SKYHACK 2.0

Beyond the Beep: Enhancing Customer Experience Through Smarter Call Center Operations

Submitted By:
Belo Abhigyan
Rajat Sharma
Department of Computer Science, University of Delhi

Problem Statement

As United Airlines continues its journey to become the best airline in the history of aviation, it is crucial to provide world-class customer service, for which one of the key areas of focus is our call center operations. Call centers play a critical role in ensuring customer issues are resolved quickly and efficiently, but we face challenges in improving metrics such as Average Handle Time (AHT) and Average Speed to Answer (AST).

Your task is to optimize these key call center metrics, helping reduce resolution times and providing faster, more efficient service to our customers. You are required to analyze our existing call center data to identify inefficiencies, determine the drivers of long AHT and AST, and suggest strategies to enhance customer satisfaction, reduce escalations, and improve overall operational efficiency.

Data Description

Link for Datasets

Calls
Contains details of customer calls, including call ID, customer ID, agent ID, call

Customers
Contains customer details, including customer ID, name, and elite membership level.

Reason
Contains call IDs with the associated primary reason for the call.

Sentiment Statistics
Contains call ID, agent ID, and agent/customer tone, average sentiment, and percentage of silence during the call.

test.csv
Contains call IDs to be used for testing purposes.

test_output.csv
Contains predicted or processed results for call IDs from **test.csv**

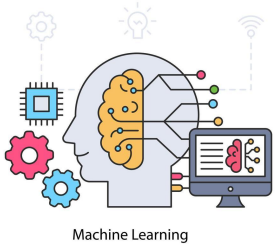
Deliverables:

(1) Long average handle time (AHT) affects both efficiency and customer satisfaction. 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. Additionally, could you quantify the percentage difference between the average handling time for the most frequent and least frequent call reasons?

(2) We often observe self-solvable issues unnecessarily escalating to agents, increasing their workload. 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. Propose specific improvements to the IVR options to effectively reduce agent intervention in these cases, along with solid reasoning to support your recommendations.

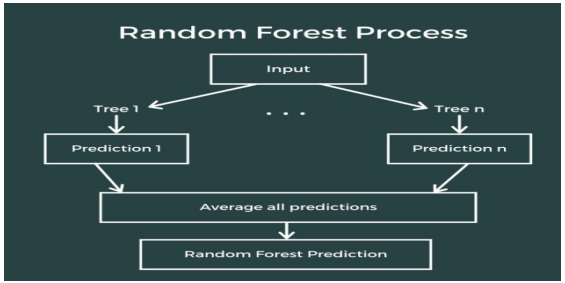
(3) Understanding the primary reasons for incoming calls is vital for enhancing operational efficiency and improving customer service. Accurately categorizing call reasons enables the call center to streamline processes, reduce manual tagging efforts, and ensure that customers are directed to the appropriate resources. In this context, analyze the dataset to uncover patterns that can assist in understanding and identifying these primary call reasons. Please outline your approach, detailing the data analysis techniques and feature identification methods you plan to use.

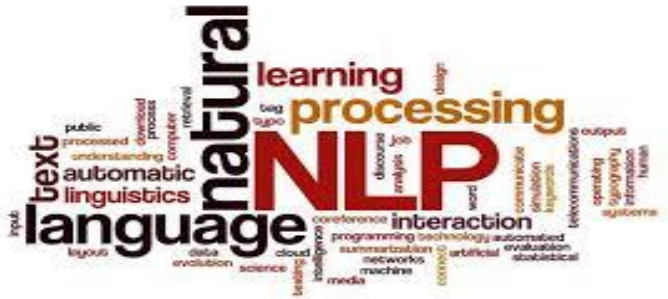
🔑 Key Concepts Utilized



Machine Learning Techniques 🧠

- **K-Means Clustering** 📊:
  - Grouped call transcripts to uncover recurring themes and issues in customer interactions.
  - Reason: Helped identify potential areas for IVR improvement by clustering based on common issues.
- **Random Forest Regression** 🌲:
  - Predicted **Average Handle Time (AHT)** based on multiple features such as customer sentiment, silence percentage, and call type.
  - 🎯 Reason: Highlighted which factors most significantly impacted AHT using feature importance analysis



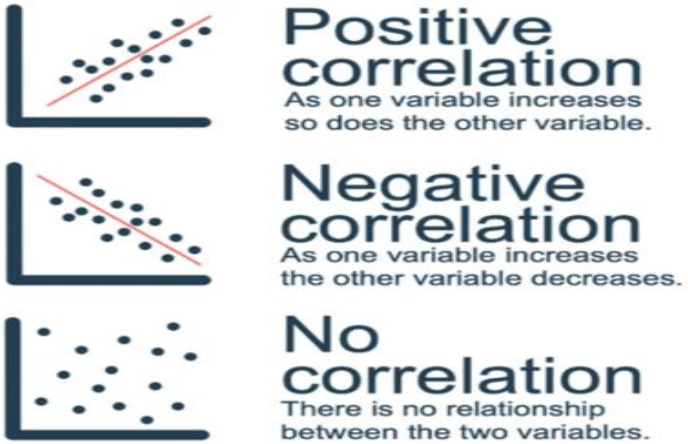


🗣️ Natural Language Processing (NLP)

- **NLP Overview 🤖**:
  - Used the **Natural Language Toolkit (NLTK)** in Python to process and analyze text.
  - Reason: NLTK supports tasks like **tokenization**, **stemming**, **tagging**, and **semantic reasoning** for rich language understanding.
- **Text Preprocessing 🧼**:
  - Cleaned and normalized call transcripts (lowercasing, punctuation removal, stopword filtering).
  - Reason: These steps prepared the text for modeling by ensuring uniformity and removing noise.
- **TF-IDF Vectorization 🧮**:
  - Used **Term Frequency-Inverse Document Frequency (TF-IDF)** to convert text into numerical features for clustering.
  - Reason: This helped in transforming call transcripts into a format suitable for machine learning models.
- **Sentiment Analysis 😊😡**:
  - Evaluated the emotional tone of customer conversations.
  - Reason: Assessed how positive or negative sentiment impacts **AHT** and overall customer experience.

📊 Correlation Analysis & Insights

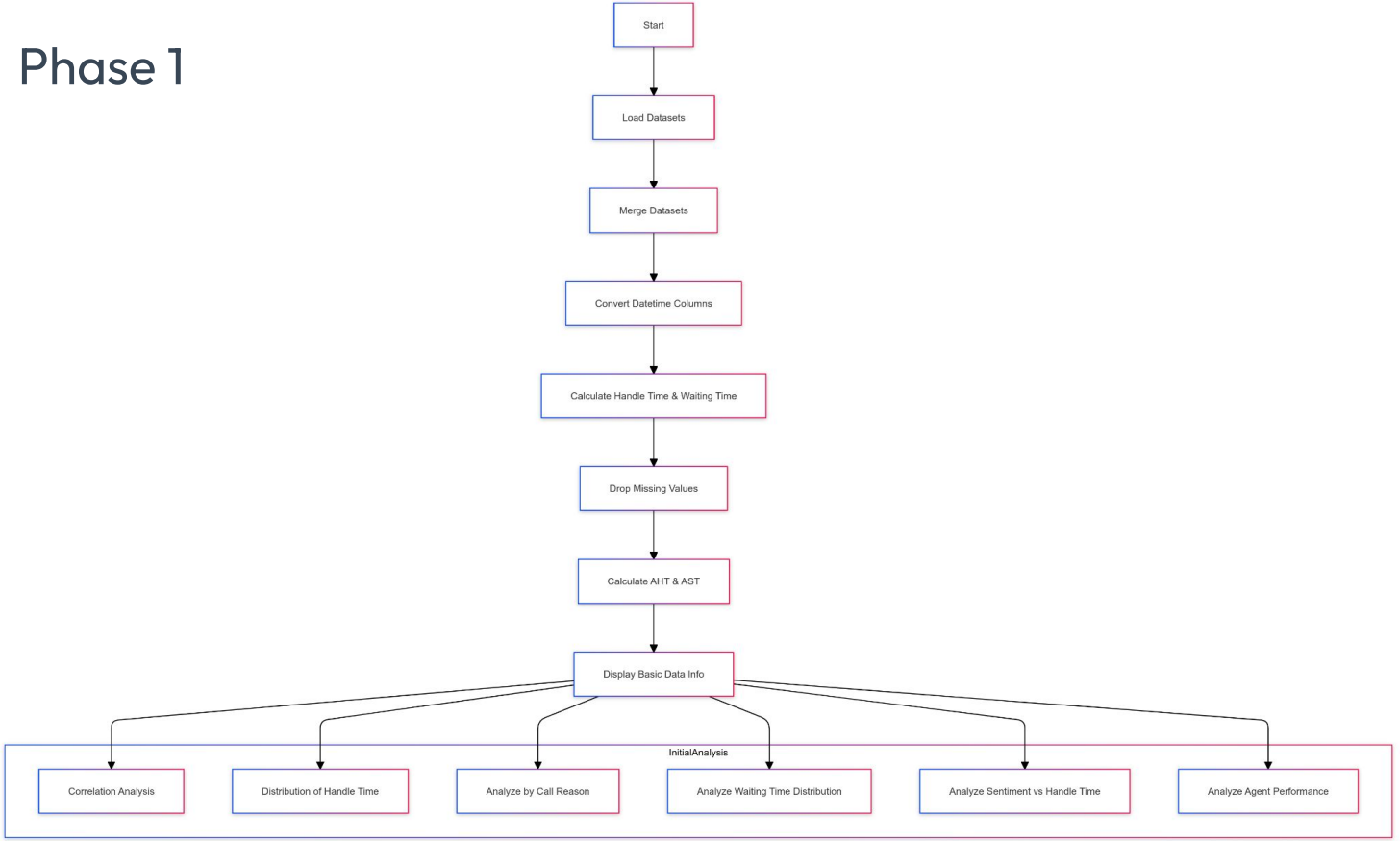
- **Heatmaps 🔥:**
  - Visualized relationships between factors like **AHT**, **waiting time**, **sentiment scores**, and **silence percentages**.
  - Reason: These visual tools made it easier to spot correlations that affect performance metrics.
- **Statistical Insights 🧠:**
  - Discovered significant correlations between sentiment and AHT, revealing that **negative sentiment** led to longer call durations.
  - Also found that extended wait times and silence during calls were major contributors to increased handle times.



Methodology and techniques applied:

Methodology

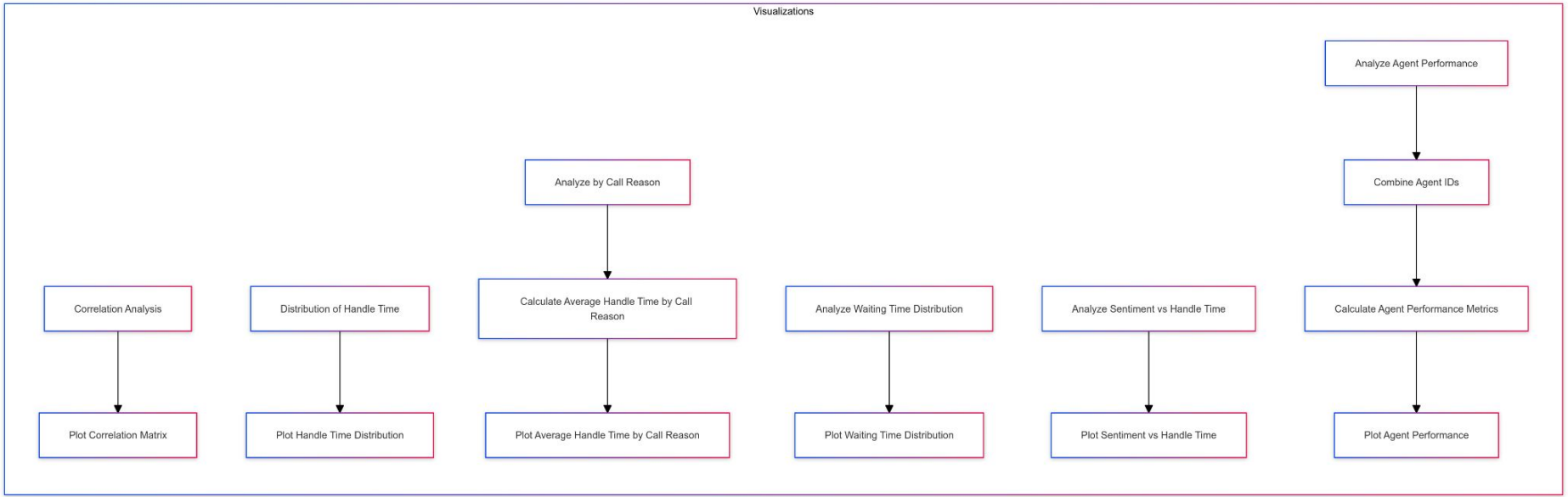
Phase 1



Phase 2


AdvancedAnalysis
Display Basic Data Info
Quantifying Handling Time Differences
Drop Rows with Missing Call Reasons
Find Most and Least Frequent Call Reasons
Calculate Percentage Difference in AHT
IVRSystemImprovement
Improving IVR System
Text Preprocessing
TF-IDF Vectorizer
K-Means Clustering
Analyze Clusters for Self-Service Options
Estimate Potential Time Savings
Propose IVR Improvements

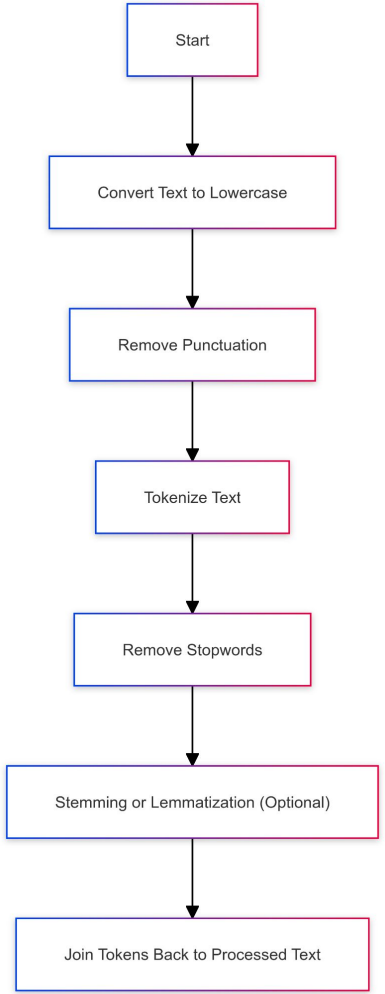
Phase 3


Visualizations
Analyze Agent Performance
Combine Agent IDs
Calculate Agent Performance Metrics
Plot Agent Performance
Analyze by Call Reason
Calculate Average Handle Time by Call Reason
Plot Average Handle Time by Call Reason
Correlation Analysis
Plot Correlation Matrix
Distribution of Handle Time
Plot Handle Time Distribution
Analyze Waiting Time Distribution
Plot Waiting Time Distribution
Analyze Sentiment vs Handle Time
Plot Sentiment vs Handle Time

Techniques applied:

1. NLP- natural language processing
   Natural language processing (NLP) is a rapidly growing field that integrates computer science, artificial intelligence, and linguistics. It enables machines to interpret and generate meaningful human language, making it a crucial tool for extracting valuable insights and automating various processes in the increasing volume of text data generated daily.

   Architecture →

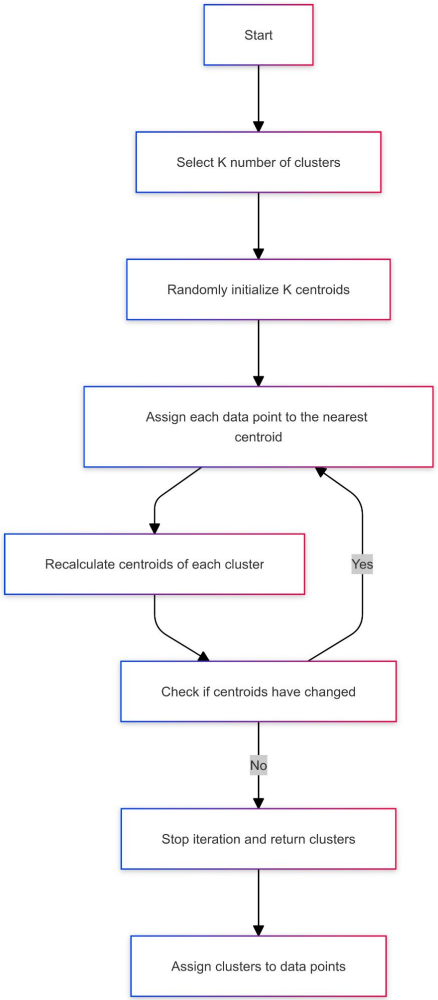


Techniques applied: cont.

2. K-means clustering

K-means clustering is an unsupervised machine learning algorithm that divides data into k clusters based on similarity. It assigns data points to the nearest centroid using Euclidean distance, recalculates them using the mean, and repeats until stabilization. It's useful for market segmentation and image reduction but requires prior knowledge.

Architecture →

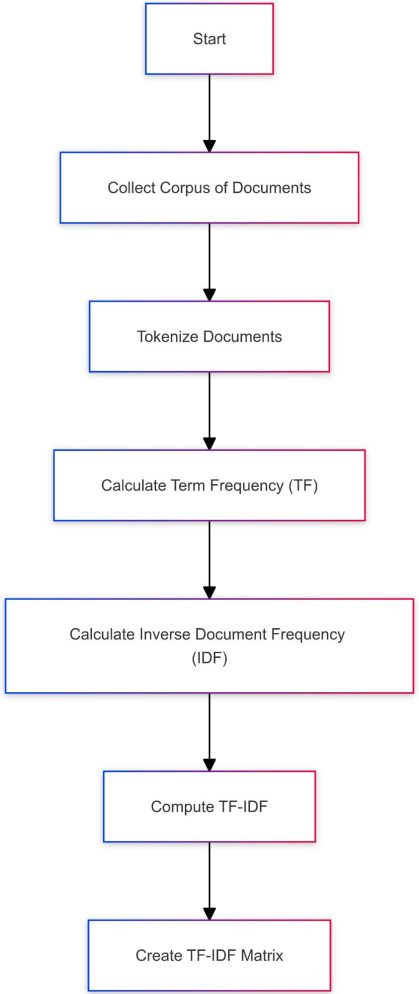


Techniques applied: cont.

3. TF-IDF

TF-IDF is a statistical metric that evaluates the relevance of a term in a text compared to its corpus. It consists of term frequency (TF) and inverse document frequency (IDF), with higher scores indicating more relevance for phrases common in one document.

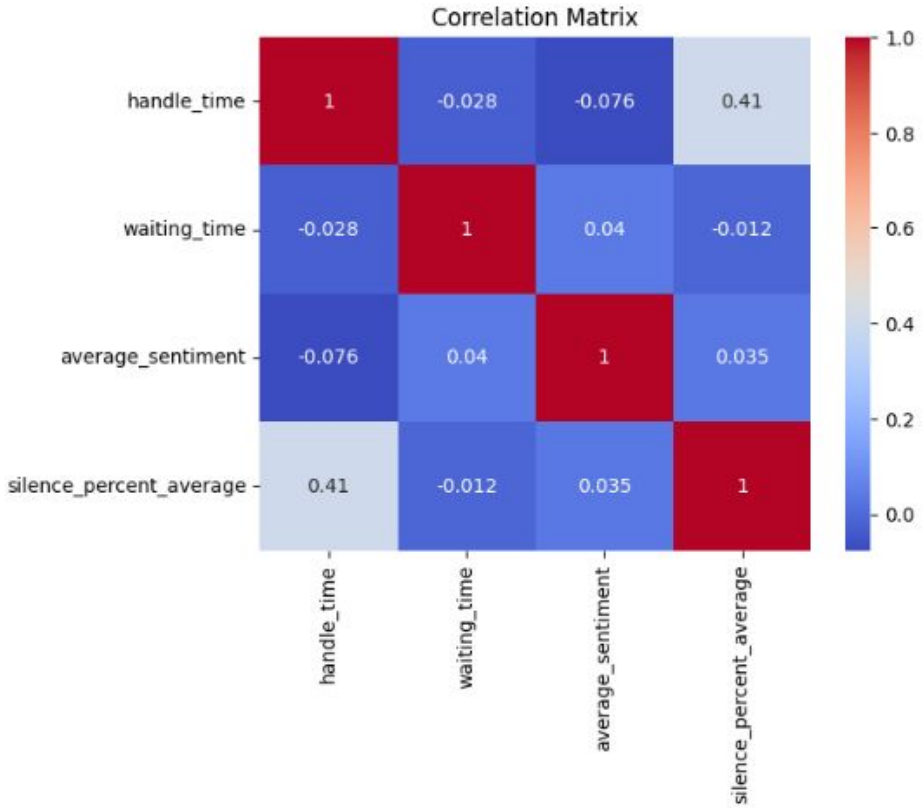
Architecture →



Identifying Key Drivers of AHT and AST

Objective: Identify factors contributing to high AHT and AST, such as call types, agent efficiency, customer sentiment, and peak call times

```
[ ]  # Correlation analysis
     correlation = df[['handle_time', 'waiting_time', 'average_sentiment',
     'silence_percent_average']].corr()
     sns.heatmap(correlation, annot=True, cmap='coolwarm')
     plt.title('Correlation Matrix')
     plt.show()
```



🌐 Insights from the Correlation Matrix: Unveiling Hidden Connections

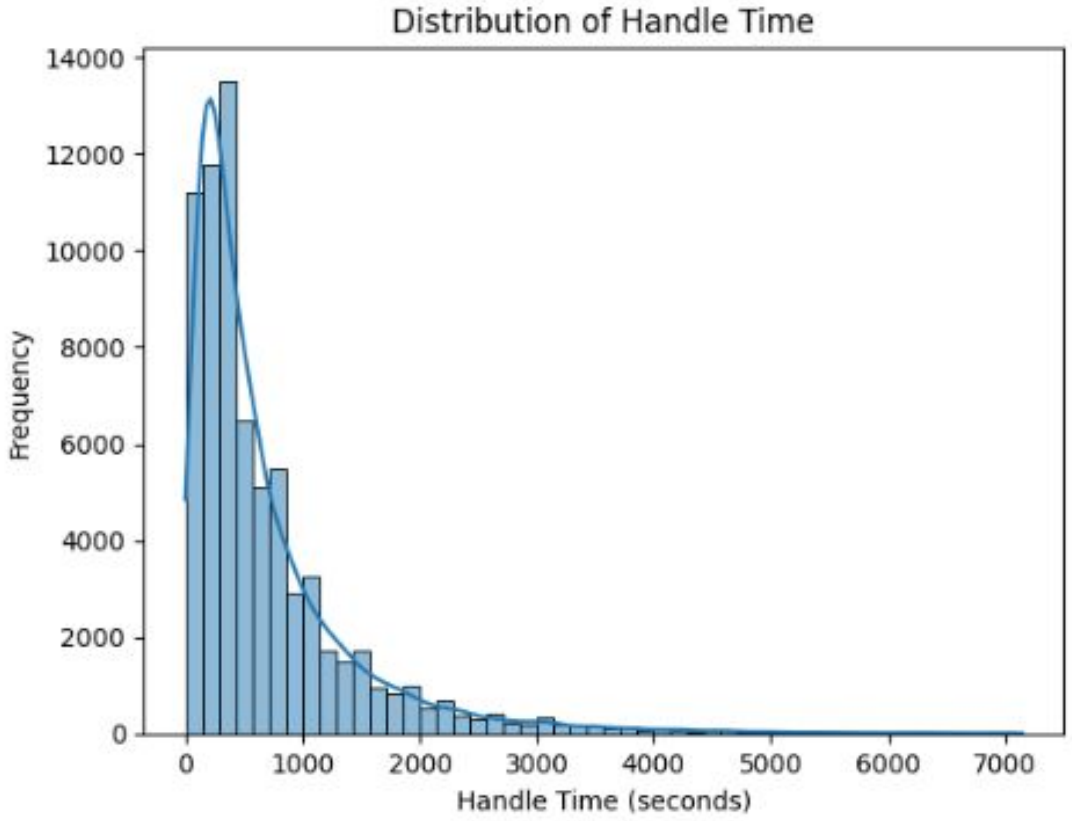
🔍 Insight Snapshot:

- **Handle Time & Waiting Time: The Frustration Feedback Loop**
  - Positive Correlation: Longer wait times usually leads to longer call durations, indicating customer frustration and complex interactions. ⏳😠📞
- **Sentiment & Handle Time: The Emotional Equation**
  - Positive Link: Negative sentiment (like complaints etc) correlates with longer handle times, requiring more time for resolution. 😡⏱️🔄
- **Silence Duration & Handle Time: The Communication Gap**
  - Positive Correlation: Extended silences signify inefficient communication, prolonging handle times. 🤔🔇⏳
- **Call Reason & Handle Time: The Complexity Connection**
  - High Correlation: Specific call reasons (e.g., "checkout" or "complaints") drive longer call durations, highlighting complex issues. 🎯🛒📢

💡 Key Insights:

- **Frustration Fuels Duration**: Minimize wait times to reduce customer frustration and call lengths. 🚫⏳😌
- **Emotions Extend Calls:** Addressing negative experiences efficiently to shorten handle times. 🌪️🕑🔧
- **Silence Isn't Golden**: Better communication to reduce silences and handle times. 🗣️🔇🔄
- **Prioritize Problem Areas**: Focus on high-correlation call reasons for operational improvements. 🎯🛠️📈

```
[ ]  # Distribution of Handle Time
     sns.histplot(df['handle_time'], bins=50, kde=True)
     plt.title('Distribution of Handle Time')
     plt.xlabel('Handle Time (seconds)')
     plt.ylabel('Frequency')
     plt.show()
```



📊 Handle Time Distribution: The 0-1000 Seconds Sweet Spot

🔍 Insight Snapshot:

- **Peak Performance Range:** The histogram highlights a busy area of calls clustering between 0 and 1000 seconds, boasting a robust frequency of 10,000 to 14,000 calls. 📈🔝
  - **What It Means**: Majority of the calls are finished in the above time range , showing that they are efficiently handled in most of the scenarios. 🎯⏱️

- **Room for Refinement**: Despite the high frequency in this range, calls stretching beyond 1000 seconds wave red flags of potential inefficiencies. 🚩⏳

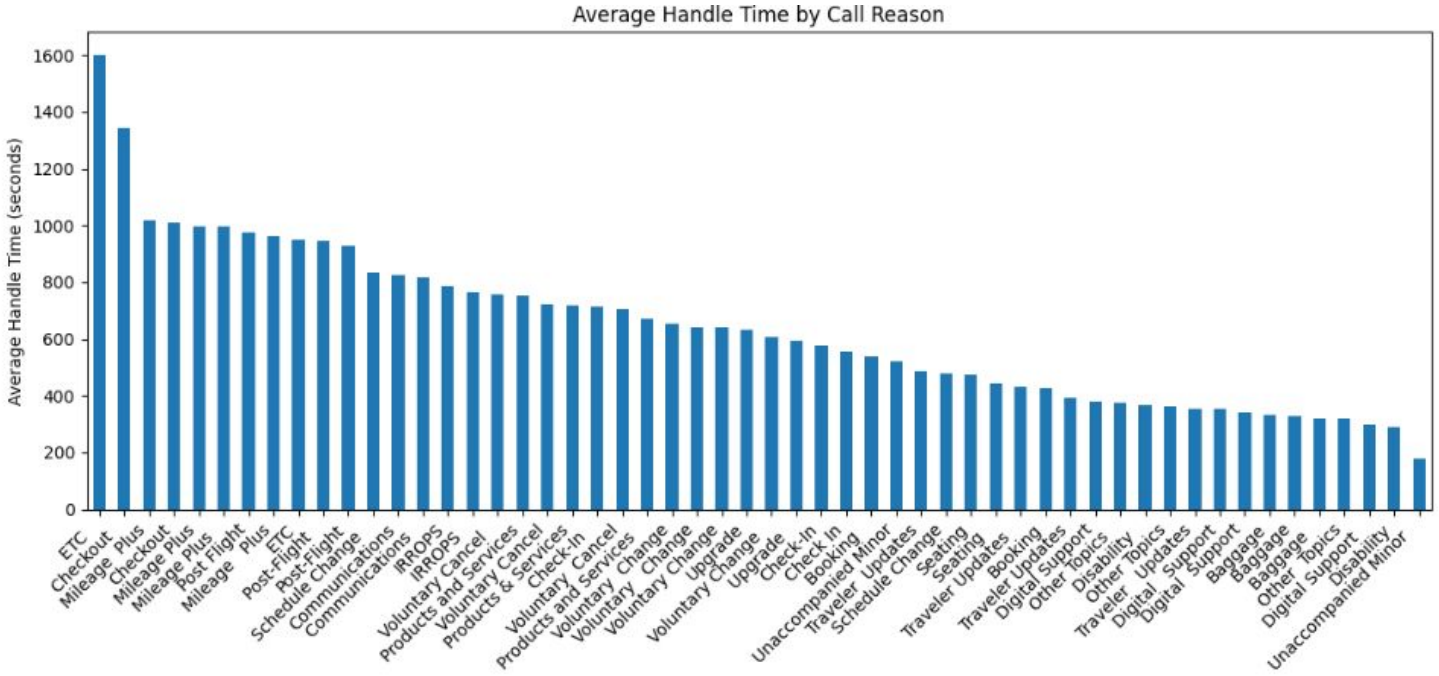
💡 Key Insights Gained 💪:

- **Standard Success, Although...:** The peak frequency in the 0-1000 seconds range hints at a generally low standard handle time, but the outliers hint at untapped efficiency reserves. 🌟🔄
- **Outlier Investigation**: Calls exceeding 1000 seconds are the outliers demanding deeper analysis to uncover the root causes of their extended durations. 🔍🕵️

🌟 Strategic Recommendation:

- **Fine-Tune the Norm**: Continue optimizing processes within the 0-1000 seconds range to maintain efficiency. 🛠️📈
- **Zoom In on Outliers:** Conduct a thorough analysis of calls surpassing 1000 seconds to identify and address underlying factors, paving the way for holistic improvement. 🧐🔄

```python
# Analyze by call reason
if 'primary_call_reason' in df.columns:
    reason_aht = df.groupby('primary_call_reason')['handle_time'].mean().sort_values(ascending=False)
    plt.figure(figsize=(12, 6))
    reason_aht.plot(kind='bar')
    plt.title('Average Handle Time by Call Reason')
    plt.xlabel('Call Reason')
    plt.ylabel('Average Handle Time (seconds)')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
```



🕑 Average Handle Time (AHT) vs. Call Reason: Unraveling the Impact

📊 Insight Snapshot:

- **Time-Consuming Titans**:
  - **ETC:** Clocking in at a hefty ~1600 seconds, these calls are the marathon runners of the call center world. 🏃‍♂️⏳
  - **Checkout:** Not far behind, averaging ~1300 seconds, these calls also demand substantial agent time. 🛒⏰
- **Why the Lag**? These categories most probably deal with complex or sensitive issues, stretching agent efforts and highlighting areas where we can improve processes to reduce Average Handling Time (AHT). 🧩🔍
- **Speedy Sprinter:**
  - Unaccompanied Minor: Zipping through at under 200 seconds, these calls are the literally the Usain Bolts of the call center, likely benefiting from well-defined, efficient processes. 🏃‍♀️⚡

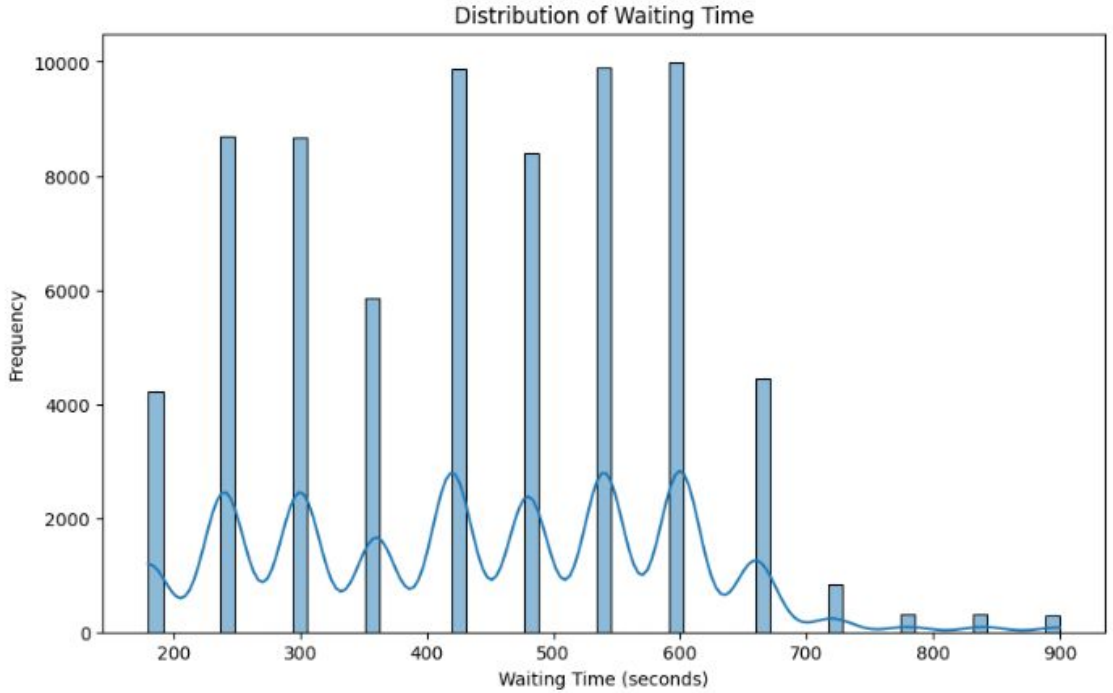
💡 **Actionable Insight Gained:**

- **Target the Time Guzzlers**: Zero in on high-AHT categories like ETC and Checkout. By fortifying agent training and refining IVR routing, United Airlines can unlock substantial efficiency gains. 🎯🚀

🌟 **Strategic Recommendation**:

- **Empower Agents**: Equip agents with specialized training to navigate the complexities of ETC and Checkout calls more swiftly. 📚💼
- **Optimize IVR:** Fine-tune IVR systems to better route these calls, ensuring they reach the most adept agents faster. 🎙️🔄

```python
# Analyze waiting time distribution
plt.figure(figsize=(10, 6))
sns.histplot(df['waiting_time'], kde=True)
plt.title('Distribution of Waiting Time')
plt.xlabel('Waiting Time (seconds)')
plt.ylabel('Frequency')
plt.show()
```



4. Histplot of Frequency vs Waiting Time:

🎯 Waiting Time Insights: Balancing Speed and Satisfaction

Efficiency in Action: The majority of calls are promptly answered, showcasing our call center's ability to manage high-volume, time-sensitive inquiries effectively.

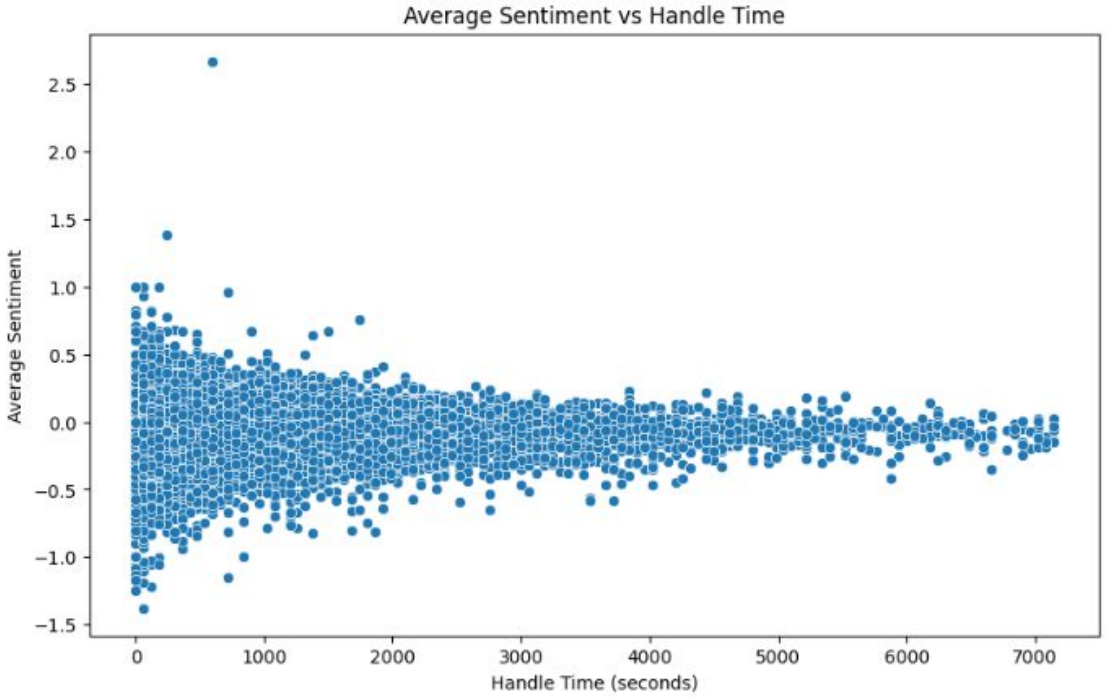
The Long Wait Dilemma: However, a significant portion of calls experience extended waiting times, which can escalate customer frustration and lead to longer call resolution times.

💡 **Strategic Focus**:

- **Enhance Agent Availability**: Increase the number of available agents during peak times to reduce wait times.
- **Optimize Call Routing:** Implement smarter call routing algorithms to ensure quicker connections.

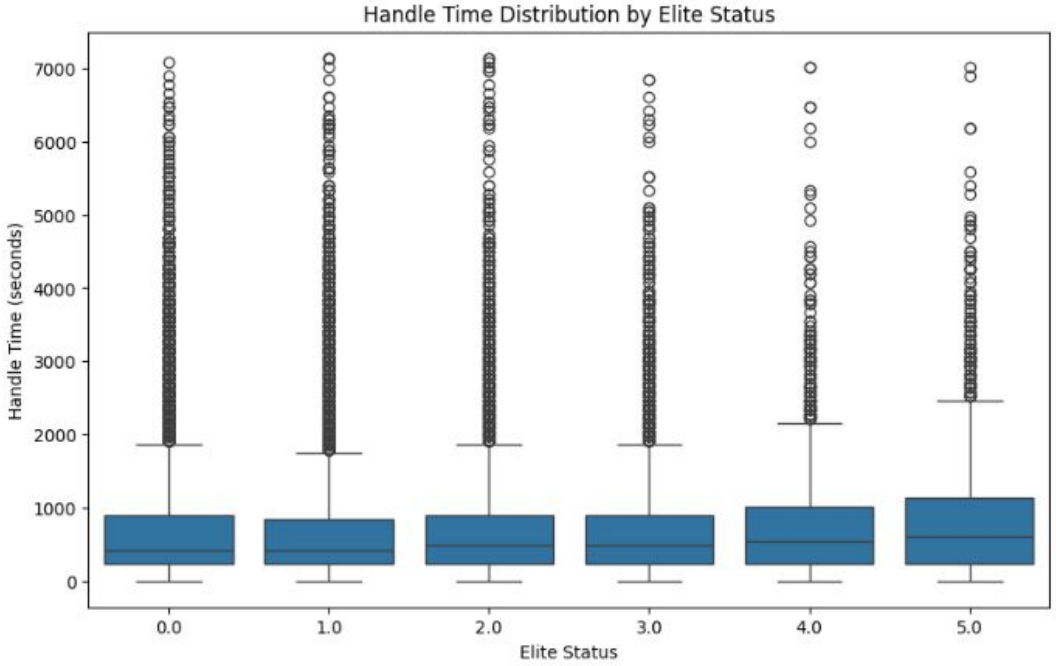
By addressing these areas, we can significantly improve customer satisfaction and streamline call handling, ultimately lowering both Average Speed to Answer (AST) and Average Handle Time (AHT).

```
[ ]  # Analyze sentiment vs handle time
     if 'average_sentiment' in df.columns:
         plt.figure(figsize=(10, 6))
         sns.scatterplot(x='handle_time', y='average_sentiment', data=df)
         plt.title('Average Sentiment vs Handle Time')
         plt.xlabel('Handle Time (seconds)')
         plt.ylabel('Average Sentiment')
         plt.show()
```

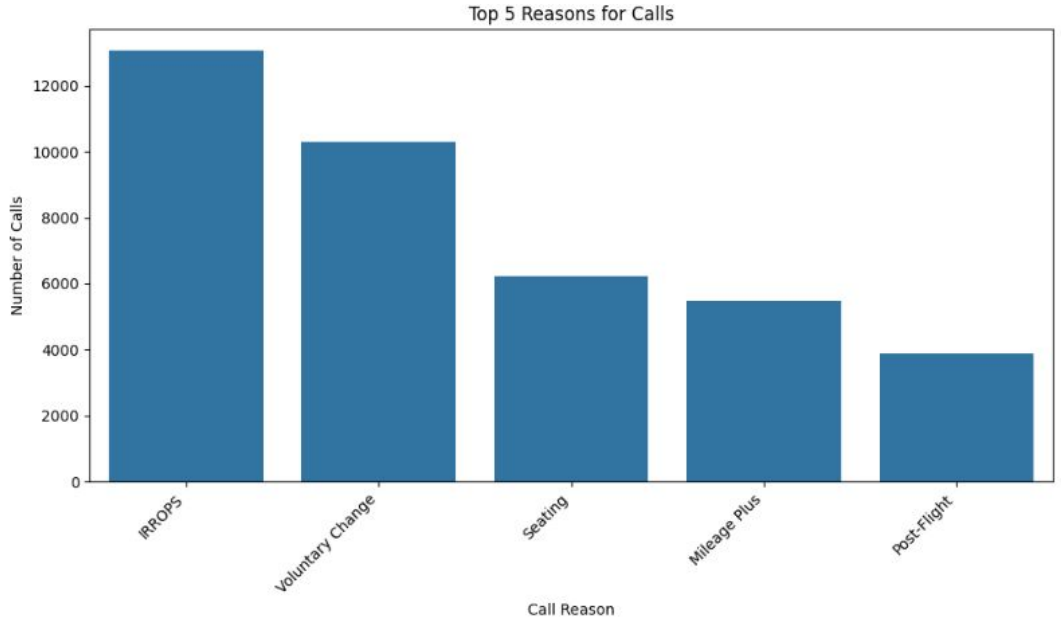


```python
# Since we have two agent ID features so we are
# Creating a single agent ID by combining agent_id_x and agent_id_y
df['agent_id'] = df['agent_id_x'].combine_first(df['agent_id_y'])

# Now you can analyze performance based on the combined agent_id
if 'agent_id' in df.columns:
    agent_performance = df.groupby('agent_id').agg({
        'handle_time': 'mean',
        'average_sentiment': 'mean',
        'silence_percent_average': 'mean'
    }).sort_values('handle_time', ascending=False)

    # Check if the DataFrame is not empty
    print(agent_performance.head())  # Print first few rows to ensure data exists

    # Plot the data
    if not agent_performance.empty:
        plt.figure(figsize=(12, 6))
        sns.scatterplot(x='handle_time', y='average_sentiment', size='silence_percent_average',
                        data=agent_performance, sizes=(20, 200))
        plt.title('Agent Performance: Handle Time vs Sentiment')
        plt.xlabel('Average Handle Time (seconds)')
        plt.ylabel('Average Sentiment')
        plt.show()
    else:
        print("No data available for plotting.")
else:
    print("Column 'agent_id' does not exist in the DataFrame.")
```

```python
# Analyze impact of elite status on handle time
if 'elite_level_code' in df.columns:
    plt.figure(figsize=(10, 6))
    sns.boxplot(x='elite_level_code', y='handle_time', data=df)
    plt.title('Handle Time Distribution by Elite Status')
    plt.xlabel('Elite Status')
    plt.ylabel('Handle Time (seconds)')
    plt.show()
```



```
[ ]
    # Additional analysis: Top 5 reasons for calls
    if 'primary_call_reason' in df.columns:
        top_5_reasons = df['primary_call_reason'].value_counts().nlargest(5)
        plt.figure(figsize=(10, 6))
        sns.barplot(x=top_5_reasons.index, y=top_5_reasons.values)
        plt.title('Top 5 Reasons for Calls')
        plt.xlabel('Call Reason')
        plt.ylabel('Number of Calls')
        plt.xticks(rotation=45, ha='right')
        plt.tight_layout()
        plt.show()
```



5. Call Reason vs Number of Calls (Top 5 Reasons):
📞 Call Volume Breakdown: Top 5 Hotspots

🔥 Top Call Drivers:
1. Complaints 📢
2. Account Woes 🔐
3. Refund/Exchange Quests ↩️
4. Booking SOS 🛫
5. ETC (Miscellaneous) 🎯

💡 **Insight Flash**:
- **Concentration Central:** These 5 categories hog the call limelight, signaling a tight cluster of customer pain points.
- **Red Flags:** Complaints and Account Issues topping the chart hint at critical CX gaps, potentially fueling dissatisfaction and stretching call durations.

🚀 Action Boost:
- **Streamline to Soar**: Fine-tune processes for these hot-button issues, especially Complaints, to turbocharge agent efficiency, slash AHT, and elevate overall customer joy! 🌟

Conclusion:

📊 **Insights Unveiled**:

- **Stress Spikes**: High correlations between wait time & handle time, and sentiment & silence duration, spotlight inefficiencies in high-stress scenarios. 🚨
- **Handle Time Hurdles**: Majority of calls clock in at 0-1000 seconds, but outliers demand a closer look for optimization. ⏱️
- **Process Pain Points**: Calls tagged as ETC and Checkout drag out handle times, signaling ripe areas for process refinement. 🛠️
- **Wait Watch**: Short waits are the norm, but lengthy delays stretch calls and tarnish customer joy. ⏳
- **Top Troubles**: Tackling the most common call reasons, particularly complaints and account issues, could unlock major performance gains.🌟

🚀 **Strategic Action Blueprint:**

- **Empower Agents**: Deploy targeted training modules to arm agents with the skills to navigate high-stress calls seamlessly. 🛡️📚
- **Process Revolution**: Overhaul and streamline processes for ETC and Checkout calls to eliminate bottlenecks and expedite resolutions. 🔄🛠️
- **Focus on Frequents**: Prioritize and optimize responses to top call reasons, particularly complaints and account issues, to drive substantial performance leaps. 🎯📈

Quantifying Handling Time Differences

Objective : Assess the percentage difference in AHT between the most frequent and least frequent call reasons to identify key drivers affecting handle time, especially during peak call volumes.

```
[ ]  # Step 1: Drop rows where primary_call_reason is missing
     df_cleaned = df.dropna(subset=['primary_call_reason'])

     # Step 2: Find the most and least frequent call reasons
     call_reason_counts = df_cleaned['primary_call_reason'].value_counts()
     most_frequent_reason = call_reason_counts.idxmax()  # Most frequent
     least_frequent_reason = call_reason_counts.idxmin()  # Least frequent

     # Step 3: Calculate average handling times for most and least frequent reasons
     most_frequent_avg_handle_time = df_cleaned[df_cleaned['primary_call_reason'] == most_frequent_reason]['handle_time'].mean()
     least_frequent_avg_handle_time = df_cleaned[df_cleaned['primary_call_reason'] == least_frequent_reason]['handle_time'].mean()

     # Step 4: Calculate the percentage difference
     percentage_difference = ((most_frequent_avg_handle_time - least_frequent_avg_handle_time) / least_frequent_avg_handle_time) * 100

     # Output the results
     print(f"Most frequent call reason: {most_frequent_reason}")
     print(f"Least frequent call reason: {least_frequent_reason}")
     print(f"Average handling time for most frequent: {most_frequent_avg_handle_time}")
     print(f"Average handling time for least frequent: {least_frequent_avg_handle_time}")
     print(f"Percentage difference: {percentage_difference}%")
```

```
Most frequent call reason: IRROPS
Least frequent call reason: Unaccompanied Minor
Average handling time for most frequent: 785.4913073447193
Average handling time for least frequent: 180.0
Percentage difference: 336.38405963595517%
```

📞 Call Reason Dynamics: From Frequent to Rare

🔍 **Insight Snapshot**:
- **Most Frequent Call Reason: IRROPS (Irregular Operations)**
  - **Average Handle Time:** A hefty **785.49 seconds**🕑📈

- **Least Frequent Call Reason: Unaccompanied Minor**
  - **Average Handle Time:**A swift **180.0 seconds** 🕑🚀

- **Percentage Difference:** The handle time for IRROPS is a staggering **336.38%** higher than for Unaccompanied Minor calls. 📊🔝

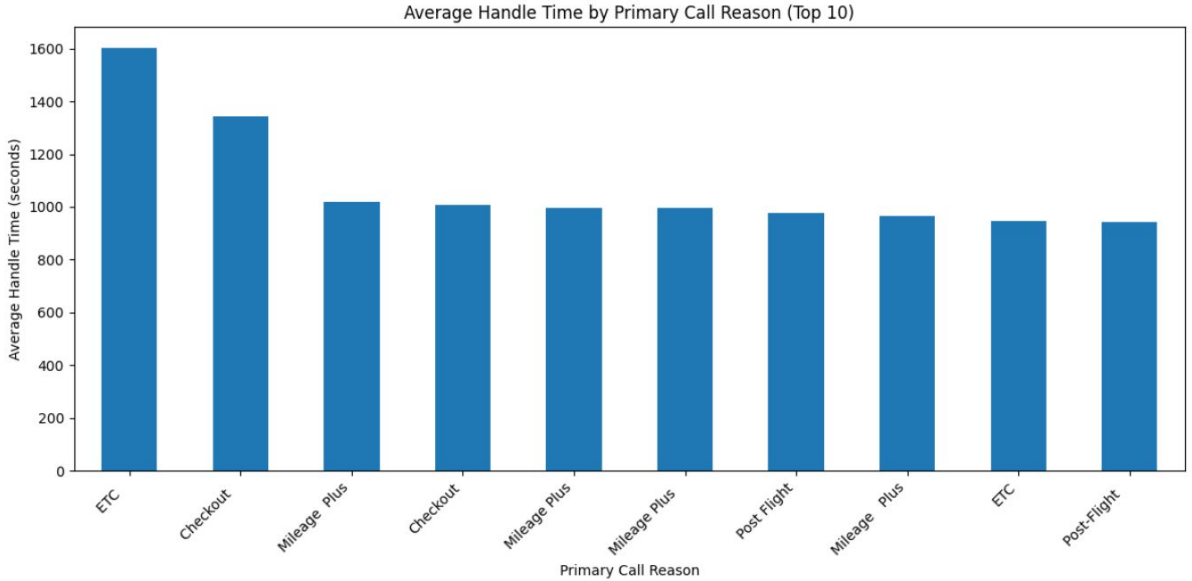
💡 **Key Insight:**
- **Time-Consuming Tug-of-War:** The vast percentage difference underscores that managing IRROPS calls is significantly more time-intensive compared to the relatively straightforward Unaccompanied Minor calls. 🌪️⏳ vs. 🚶⚡

🌟 **Strategic Recommendation**:
- **Streamline IRROPS Handling**:Given the substantial time drain, prioritize process optimizations and specialized training for IRROPS to enhance efficiency. 🛠️📚
- **Leverage Efficiency in Unaccompanied Minor Calls:**** Continue to capitalize on the efficient handling of Unaccompanied Minor calls, possibly using them as a benchmark for other call types. 🌟🔄

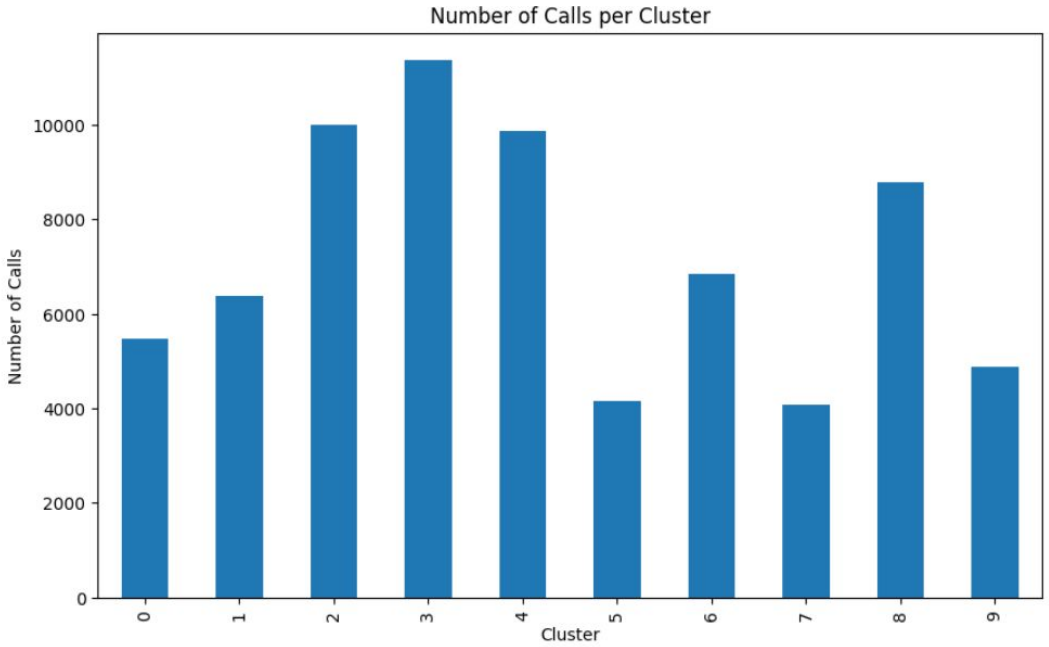
Improving IVR System to Reduce Agent Escalations

Objective: Enhance the Interactive Voice Response (IVR) system to handle common call reasons through self-service,thereby reducing the likelihood of agent escalations and improving overall call center efficiency.

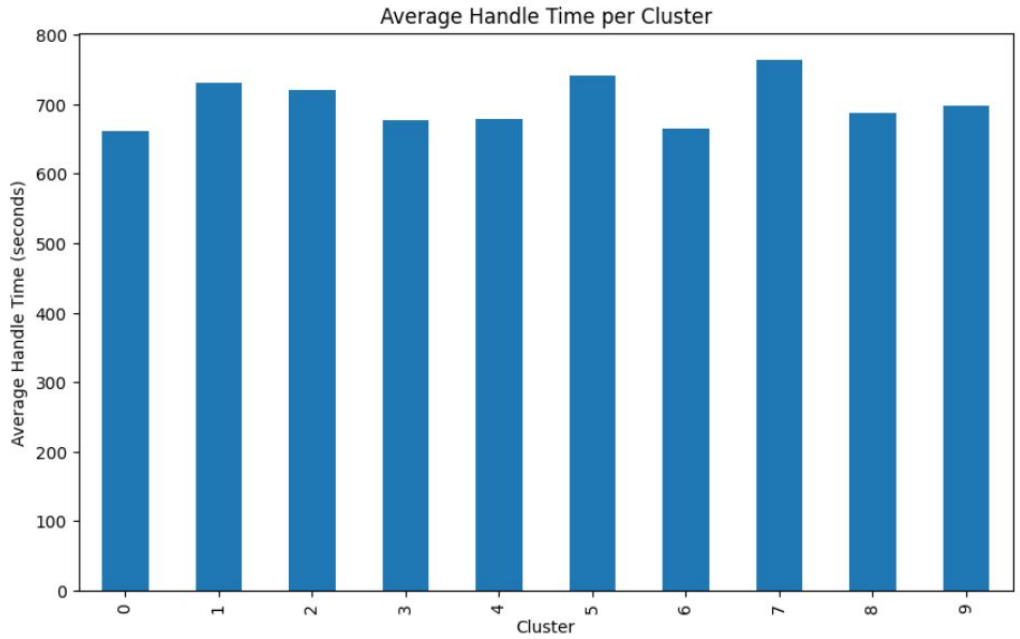
```python
# Analyze relationship between call reasons and handle time
reason_handle_time = df.groupby('primary_call_reason')['handle_time'].mean().sort_values(ascending=False)
plt.figure(figsize=(12, 6))
reason_handle_time.head(10).plot(kind='bar')
plt.title('Average Handle Time by Primary Call Reason (Top 10)')
plt.xlabel('Primary Call Reason')
plt.ylabel('Average Handle Time (seconds)')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



```
[ ]  # Visualize cluster sizes
     plt.figure(figsize=(10, 6))
     df['cluster'].value_counts().sort_index().plot(kind='bar')
     plt.title('Number of Calls per Cluster')
     plt.xlabel('Cluster')
     plt.ylabel('Number of Calls')
     plt.show()
```



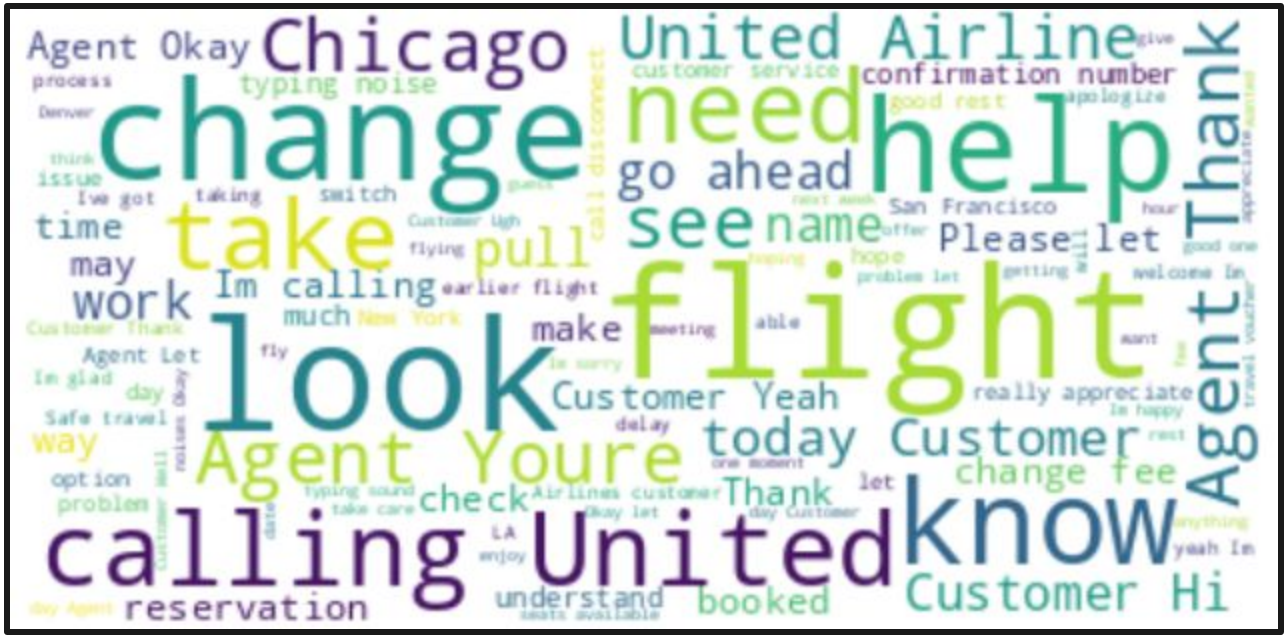
```
[ ]  # Visualize average handle time per cluster
     plt.figure(figsize=(10, 6))
     df.groupby('cluster')['handle_time'].mean().plot(kind='bar')
     plt.title('Average Handle Time per Cluster')
     plt.xlabel('Cluster')
     plt.ylabel('Average Handle Time (seconds)')
     plt.show()
```



Optimizing Call Center Metrics: A Data-Driven Approach

Objective: Enhance the Interactive Voice Response (IVR) system to handle common call reasons through self-service,thereby reducing the likelihood of agent escalations and improving overall call center efficiency.

Word Cloud from transcript of calls



🔍 Inference from Word Cloud

- **Primary Call Drivers:**
  🚀 Flight Changes dominate the discussions, followed by Flight Confirmations and Timing Adjustments.
  📊 This indicates that customers frequently seek assistance for these specific issues.

🌥️ Word Clouds for Granular Insights could be utilized for obtaining solutions for Deliverable 2 (as given): ☁️

- 🔍 **Analyzing Transcripts & Call Reasons:**
  Word Clouds can help visualize recurring keywords in call transcripts, highlighting granular reasons behind frequent customer inquiries.
- 🛠️ Identifying Self-Service Opportunities:
  By mapping these keywords to specific recurring problems, we can identify areas suitable for self-service options in the IVR system.
  💡 For example, frequent mentions of "change flight" or "cancel booking" could be automated in the IVR, reducing customer wait times and agent workload.

Through NLP and Clustering:

Specifying the technical approaches we have used to achieve our goal of enhancing IVR systems

```
def preprocess_text(text):
    text = str(text).lower()
    text = re.sub(r'[^\w\s]', '', text)
    stop_words = set(stopwords.words('english'))
    words = text.split()
    return ' '.join([word for word in words if word not in stop_words])

# Apply preprocessing to call transcripts
df['processed_transcript'] = df['call_transcript'].apply(preprocess_text)

# Use TF-IDF to extract features from transcripts
vectorizer = TfidfVectorizer(max_features=1000)
tfidf_matrix = vectorizer.fit_transform(df['processed_transcript'])

# Cluster transcripts to identify common themes
n_clusters = 10
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
df['cluster'] = kmeans.fit_predict(tfidf_matrix)

# Analyze clusters to identify common themes and potential self-solvable issues
cluster_themes = {}
potential_self_service_options = {}
```

```python
for cluster in range(n_clusters):
    cluster_transcripts = df[df['cluster'] == cluster]['processed_transcript']
    cluster_words = ' '.join(cluster_transcripts).split()
    word_freq = Counter(cluster_words)
    top_words = word_freq.most_common(10)
    cluster_themes[cluster] = [word for word, _ in top_words]

    print(f"Cluster {cluster} top words:")
    print(', '.join([word for word, _ in top_words]))

    # Analyze call reasons within the cluster
    cluster_reasons = df[df['cluster'] == cluster]['primary_call_reason'].value_counts()
    print("Top call reasons for this cluster:")
    print(cluster_reasons.head())

    # Identify potential self-service options based on top words and call reasons
    potential_options = []
    for word, _ in top_words:
        if any(word in reason.lower() for reason in cluster_reasons.index):
            potential_options.append(word)

    potential_self_service_options[cluster] = potential_options

    print("Potential self-service options for this cluster:")
    print(', '.join(potential_options))
    print("\n")
```

```python
# Analyze handle time for potential self-service options
for cluster, options in potential_self_service_options.items():
    for option in options:
        option_calls = df[(df['cluster'] == cluster) & (df['primary_call_reason'].str.contains(option, case=False, na=False))]
        if not option_calls.empty:
            avg_handle_time = option_calls['handle_time'].mean()
            print(f"Cluster {cluster}, Option '{option}' average handle time: {avg_handle_time:.2f} seconds")

# Estimate potential time savings
total_calls = len(df)
potential_self_service_calls = sum(len(df[(df['cluster'] == cluster) & (df['primary_call_reason'].str.contains('|'.join(options), case=False, na=False))])
                                   for cluster, options in potential_self_service_options.items())
potential_time_saved = sum(df[(df['cluster'] == cluster) & (df['primary_call_reason'].str.contains('|'.join(options), case=False, na=False))]['handle_time'].sum()
                           for cluster, options in potential_self_service_options.items())

print(f"\nPotential self-service calls: {potential_self_service_calls} ({potential_self_service_calls/total_calls*100:.2f}% of total calls)")
print(f"Potential time saved: {potential_time_saved/3600:.2f} hours")


print("\nProposed IVR Improvements:")
for cluster, options in potential_self_service_options.items():
    for option in options:
        print(f"1. Add a self-service option for '{option}' related inquiries")
print("2. Implement a natural language processing system to better understand customer intents")
print("3. Create a feedback loop to continuously improve IVR options based on customer interactions")
print("4. Integrate the IVR system with relevant databases for up-to-date information")
print("5. Offer callback options for complex issues that cannot be resolved through self-service")
```

🌟 Summary of Potential Time Savings 🌟

- 🚀 Potential Self-Service Calls: 11,561 calls (16.10% of total calls)
- ⏳ Potential Time Saved: 2,387.30 hours

By implementing the proposed self-service options, we could easily and efficiently manage approximately 16.10% of total calls, leading to significant time savings and enhanced operational efficiency! 💼✨

📞 IVR Improvement Suggestions 📞

1. ✈️ Self-Service for Flight & Change: Enable self-service for common ‘flight’ and ‘change’ inquiries to manage requests efficiently.
2. 🤖 Enhanced NLP: Improve intent recognition for better call routing.
3. 🔄 Feedback Loop: Continuously refine IVR options based on user feedback.
4. 📅 Database Access: Provide up-to-date flight information seamlessly and regularly.
5. 📞 Callback Feature: Introduce a callback feature for unresolved complex issues instead of holding the customer on call.

Cluster analysis

Cluster	Top Words	Top Call Reasons	Potential Self Service Options	Average Handle Time (seconds)
Cluster 0	flight, agent, customer, change, let, friday, help, would, need, im	- IRROPS: 1,205 - Voluntary Change: 839 - Seating: 517 - Mileage Plus: 361 - Communications: 299	flight, change	- flight: 882.73 - change: 594.43
Cluster 1	customer, agent, flight, im, let, united, delay, thank, like, help	- IRROPS: 1,312 - Voluntary Change: 827 - Seating: 541 - Mileage Plus: 432 - Communications: 339	flight	- flight: 1,089.13

Cluster 2	agent, customer, flight, change, let, fee, would, help, im, thank	- IRROPS: 1,970 - Voluntary Change: 1,375 - Seating: 836 - Mileage Plus: 763 - Communications: 572	flight, change	- flight: 960.20 - change: 648.66
Cluster 3	flight, agent, customer, get, let, im, like, united, thank, today	- IRROPS: 2,110 - Voluntary Change: 1,717 - Seating: 1,013 - Mileage Plus: 826 - Post-Flight: 629	flight	- flight: 876.24
Cluster 4	flight, agent, customer, change, let, would, help, like, im, day	- IRROPS: 1,605 - Voluntary Change: 1,358 - Seating: 851 - Mileage Plus:759 - Post-Flight: 543	flight, change	- flight: 932.51 - change: 607.29

Cluster 5	flight, agent, customer, change, return, let, date, help, would, im	- Voluntary Change: 654 - IRROPS: 564 - Mileage Plus: 410 - Seating: 313 - Post-Flight: 229	flight, change, date	- flight: 992.02 - change: 635.64 - date: 429.77
Cluster 6	agent, flight, customer, let, help, check, im, time, next, like	- Voluntary Change: 1,140 - IRROPS: 1,005 - Seating: 655 - Mileage Plus: 530 - Communications: 338	flight, check	- flight: 904.45 - check: 754.84
Cluster 7	customer, flight, agent, im, united, let, experience, thank, like, help	- IRROPS: 812 - Voluntary Change: 547 - Seating: 355 - Mileage Plus: 321 - Post-Flight: 224	flight	- flight: 1,015.08

Cluster 8	flight, agent, customer, let, im, like, get, help, would, change	- IRROPS: 1,498 - Voluntary Change: 1,148 - Mileage Plus: 719 - Seating: 700 - Post-Flight: 572	flight, change	- flight: 902.38 - change: 629.20
Cluster 9	flight, agent, customer, change, monday, let, would, help, im, work	- IRROPS: 976 - Voluntary Change: 686 - Seating: 442 - Mileage Plus: 366 - Communications: 254	flight, change	- flight: 842.42 - change: 630.70

📊 Explanation of Table Columns 📊

1. 🌐 **Potential Self-Service Options**
   - Flight: Options for self-service related to flight inquiries.
   - Change: Self-service options for modifying bookings.

2. ⏱️ **Average Handle Time (seconds)**
   - Flight: 842.42 seconds
   - Change: 630.70 seconds
   - This reflects the average time agents spend handling calls for each potential self-service option within the cluster.

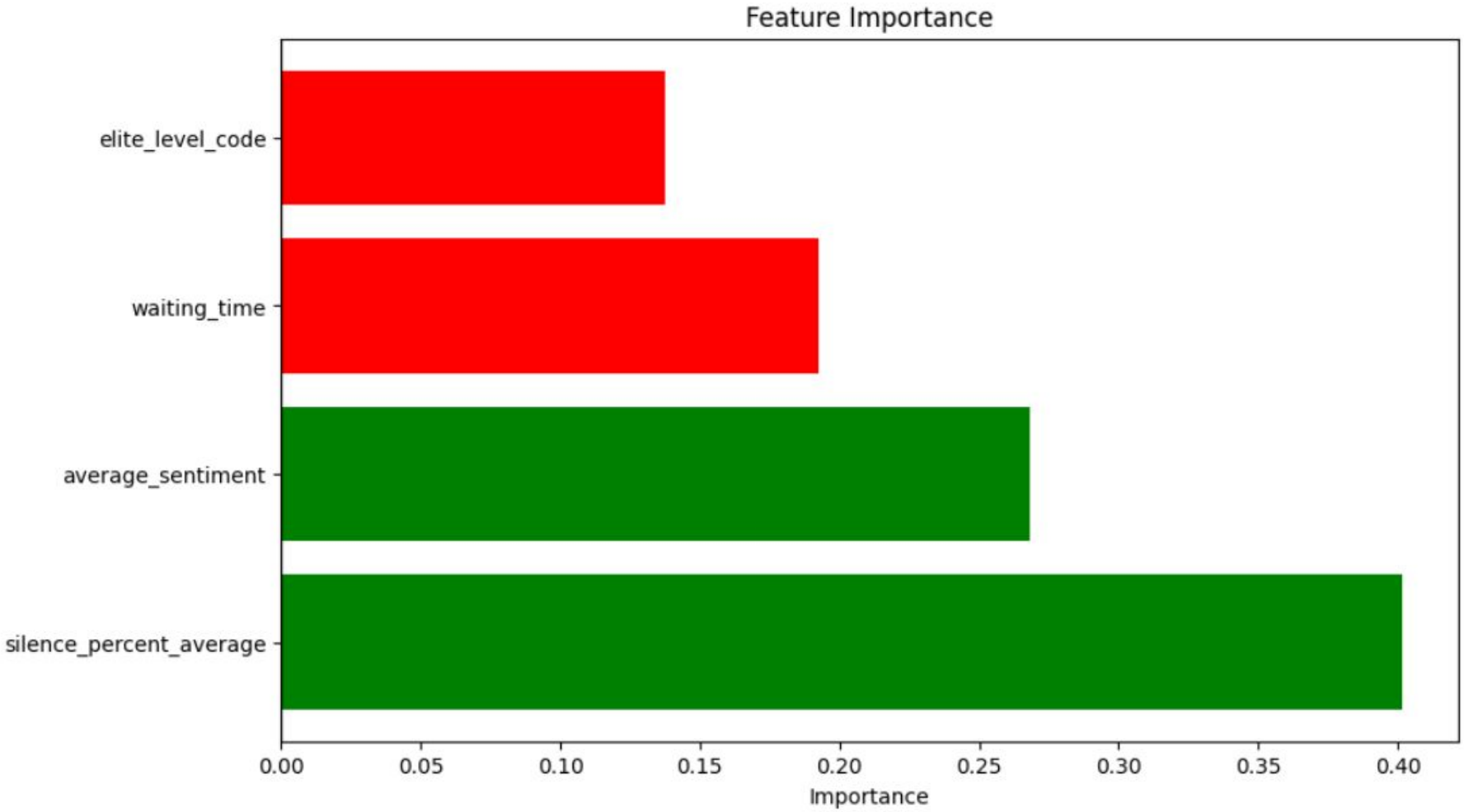
3. 🗣️ **Top Words**
   - The most frequently occurring words in the call transcripts for each cluster, highlighting common themes and topics that arise during calls.

🔎 Unveiling Common Themes and Self-Service Opportunities 📈

- **Cluster:** 🤖 Unique identifiers for 10 distinct clusters, revealing patterns in call transcripts

- **Dominant Call Reasons** 🎯📞
  a. Key call categories within clusters, such as:
     i. **IRROPS (Irregular Operations)** ✈️⚠️
     ii. **Voluntary Change** 🔄📝
     iii. **Seating Issues** 💺❌
     iv. **Mileage Plus Inquiries** 🛫💳
     v. **Communications** 📡💬
     vi. **Post-Flight Concerns** 🛬😕
     vii. **Return and Date Changes** 📅🔀
     viii. **Check-in Processes** 🛄✅

- **Average Handle Time (Seconds)** ⏱️📊
  a. **Insight:** Average duration agents spend on calls for each potential self-service topic.
  b. **Impact:** Introducing self-service here can significantly cut down handle times, boosting efficiency. ⚡


Feature Importance
elite_level_code
waiting_time
average_sentiment
silence_percent_average
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Importance

🎯 Overview of Predictive Modeling (Optional Task)

Objective🎯:
Develop a predictive model to forecast the primary call reason for incoming calls.

Purpose 📞:
Enable the call center to proactively manage resources, allocate agents more effectively, and enhance customer satisfaction by anticipating their needs.

📊 Data Preparation & Challenges 🛠️

Data Gathering:

- **Training Data**: Merged multiple datasets (calls.csv, customers.csv, sentiment_statistics.csv, reason.csv) to create a comprehensive dataset with features and target labels.
- **Test Data**: test.csv initially only had call_id.

Challenge:

🚫 Missing Features in Test Data

- **Issue**: test.csv lacked crucial features like call_transcript, essential for prediction.
- **Solution**:
  - 🔄 **Data Merging**: Combined test.csv with calls.csv on call_id to get call_transcript.
  - 🧼 Missing Values Handling:
    - Training Data: Dropped or filled missing values in processed_transcript and primary_call_reason.
    - Test Data: Filled missing call_transcript entries with empty strings to allow processing.

```
# Load datasets
calls = pd.read_csv('calls.csv')
customers = pd.read_csv('customers.csv')
sentiment = pd.read_csv('sentiment_statistics.csv')
reason = pd.read_csv('reason.csv')

# Merge datasets for training
df = (
    calls.merge(customers, on='customer_id', how='left')
         .merge(sentiment, on='call_id', how='left')
         .merge(reason, on='call_id', how='left')
)
```

🧠 Modeling Approach

Text Preprocessing:

- Converted transcripts to lowercase, removed punctuation, and eliminated stopwords.
- Result: A clean processed_transcript ready for feature extraction. 🧽✨

Feature Extraction:

- 📊 Utilized TF-IDF Vectorization to convert text data into numerical features.
- Captured the importance of words in the context of call transcripts. 📝➡️📊

Model Selection:

- 🏆 Chose Logistic Regression for its effectiveness in multi-class classification and interpretability.

Training & Validation:

- Split the data into training and validation sets (80/20 split).
- Trained the model on the training data and evaluated performance on the validation set. 🏋️📈

Prediction:

- Transformed test data using the same TF-IDF vectorizer.
- Predicted the primary_call_reason for each call in the test set. 🔮📞

```python
# Define the preprocessing function
def preprocess_text(text):
    text = str(text).lower()
    text = re.sub(r'[^\w\s]', '', text)
    stop_words = set(stopwords.words('english'))
    words = text.split()
    return ' '.join([word for word in words if word not in stop_words])

# Preprocess the call transcripts
df['processed_transcript'] = df['call_transcript'].apply(preprocess_text)

# Checking for NaNs in training data
print("Number of NaNs in each feature (training data):")
print(df[['processed_transcript', 'primary_call_reason']].isnull().sum())

# # Checking for NaNs in test data
# print("\nNumber of NaNs in 'processed_transcript' (test data):")
# print(test_df['processed_transcript'].isnull().sum())

# Drop rows with NaNs in 'processed_transcript' or 'primary_call_reason'
df = df.dropna(subset=['processed_transcript', 'primary_call_reason'])

# Feature Extraction
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X = vectorizer.fit_transform(df['processed_transcript'])
y = df['primary_call_reason']

# Train-Test Split
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Model Training
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

```python
# Save the model and vectorizer
with open('model.pkl', 'wb') as f:
    pickle.dump(model, f)
with open('vectorizer.pkl', 'wb') as f:
    pickle.dump(vectorizer, f)

# **Prediction Steps**

# Load test data
test_df = pd.read_csv('test.csv')  # Contains only 'call_id'

# Merge with 'calls.csv' to get 'call_transcript'
test_df = test_df.merge(calls[['call_id', 'call_transcript']], on='call_id', how='left')

# Check for missing transcripts and handle them
missing_transcripts = test_df['call_transcript'].isnull().sum()
print(f"Missing call transcripts: {missing_transcripts}")
test_df.dropna(subset=['call_transcript'], inplace=True)

# Preprocess the call transcripts
test_df['processed_transcript'] = test_df['call_transcript'].apply(preprocess_text)

# Load the model and vectorizer
with open('model.pkl', 'rb') as f:
    model = pickle.load(f)
with open('vectorizer.pkl', 'rb') as f:
    vectorizer = pickle.load(f)

# Transform the test data
X_new = vectorizer.transform(test_df['processed_transcript'])

# Make predictions
predictions = model.predict(X_new)
test_df['primary_call_reason'] = predictions
```

🚀 Insights Gained

Enhanced Resource Allocation:

⚙️ By predicting call reasons, the call center can allocate agents with specific expertise to handle anticipated inquiries.

⏳ Enables scheduling adjustments to meet predicted demand for certain call types. 📅

Proactive Customer Service:

🔮 **Preemptive Issue Resolution:** Address common issues before they escalate by preparing solutions in advance.

🗣️ **Personalized IVR Routing:** Tailor the IVR menu based on predicted call reasons to guide customers more efficiently. 🎯

Improved Operational Efficiency:

⏱️ **Reduced AHT and AST:** Anticipating call reasons helps in reducing average handle time and speed to answer by connecting customers to the right resources promptly.

📊 Workload Management: Balances agent workload by forecasting call volumes for specific issues.

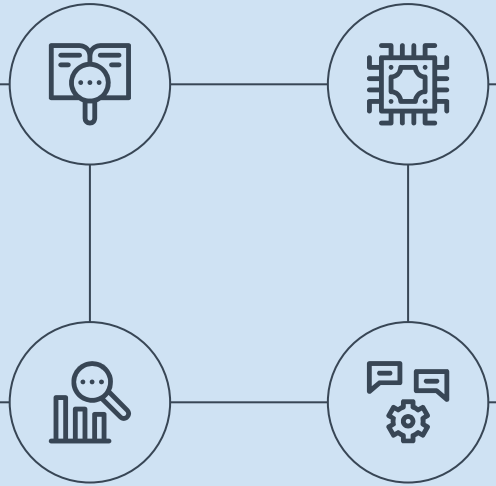
Recommendations for better performance

REGULAR FEEDBACK REVIEWS

Conduct regular reviews of call centre metrics and customer feedback to find areas of improvement

PREDICTIVE ANALYTICS

To forecast call volumes based on historical data, helping with staffing decisions and managing peak times.



PERFORMANCE METRICS

Implement metrics other than AHT & AST to evaluate agent performance like customer satisfaction scores and first call resolution rate

Linguistics

- Taking care of the region and language of different regions
- Ignoring useless and lengthy greeting sentences during conversation

Thank You!