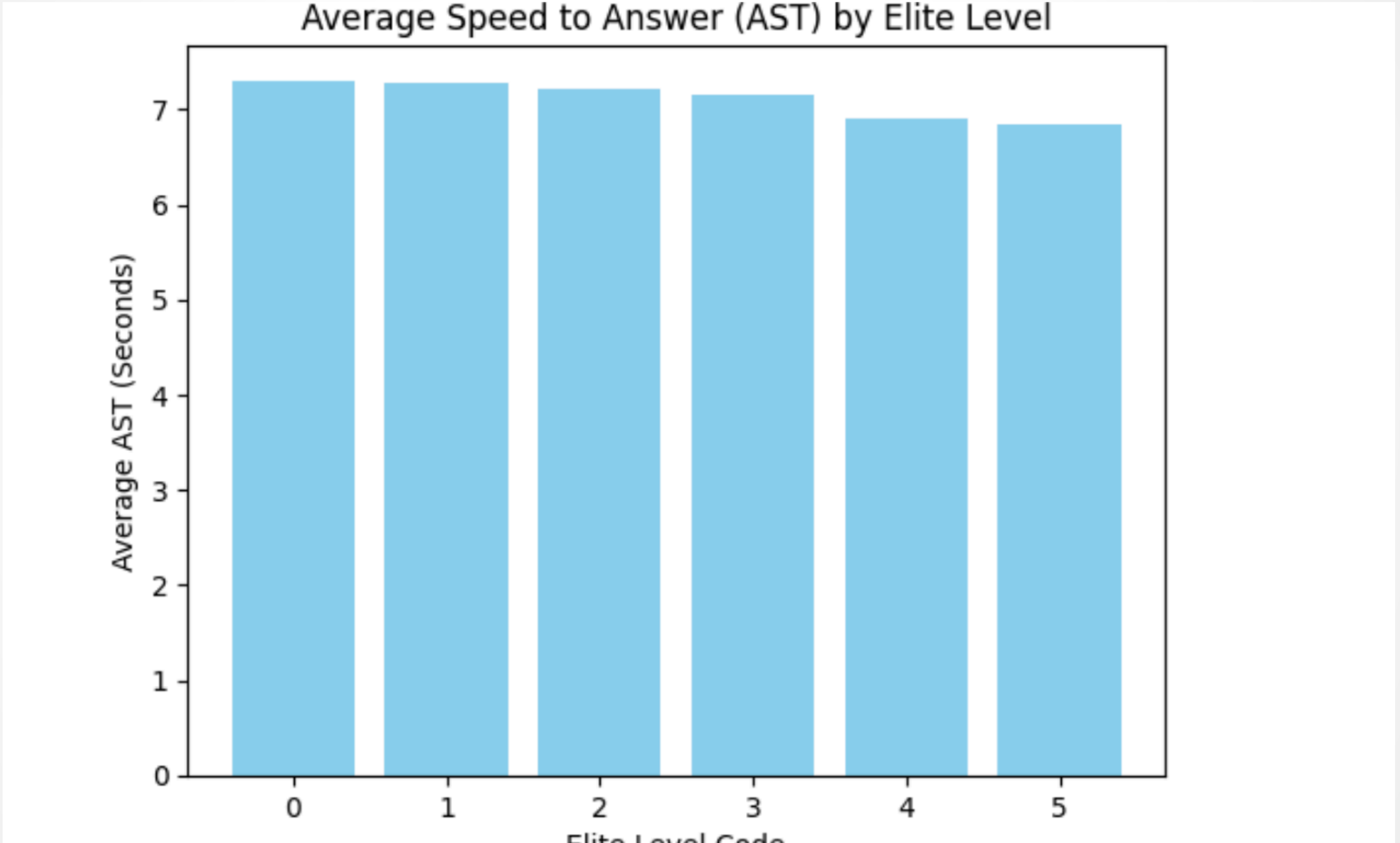
RECOMENDATIONS BASED ON DATA INSIGHTS

Based on the following data insights we can make following data reccomendations:

- Conduct a monthly analysis of agent performance by evaluating agent sentiment and average call duration for each call. By identifying agents who consistently exhibit negative emotions like anger or frustration (which correlate with higher AHT), corrective measures, such as targeted training or coaching, can be implemented. We can do this by creating dashboard to track sentiment scores and call duration per agent.
- Address the issue of unequal distribution of customer calls. Implement a system that dynamically tracks agent availability and workload. Agents who are free for a duration longer than a defined threshold should be prioritized for customer assignment.
- Use backend tools to monitor agent idle time and call handling capacity in real-time.
- Establish a smart routing system that automatically queues incoming calls to agents with the lowest call load or longest idle time.
- Introduce a performance-based incentive program that rewards agents who maintain low AHT while keeping high sentiment scores.
- Use a backend algorithm that tracks real-time availability and agent performance metrics to queue customers to the most suitable agent.
- Adjust the call queue dynamically based on both the complexity of the issue and the agent's expertise.
- Assign workload based on agents performance, calmer and polite agents get high priority and important calls.

ANALYSIS OF CUSTOMER ELITE CODE AND HOW IT AFFECTS AST

```
ast_by_elite_level = df.groupby('elite_level_code')['Speed_To_Answer'].mean().reset_index()

# Display the result
print(ast_by_elite_level)

# Visualization: Average AST by elite level
plt.bar(ast_by_elite_level['elite_level_code'], ast_by_elite_level['Speed_To_Answer'], color='skyblue')
plt.title('Average Speed to Answer (AST) by Elite Level')
plt.xlabel('Elite Level Code')
plt.ylabel('Average AST (Seconds)')
plt.show()
```


Average Speed to Answer (AST) by Elite Level
Average AST (Seconds)
0
1
2
3
4
5
6
7
0
1
2
3
4
5

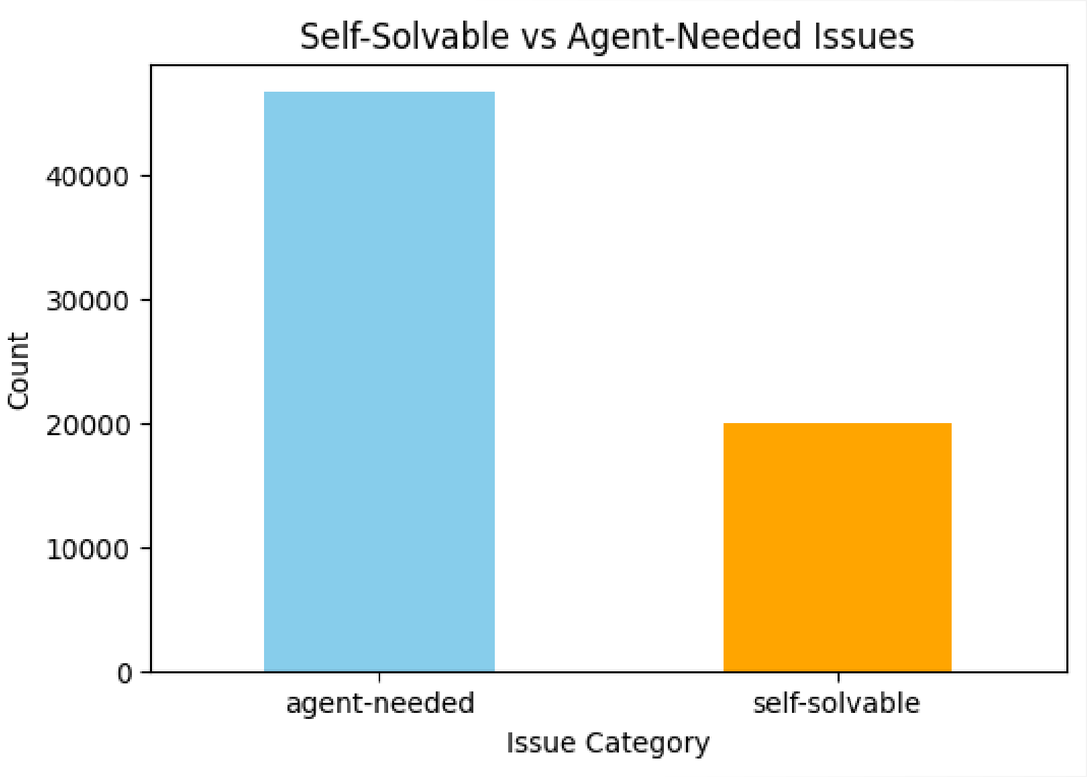
Inference from the data and data insights:

From the data, we inferred a weakness about company’s scheme of giving priority codes to customer . From the graph, we can observe that customers with higher priority codes don’t get a significant advantage over customers with no priority codes. The avg AST for both type of customers is very similar which does not provide any significant incentive for customers to buy memberships. Also this affects the overall AST which can be reduced if query time of high elite code customers is reduced

KEY RECOMMENDATIONS: we need to prioritize customers with higher elite code and queue them from the backend more earlier than other customers

Tagging Call Reasons: Self-Solvable vs. Agent-Needed

1. **Preprocessing Call Transcripts**:
   - Utilized Regex to clean and preprocess the call transcripts, ensuring structured text for further analysis.
2. **TF-IDF Matrix Analysis**:
   - Applied Term Frequency-Inverse Document Frequency (TF-IDF) to generate a matrix that highlights the most relevant terms in the dataset.
3. **Identifying Common Queries**:
   - Analyzed the data to find frequent reasons that could be handled by the company's IVR system, such as:
     - Ticket cancellation
     - Rescheduling
4. **Enhancing Keyword Detection with AI**:
   - Leveraged OpenAI to suggest additional keywords related to common queries (e.g., cancellation, schedule changes).
   - These keywords were then mapped to the dataset to identify self-solvable vs. agent-needed queries.
5. **Binary Classification Using Logistic Regression**:
   - Built a logistic regression model to classify queries as either self-solvable or agent-needed, treating it as a binary classification problem.



	precision	recall	f1-score	support
agent-needed	0.89	0.97	0.93	9372
self-solvable	0.91	0.73	0.81	3956
accuracy			0.90	13328
macro avg	0.90	0.85	0.87	13328
weighted avg	0.90	0.90	0.89	13328

BiLSTM for Sequence Modeling and Text Understanding

- BiLSTM models are an advanced form of RNN (Recurrent Neural Network) that process information in both forward and backward directions, which is critical for context-aware text classification.
- In this scenario, the call transcripts (which are sequences of words) were used as input to the BiLSTM model. The forward LSTM processed the text from the beginning to the end of the sequence, while the backward LSTM processed the text from the end to the beginning. This bidirectional approach allowed the model to capture contextual relationships between words more effectively than a unidirectional model could.
- As a result, the BiLSTM model learned to understand the overall meaning of the transcript, helping it match the most appropriate call reason based on both past and future word context within the conversation.
- **Data Preprocessing for the BiLSTM Model**
- We started by preprocessing the call transcripts to prepare them for model training:
- Text cleaning steps included removing unnecessary symbols, punctuation, and stop words.
- Tokenization was performed to convert the text into numerical sequences, where each word was mapped to a unique integer.
- These sequences were padded to ensure uniform input lengths for the BiLSTM model.
- Additionally, the call reasons were transformed into categorical labels (using one-hot encoding) so that the model could be trained to predict these labels from the text.
- **Model Training and Call Reason Tagging**
- Once the data was preprocessed, we trained the BiLSTM model using the prepared transcripts as input and their corresponding call reasons as labels.
- The BiLSTM model was trained to minimize the loss between predicted call reasons and the actual call reasons, effectively learning the relationship between the content of the call and the reason it was initiated.
- Through multiple training epochs and using categorical cross-entropy loss as the evaluation metric, the model gradually improved in its ability to predict the correct call reason based on the transcript.
- We used early stopping to prevent overfitting, ensuring that the model performed well not only on the training data but also on unseen test data.
- . Automatic Tagging for New Call Transcripts
- After training, the BiLSTM model was able to automatically tag new call transcripts with the most likely call reason, effectively replacing the need for manual tagging.
- The model could now take an unseen call transcript as input, and based on the context and sequence of words, it could predict the most appropriate call reason.

Conclusion

THE CONCLUSION FROM THE DATA ANALYSIS ARE AS FOLLOWS:

- In this project, we effectively leveraged machine learning, deep learning, and data analysis to improve operational efficiency and enhance customer satisfaction in a call center environment.
- **Use of Machine Learning for Identifying Self-Solvable Call Reasons:**
- By applying Natural Language Processing (NLP) techniques, such as TF-IDF and keyword extraction, we identified common call reasons that could be resolved without human intervention.
- This enabled us to categorize self-solvable queries (e.g., ticket cancellations and rescheduling) that could be handled by an IVR system, reducing the load on agents and optimizing resource allocation.
- **Leveraging BiLSTM for Automatic Call Reason Tagging:**
- We utilized a BiLSTM model to automatically classify and tag call transcripts to their corresponding reasons, eliminating the need for manual tagging.
- This approach not only improved accuracy in classifying call reasons (achieving a 97% match) but also significantly streamlined the tagging process, making it more efficient and scalable.
- **Data Analysis to Reduce AHT, AST, and Enhance Customer Satisfaction:**
- Through data-driven insights, we identified factors such as agent sentiment and call distribution that were influencing Average Handling Time (AHT) and Average Speed of Answer (AST).
- Sentiment analysis revealed that agents experiencing anger or frustration were linked to higher AHT, leading us to recommend periodic performance reviews based on sentiment scores and average call duration to ensure smoother operations.
- We also addressed the unequal distribution of calls by recommending a dynamic customer assignment system based on agent availability, helping to reduce AST and ensure a balanced workload.

KEY RECOMMENDATIONS

- **Monitor Agent Sentiment and Call Duration**: Implement a monthly performance review based on sentiment scores and average call duration to improve agent efficiency and reduce AHT.
- **Optimize Call Distribution:** Introduce a smart call routing system that dynamically assigns customers to agents based on real-time availability, reducing the inequality in workload and improving AST.
- **Automate Self-Solvable Queries**: Implement an IVR system for common, self-solvable queries like ticket cancellations and rescheduling, reducing the overall burden on agents and enhancing customer satisfaction.
- **Continuous Agent Training:** Provide emotion management training for agents exhibiting negative sentiment, helping them manage calls more effectively, further lowering AHT.

SKYHACK 2024

THANK YOU

FOR YOUR ATTENTION

PRESENTED BY - AAKHYAT BAGGA & KAMAL KUMAR