# From Black Box to Clarity: Simplifying Aviation Conversations

Done By:

1. Surya Vinay Kumar - 002244969
2. Anagha Veena Sanjeev - 002244906
3. Sneha Manjunath Chakrabhavi - 002836841

```
import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt
import seaborn as sns
from textblob import TextBlob
from flashtext import KeywordProcessor
from collections import Counter
from google.colab import files
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from tensorflow.keras.preprocessing.sequence import pad_sequences
from nltk.translate.bleu_score import corpus_bleu
```

## Load and Inspect the Data

```
uploaded = files.upload()
```

Choose Files 3 files

- **output (1).csv**(text/csv) - 889322 bytes, last modified: 12/1/2024 - 100% done
- **Meanings (2).xlsx**(application/vnd.openxmlformats-officedocument.spreadsheetml.sheet) - 25116 bytes, last modified: 11/30/2024 - 100% done
- **transcriptions.xlsx**(application/vnd.openxmlformats-officedocument.spreadsheetml.sheet) - 160353 bytes, last modified: 11/30/2024 - 100% done

```
Saving output (1).csv to output (1) (1).csv
Saving Meanings (2).xlsx to Meanings (2) (1).xlsx
Saving transcriptions.xlsx to transcriptions (1).xlsx
```

```
meanings_df = pd.read_excel('Meanings (2) (1).xlsx')
transcriptions = pd.read_excel('transcriptions.xlsx')
```

```
# Load datasets
transcriptions = transcriptions  # Contains "transcription"
wordlist = meanings_df  # Contains "Words" and "Meanings"

# Display basic information
print("Transcriptions Dataset Info:")
print(transcriptions.info())
print("\nWordlist Dataset Info:")
print(wordlist.info())

# Display first few rows
print("\nSample Transcriptions:")
print(transcriptions.head())
print("\nSample Wordlist:")
print(wordlist.head())
```

```
Transcriptions Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7595 entries, 0 to 7594
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   sentiment        7595 non-null   float64
 1   simplified_text  7595 non-null   object 
 2   transcription    7595 non-null   object 
dtypes: float64(1), object(2)
memory usage: 178.1+ KB
None

Wordlist Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 696 entries, 0 to 695
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   Word     696 non-null    object
 1   Meaning  696 non-null    object
dtypes: object(2)
memory usage: 11.0+ KB
None

Sample Transcriptions:
   sentiment                                 simplified_text  \
0   0.285714               psa 810 Turn Right To Trasadingen   
1   0.000000       Lufthansa 5318 Contact Zurich at 134.6 Hz   
2   0.000000              psa 810 Contact Zurich at 133.4 Hz   
3   0.000000                Sabena 481 rhein area Identified   
4   0.000000  Transwede 101 rhein area Identified Set Toward...   

                                       transcription  
0       psa eight one zero turn right to trasadingen  
1  lufthansa five three one eight contact zurich ...  
2  psa eight one zero contact zurich one three th...  
3             sabena four eight one rhein identified  
4  transwede one zero one rhein identified set co...  

Sample Wordlist:
      Word  Meaning
0     Able     Able
1    About    About
2    Above    Above
3   Accept   Accept
4  Address  address
```

## Check for missing values

```
# Check for missing values
print("\nMissing Values in Transcriptions:")
print(transcriptions.isnull().sum())

print("\nMissing Values in Wordlist:")
print(wordlist.isnull().sum())
```

```
Missing Values in Transcriptions:
sentiment          0
simplified_text    0
transcription      0
dtype: int64

Missing Values in Wordlist:
Word       0
Meaning    0
dtype: int64
```

## Basic Statistics

```
# Transcriptions dataset
print("\nTranscriptions Dataset Stats:")
print(f"Total Transcriptions: {len(transcriptions)}")
print(f"Average Length of Transcriptions: {transcriptions['transcription'].apply(len).mean():.2f}")

# Wordlist dataset
print("\nWordlist Dataset Stats:")
print(f"Total Words in Wordlist: {len(wordlist)}")
print(f"Unique Words in Wordlist: {wordlist['Word'].nunique()}")
print(f"Average Length of Words: {wordlist['Word'].apply(len).mean():.2f}")
```

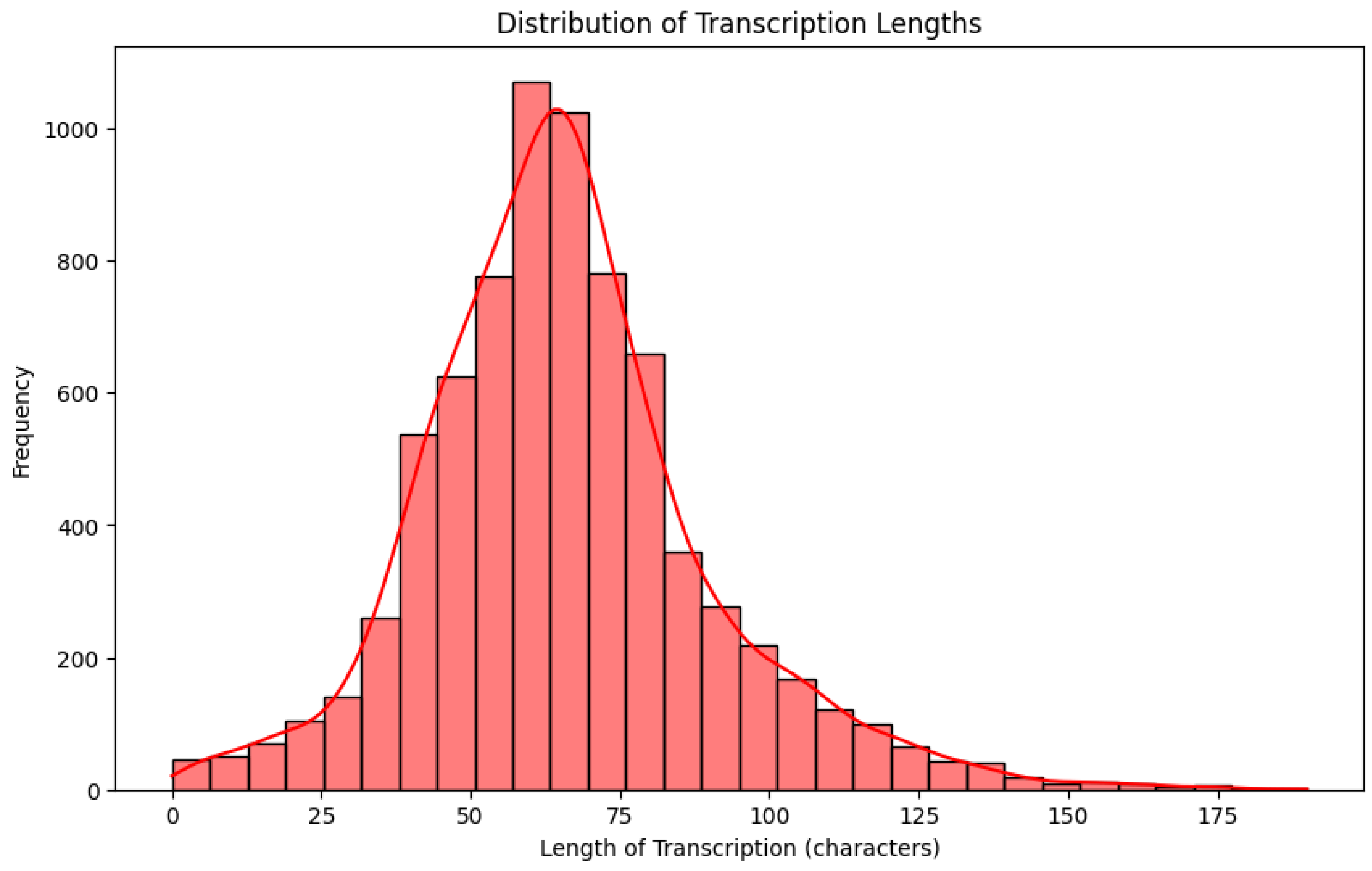
```
Transcriptions Dataset Stats:
Total Transcriptions: 7595
Average Length of Transcriptions: 65.79

Wordlist Dataset Stats:
Total Words in Wordlist: 696
Unique Words in Wordlist: 696
Average Length of Words: 5.86
```

```
# Function to remove specific words from text
def remove_words(text, words_to_remove):
    # Split the text into words
    word_list = text.split()
    # Filter out the words to be removed
    filtered_words = [word for word in word_list if word.lower() not in words_to_remove]
    # Join the words back into a string
    return " ".join(filtered_words)

# List of words to remove
words_to_remove = {'ah', 'oh', 'ot', 'fl'}

# Apply the function to the transcription column
transcriptions['transcription_cleaned'] = transcriptions['transcription'].apply(lambda x: remove_words(x, words_to_remove))

# Display the updated DataFrame
print(transcriptions.head())

# Save to Excel file if needed
output_file = 'transcriptions_cleaned.xlsx'
transcriptions.to_excel(output_file, index=False)
print(f"Cleaned transcriptions saved to {output_file}")
```

```
    sentiment                                    simplified_text  \
0   0.285714                  psa 810 Turn Right To Trasadingen
1   0.000000          Lufthansa 5318 Contact Zurich at 134.6 Hz
2   0.000000                 psa 810 Contact Zurich at 133.4 Hz
3   0.000000                   Sabena 481 rhein area Identified
4   0.000000  Transwede 101 rhein area Identified Set Toward...

                                       transcription  \
0       psa eight one zero turn right to trasadingen
1  lufthansa five three one eight contact zurich ...
2  psa eight one zero contact zurich one three th...
3            sabena four eight one rhein identified
4  transwede one zero one rhein identified set co...

                               transcription_cleaned
0       psa eight one zero turn right to trasadingen
1  lufthansa five three one eight contact zurich ...
2  psa eight one zero contact zurich one three th...
3            sabena four eight one rhein identified
4  transwede one zero one rhein identified set co...
Cleaned transcriptions saved to transcriptions_cleaned.xlsx
```

## Distribution of Transcription Lengths

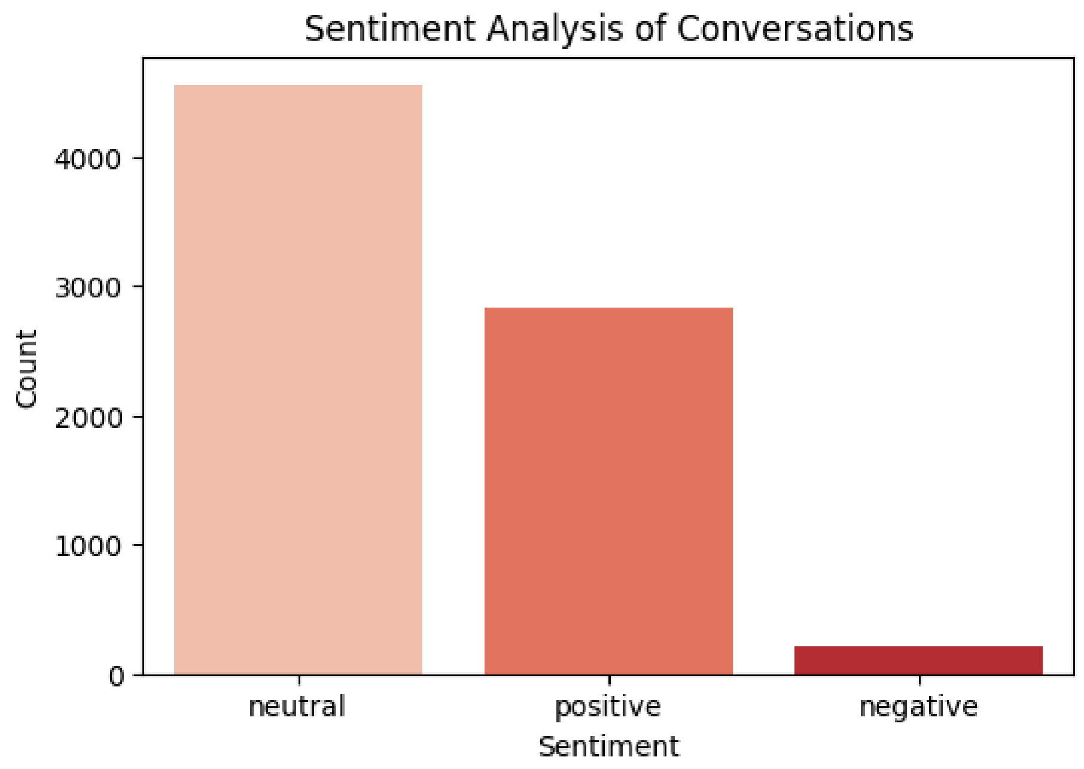
```
# Plot distribution of transcription lengths
# Calculate transcription lengths
transcriptions['length'] = transcriptions['transcription_cleaned'].apply(len)

# Plot distribution
plt.figure(figsize=(10, 6))
sns.histplot(transcriptions['length'], bins=30, kde=True, color='red')
plt.title('Distribution of Transcription Lengths')
plt.xlabel('Length of Transcription (characters)')
plt.ylabel('Frequency')
plt.show()
```



## Sentiment Analysis of Conversations

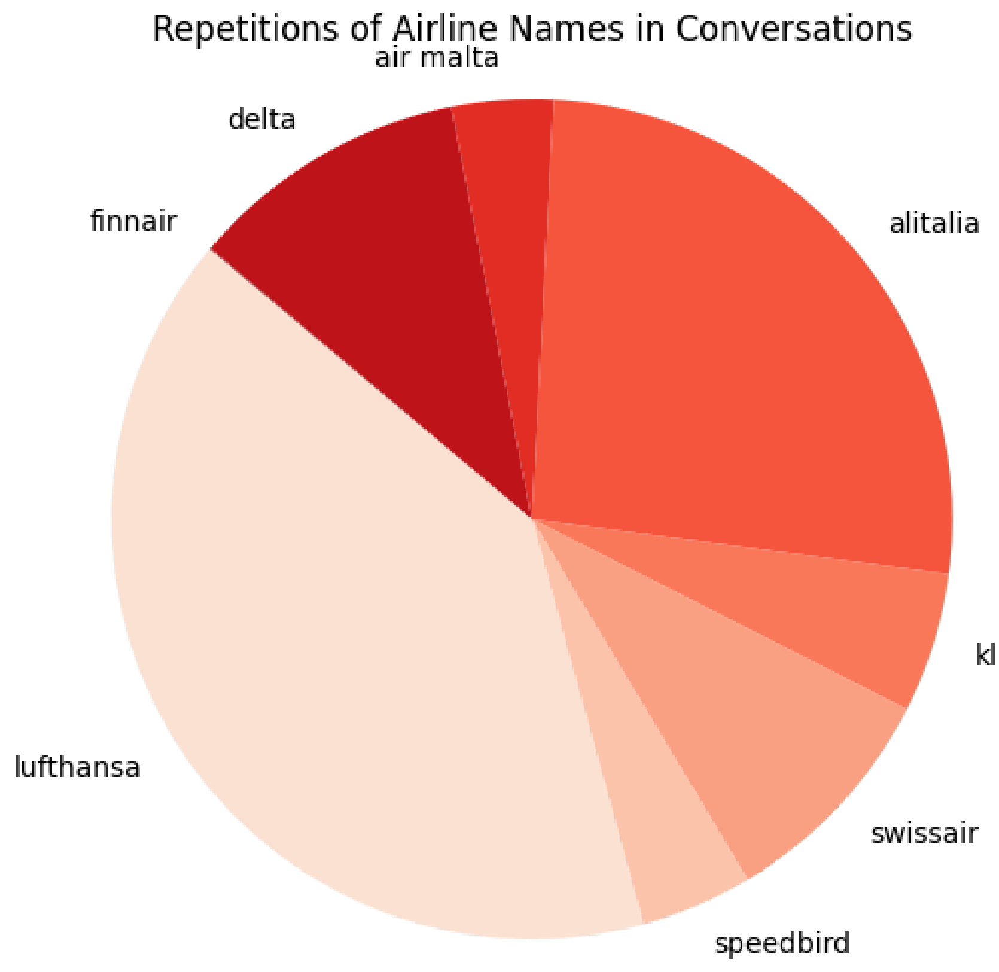
```
transcriptions['sentiment'] = transcriptions['transcription'].apply(lambda x: TextBlob(x).sentiment.polarity)
sentiment_counts = transcriptions['sentiment'].apply(lambda x: 'positive' if x > 0 else 'negative' if x < 0 else 'neutral').value_counts()

plt.figure(figsize=(6, 4))
sns.barplot(x=sentiment_counts.index, y=sentiment_counts.values, palette='Reds')
plt.title('Sentiment Analysis of Conversations')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()
```

```
<ipython-input-51-a50043251883>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=sentiment_counts.index, y=sentiment_counts.values, palette='Reds')
```



## Repetitions of Airline Names in Conversations

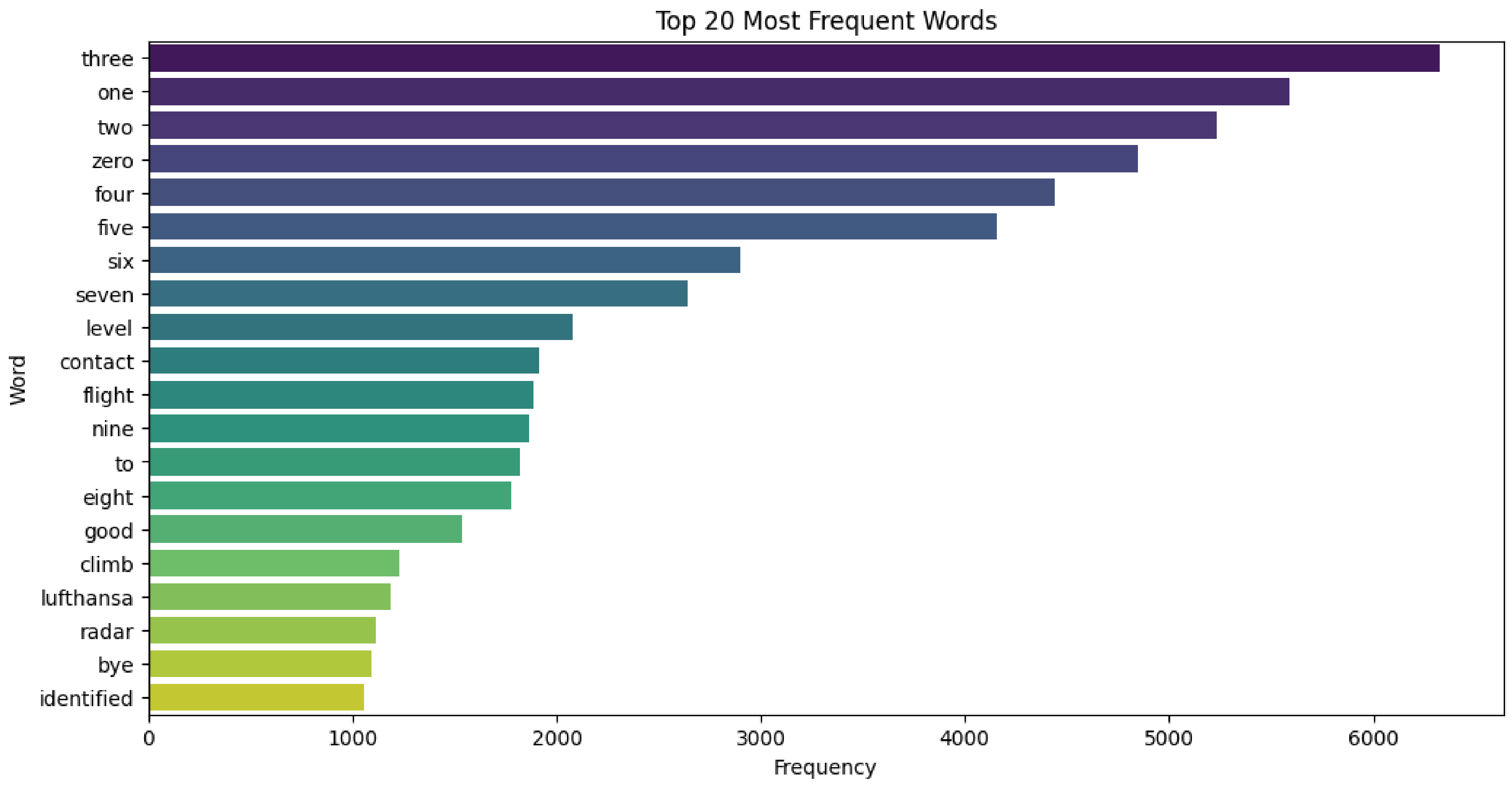
```
t1 = pd.read_excel('transcriptions.xlsx')
airline_names = [
    "finnair", "delta", "lufthansa", "swissair",
    "klm" , "air malta", "alitalia", "speedbird"
]

keyword_processor = KeywordProcessor()
for name in airline_names:
    keyword_processor.add_keyword(name.lower())
all_transcriptions = t1['transcription'].tolist()
extracted_airlines = []

for transcription in all_transcriptions:
    extracted_airlines.extend(keyword_processor.extract_keywords(transcription.lower()))  # Convert to lowercase for case-insensitive matching

airline_counts = Counter(extracted_airlines)
airline_labels = list(airline_counts.keys())
airline_sizes = list(airline_counts.values())

plt.figure(figsize=(9, 6))
plt.pie(airline_sizes, labels=airline_labels, startangle=140, colors=sns.color_palette('Reds', len(airline_labels)), pctdistance=0.85)
plt.axis('equal')
plt.title('Repetitions of Airline Names in Conversations')
plt.show()
```



## Word Frequency Analysis

```
# Tokenize all transcriptions and count word frequencies
all_words = " ".join(transcriptions['transcription_cleaned'].str.lower().tolist()).split()
word_counts = Counter(all_words)

# Convert to DataFrame for analysis
word_freq_df = pd.DataFrame(word_counts.items(), columns=['Word', 'Frequency']).sort_values(by='Frequency', ascending=False)

# Top 20 most frequent words
print("Top 20 Most Frequent Words:")
print(word_freq_df.head(20))

# Plot top 20 most frequent words
plt.figure(figsize=(12, 6))
sns.barplot(x='Frequency', y='Word', data=word_freq_df.head(20), palette='viridis')
plt.title("Top 20 Most Frequent Words")
plt.xlabel("Frequency")
plt.ylabel("Word")
plt.show()
```

```
Top 20 Most Frequent Words:
           Word  Frequency
10        three       6322
2           one       5587
25          two       5234
3          zero       4849
13         four       4438
9          five       4152
15          six       2901
26        seven       2641
36        level       2079
11      contact       1917
35       flight       1886
31         nine       1862
6            to       1820
1         eight       1778
113        good       1532
34        climb       1228
8     lufthansa       1184
28        radar       1115
114         bye       1093
18   identified       1056
<ipython-input-53-7ec43fb3c0cc>:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x='Frequency', y='Word', data=word_freq_df.head(20), palette='viridis')
```



```
transcriptions = transcriptions.drop(columns=['length'])
transcriptions = transcriptions.drop(columns=['transcription'])
transcriptions = transcriptions.rename(columns={'transcription_cleaned': 'transcription'})
```

## Text Simplification Using a Dictionary and Regular Expressions

```
# Create a dictionary mapping from word to simplified version
meanings_dict = dict(zip(wordlist['Word'].str.lower(), wordlist['Meaning']))

def simplify_text(text):
    # Step 1: Split text into words
    words = text.split()
    simplified_words = []

    # Step 2: Simplify each word using the dictionary (convert to string to avoid issues)
    for word in words:
        simplified_word = meanings_dict.get(word.lower(), word)
        simplified_words.append(str(simplified_word))  # Ensure each word is a string

    # Step 3: Combine consecutive numeric values into a single group
    combined_text = []
    temp_numeric_group = ""

    for word in simplified_words:
        if word.isdigit():  # If the word is numeric
            if temp_numeric_group == "":  # Start a new numeric group
                temp_numeric_group = word
            else:
                temp_numeric_group += word  # Add to the existing numeric group
        else:
            if temp_numeric_group:  # If there's a numeric group, append it
                combined_text.append(temp_numeric_group)
                temp_numeric_group = ""  # Reset the numeric group
            combined_text.append(word)  # Add non-numeric word

    # Add any remaining numeric group
    if temp_numeric_group:
        combined_text.append(temp_numeric_group)

    # Step 4: Join words into a single string
    simplified_text = " ".join(combined_text)

    # Step 5: Modify the simplified text to add "at" before and "Hz" after decimal numbers
    # Correctly format decimal numbers by ensuring proper spacing
    simplified_text = re.sub(r'(\d+)\s*\.\s*(\d+)', r'at \1.\2 Hz', simplified_text)

    return simplified_text

# Apply the simplify_text function
transcriptions['simplified_text'] = transcriptions['transcription'].apply(simplify_text)

# Output the simplified data
```

```
transcriptions[['transcription','simplified_text']].head()
```

1 to 5 of 5 entries Filter

| index ▲ | transcription | simplified_text |
|---|---|---|
| 0 | psa eight one zero turn right to trasadingen | psa 810 Turn Right To Trasadingen |
| 1 | lufthansa five three one eight contact zurich one three four decimal six | Lufthansa 5318 Contact Zurich at 134.6 Hz |
| 2 | psa eight one zero contact zurich one three three decimal four | psa 810 Contact Zurich at 133.4 Hz |
| 3 | sabena four eight one rhein identified | Sabena 481 rhein area Identified |
| 4 | transwede one zero one rhein identified set course trasadingen | Transwede 101 rhein area Identified Set Towards Trasadingen |

Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

```
output_file = 'simplified_transcriptions.xlsx'

# Export the DataFrame with simplified text to an Excel file
transcriptions[['transcription', 'simplified_text']].to_excel(output_file, index=False)

# Inform the user that the file has been saved
print(f"Excel file saved as {output_file}")
```

```
Excel file saved as simplified_transcriptions.xlsx
```

## Model Training

```
from google.colab import files
uploaded = files.upload()
```

Choose Files output (1).csv

- **output (1).csv**(text/csv) - 889322 bytes, last modified: 12/1/2024 - 100% done

```
Saving output (1).csv to output (1).csv
```

```
df_output = pd.read_csv('output (1).csv')
```

```
# Step 1: Loading dataset

# Extract the relevant columns
input_texts = df_output['transcription_cleaned'].tolist()
simplified_texts = df_output['simplified_text'].tolist()

# Step 2: Preprocess the input text to handle numbers and non-string values
def preprocess_input_text(input_text):
    if not isinstance(input_text, str):
        return ""  # Replace non-string entries with an empty string
    # Replace all numeric sequences with their proper format (e.g., 292 as '292')
    input_text = re.sub(r'\d+', lambda x: str(int(x.group())), input_text)
    return input_text

# Apply preprocessing to the input texts
input_texts = [preprocess_input_text(text) for text in input_texts]

# Step 3: Create Tokenizers for both input and output texts
input_tokenizer = Tokenizer(oov_token="<OOV>")
input_tokenizer.fit_on_texts(input_texts)

y_tokenizer = Tokenizer(oov_token="<OOV>")
y_tokenizer.fit_on_texts(simplified_texts)

# Step 4: Convert text to sequences
input_sequences = input_tokenizer.texts_to_sequences(input_texts)
output_sequences = y_tokenizer.texts_to_sequences(simplified_texts)

# Find the maximum sequence length for padding
max_input_length = max([len(seq) for seq in input_sequences])
max_output_length = max([len(seq) for seq in output_sequences])

# Step 5: Pad the sequences to ensure consistent length
input_sequences_padded = pad_sequences(input_sequences, maxlen=max_input_length, padding='post')
# Pad output sequences to match the model's output length (38)
output_sequences_padded = pad_sequences(output_sequences, maxlen=38, padding='post')

# Step 6: Build the model
model = Sequential()

# Embedding layer for the input sequence
model.add(Embedding(input_dim=len(input_tokenizer.word_index) + 1, output_dim=100, input_length=max_input_length))

# LSTM layer
model.add(LSTM(units=128, return_sequences=True))

# Dropout for regularization
model.add(Dropout(0.2))

# Dense layer
model.add(Dense(64, activation='relu'))

# Output layer: Predict one token at each time step
model.add(Dense(len(y_tokenizer.word_index) + 1, activation='softmax'))  # Output size matches vocab size of target

# Compile the model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Step 7: Train the model
history = model.fit(input_sequences_padded, output_sequences_padded, epochs=20, batch_size=32, validation_split=0.2)

# Step 8: Evaluate the model on the test data
test_loss, test_acc = model.evaluate(input_sequences_padded, output_sequences_padded)
print(f'Test Loss: {test_loss}')
print(f'Test Accuracy: {test_acc}')

# Function to simplify new input text
def simplify_new_input(input_text, input_tokenizer, y_tokenizer, max_input_length, model):
    # Step 1: Preprocess the input (tokenize and pad)
    input_seq = input_tokenizer.texts_to_sequences([input_text])  # Convert text to sequence
    input_padded = pad_sequences(input_seq, maxlen=max_input_length, padding='post')  # Pad the sequence

    # Step 2: Make predictions using the trained model
    prediction = model.predict(input_padded)

    # Step 3: Convert the predicted sequence back to text
    predicted_sequence = prediction.argmax(axis=-1)  # Get the predicted token indices
    simplified_text = y_tokenizer.sequences_to_texts(predicted_sequence)  # Convert indices to words

    # Step 4: Remove the <OOV> tokens from the output
    simplified_text_clean = ' '.join([word for word in simplified_text[0].split() if word != "<OOV>"])

    return simplified_text_clean
```

```
Epoch 1/20
190/190 ━━━━━━━━━━━━━━━━━━━━ 31s 146ms/step - accuracy: 0.7525 - loss: 3.0761 - val_accuracy: 0.7736 - val_loss: 1.3912
Epoch 2/20
190/190 ━━━━━━━━━━━━━━━━━━━━ 42s 151ms/step - accuracy: 0.7861 - loss: 1.2623 - val_accuracy: 0.7818 - val_loss: 1.2598
Epoch 3/20
190/190 ━━━━━━━━━━━━━━━━━━━━ 40s 149ms/step - accuracy: 0.7933 - loss: 1.1109 - val_accuracy: 0.7952 - val_loss: 1.1497
Epoch 4/20
190/190 ━━━━━━━━━━━━━━━━━━━━ 40s 141ms/step - accuracy: 0.8084 - loss: 0.9570 - val_accuracy: 0.7993 - val_loss: 1.0780
Epoch 5/20
190/190 ━━━━━━━━━━━━━━━━━━━━ 42s 145ms/step - accuracy: 0.8172 - loss: 0.8911 - val_accuracy: 0.8085 - val_loss: 1.0296
Epoch 6/20
190/190 ━━━━━━━━━━━━━━━━━━━━ 40s 142ms/step - accuracy: 0.8242 - loss: 0.8420 - val_accuracy: 0.8146 - val_loss: 1.0011
Epoch 7/20
190/190 ━━━━━━━━━━━━━━━━━━━━ 41s 140ms/step - accuracy: 0.8363 - loss: 0.7792 - val_accuracy: 0.8180 - val_loss: 0.9647
Epoch 8/20
190/190 ━━━━━━━━━━━━━━━━━━━━ 27s 141ms/step - accuracy: 0.8394 - loss: 0.7537 - val_accuracy: 0.8203 - val_loss: 0.9573
Epoch 9/20
190/190 ━━━━━━━━━━━━━━━━━━━━ 41s 143ms/step - accuracy: 0.8460 - loss: 0.7199 - val_accuracy: 0.8236 - val_loss: 0.9379
Epoch 10/20
190/190 ━━━━━━━━━━━━━━━━━━━━ 41s 144ms/step - accuracy: 0.8476 - loss: 0.7085 - val_accuracy: 0.8270 - val_loss: 0.9225
Epoch 11/20
190/190 ━━━━━━━━━━━━━━━━━━━━ 40s 142ms/step - accuracy: 0.8540 - loss: 0.6733 - val_accuracy: 0.8318 - val_loss: 0.9067
Epoch 12/20
190/190 ━━━━━━━━━━━━━━━━━━━━ 41s 141ms/step - accuracy: 0.8586 - loss: 0.6426 - val_accuracy: 0.8322 - val_loss: 0.9036
Epoch 13/20
190/190 ━━━━━━━━━━━━━━━━━━━━ 27s 141ms/step - accuracy: 0.8599 - loss: 0.6322 - val_accuracy: 0.8337 - val_loss: 0.8961
Epoch 14/20
190/190 ━━━━━━━━━━━━━━━━━━━━ 28s 150ms/step - accuracy: 0.8602 - loss: 0.6326 - val_accuracy: 0.8348 - val_loss: 0.8862
Epoch 15/20
190/190 ━━━━━━━━━━━━━━━━━━━━ 40s 146ms/step - accuracy: 0.8651 - loss: 0.5999 - val_accuracy: 0.8365 - val_loss: 0.8787
Epoch 16/20
190/190 ━━━━━━━━━━━━━━━━━━━━ 41s 146ms/step - accuracy: 0.8649 - loss: 0.5963 - val_accuracy: 0.8372 - val_loss: 0.8804
Epoch 17/20
190/190 ━━━━━━━━━━━━━━━━━━━━ 52s 204ms/step - accuracy: 0.8658 - loss: 0.5869 - val_accuracy: 0.8388 - val_loss: 0.8702
Epoch 18/20
190/190 ━━━━━━━━━━━━━━━━━━━━ 53s 265ms/step - accuracy: 0.8699 - loss: 0.5657 - val_accuracy: 0.8400 - val_loss: 0.8682
Epoch 19/20
190/190 ━━━━━━━━━━━━━━━━━━━━ 58s 140ms/step - accuracy: 0.8697 - loss: 0.5612 - val_accuracy: 0.8403 - val_loss: 0.8635
Epoch 20/20
190/190 ━━━━━━━━━━━━━━━━━━━━ 40s 137ms/step - accuracy: 0.8703 - loss: 0.5560 - val_accuracy: 0.8418 - val_loss: 0.8652
```

```
238/238 ━━━━━━━━━━━━━━━━━━━━ 14s 60ms/step - accuracy: 0.8839 - loss: 0.4882
Test Loss: 0.5891261696815491
Test Accuracy: 0.8698136806488037
```

## Model Testing

```
def simplify_new_input(input_text, input_tokenizer, y_tokenizer, max_input_length, model):
    # Step 1: Preprocess the input (tokenize and pad)
    input_seq = input_tokenizer.texts_to_sequences([input_text])  # Convert text to sequence
    input_padded = pad_sequences(input_seq, maxlen=max_input_length, padding='post')  # Pad the sequence

    # Step 2: Make predictions using the trained model
    prediction = model.predict(input_padded)

    # Step 3: Convert the predicted sequence back to text
    predicted_sequence = prediction.argmax(axis=-1)  # Get the predicted token indices
    simplified_text = y_tokenizer.sequences_to_texts(predicted_sequence)  # Convert indices to words

    # Step 4: Remove the <OOV> tokens from the output
    simplified_text_clean = ' '.join([word for word in simplified_text[0].split() if word != "<OOV>"])
    simplified_text_clean.head()
    return simplified_text_clean

# Test the function with an example input

input_text = df_output
simplified_output = simplify_new_input(input_text, input_tokenizer, y_tokenizer, max_input_length, model)
```

1 to 5 of 5 entries Filter

| index | Transcription | Simplified English |
|---|---|---|
| 0 | psa eight one zero turn right to trasadingen | PSA 810, turn right toward Trasadingen. |
| 1 | lufthansa five three one eight contact zurich one three four decimal six | Lufthansa 5318, contact Zurich on frequency 134.6. |
| 2 | psa eight one zero contact zurich one three three decimal four | PSA 810, contact Zurich on frequency 133.4. |
| 3 | sabena four eight one rhein identified | Sabena 481, Rhein Radar, identified. |
| 4 | transwede one zero one rhein identified set course trasadingen | Transwede 101, Rhein Radar, identified. Set course toward Trasadingen. |

Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.