```
# Install dependencies with explicit version control
print("Installing dependencies...")
!pip install numpy==1.26.4
!pip install pydub==0.25.1 google-cloud-speech==2.25.0 pandas==2.2.3 matplotlib==
!pip install textblob==0.18.0.post0
!pip install gensim==4.3.3
!apt-get update && apt-get install -y ffmpeg

# Verify installed packages
import pkg_resources
required_packages = ['numpy', 'pydub', 'google-cloud-speech', 'pandas', 'matplotl
print("\nPackage versions:")
for pkg in required_packages:
    try:
        version = pkg_resources.get_distribution(pkg).version
        print(f"{pkg}: {version}")
    except pkg_resources.DistributionNotFound:
        print(f"{pkg}: Not installed")

# Import libraries with error handling
import os
import io
import sys
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import json

# Suppress warnings
warnings.filterwarnings('ignore')

# Try importing specialized libraries with better error handling
try:
    import librosa
    import librosa.display
    from pydub import AudioSegment
    from google.cloud import speech
    import nltk
    from nltk.tokenize import word_tokenize, sent_tokenize
    from nltk.corpus import stopwords
    from nltk.sentiment.vader import SentimentIntensityAnalyzer
    from textblob import TextBlob
    from sklearn.preprocessing import StandardScaler
    from sklearn.decomposition import PCA
    from sklearn.cluster import KMeans, DBSCAN
    from sklearn.ensemble import IsolationForest
    from sklearn.metrics.pairwise import cosine_similarity
    from gensim import corpora
    from gensim.models import LdaModel

    print("All critical libraries imported successfully.")
except ImportError as e:
    print(f"Critical import error: {e}")
    print("Please restart the runtime and ensure all dependencies are installed."
    sys.exit(1)

# Download NLTK resources with better error handling
try:
    nltk.download('punkt', quiet=True)
    nltk.download('stopwords', quiet=True)
    nltk.download('vader_lexicon', quiet=True)
except Exception as e:
    print(f"Error downloading NLTK resources: {e}")
    print("Will attempt to continue, but some NLP features may not work.")
```

Show hidden output

```
import os

# Replace 'MyNewFolder' with your desired folder name
folder_name = "audio2_folder"

# Create the folder
os.makedirs(folder_name, exist_ok=True)

print(f"Folder '{folder_name}' created successfully!")
```

```
Folder 'audio2_folder' created successfully!
```

pca_visualization.png

Voice Analysis Report

Summary

Processed 14 audio samples. Identified 2 clusters:

- Cluster 1: 8 samples
- Cluster 0: 6 samples Detected 14 potential anomalies.

Detailed Results

Filename	Risk Score	Cluster	Anomaly	Similarity Score
260-123288-0000.mp3	1.00	1	Yes	0.50
237-134493-0015.mp3	1.00	1	Yes	0.08
237-134500-0006.mp3	1.00	1	Yes	0.18
260-123288-0003.mp3	1.00	0	Yes	0.24
260-123288-0002.mp3	1.00	0	Yes	0.10
237-134500-0002.mp3	1.00	0	Yes	-0.76
237-134493-0014.mp3	1.00	1	Yes	0.27
237-134500-0004.mp3	1.00	1	Yes	-0.05
237-134500-0005.mp3	1.00	1	Yes	-0.17
237-134500-0007.mp3	1.00	0	Yes	-0.33
237-134500-0003.mp3	1.00	0	Yes	-0.40
260-123288-0001.mp3	1.00	0	Yes	-0.00
237-134500-0010.mp3	1.00	1	Yes	-0.02
237-134500-0011.mp3	1.00	1	Yes	-0.31

```
# Enhanced visualization functions to add to the original code

def generate_visualizations(features_df, results, output_dir="."):
    """Generate comprehensive visualizations for voice analysis results"""
    import matplotlib.pyplot as plt
    import seaborn as sns
    import numpy as np
    import pandas as pd
    from sklearn.decomposition import PCA
    from sklearn.manifold import TSNE
    import os

    # Ensure output directory exists
    os.makedirs(output_dir, exist_ok=True)

    # Check if we have enough data to create visualizations
    if features_df.empty or len(results) < 2:
        print("Not enough data for visualizations")
        return

    # 1. Feature distribution heatmap
    try:
        plt.figure(figsize=(12, 10))
        # Select only numeric columns
        numeric_features = features_df.select_dtypes(include=['float64', 'int64']
        # Normalize features for better visualization
        normalized_features = (numeric_features - numeric_features.mean()) / nume
        sns.heatmap(normalized_features.T, cmap="viridis",
                    xticklabels=[r['filename'] for r in results],
                    yticklabels=normalized_features.columns)
        plt.title("Feature Distribution Across Samples")
        plt.xticks(rotation=45)
        plt.tight_layout()
        plt.savefig(os.path.join(output_dir, "feature_heatmap.png"))
        plt.close()
        print("Created feature heatmap visualization")
    except Exception as e:
        print(f"Error creating feature heatmap: {e}")

    # 2. Risk score correlation with features
    try:
        risk_scores = np.array([r.get('risk_score', 0) for r in results])
        correlation_data = []

        for col in numeric_features.columns:
            if numeric_features[col].nunique() > 1:  # Skip constant features
                corr = np.corrcoef(numeric_features[col], risk_scores)[0, 1]
                correlation_data.append({'Feature': col, 'Correlation': corr})

        if correlation_data:
            corr_df = pd.DataFrame(correlation_data)
            plt.figure(figsize=(12, 8))
            sns.barplot(x='Correlation', y='Feature', data=corr_df.sort_values('C
            plt.title("Feature Correlation with Risk Score")
            plt.axvline(x=0, color='r', linestyle='--')
            plt.tight_layout()
            plt.savefig(os.path.join(output_dir, "risk_correlation.png"))
            plt.close()
            print("Created risk correlation visualization")
    except Exception as e:
        print(f"Error creating correlation plot: {e}")

    # 3. Dimensionality reduction visualizations
    try:
        # Create a DataFrame with all relevant information
        viz_df = pd.DataFrame({
            'filename': [r['filename'] for r in results],
            'risk_score': [r.get('risk_score', 0) for r in results],
            'kmeans_cluster': [r.get('kmeans_cluster', 0) for r in results],
            'is_anomaly': [1 if r.get('iso_label', 0) == -1 or r.get('dbscan_labe
        })

        # Add features
        for col in numeric_features.columns:
            viz_df[col] = numeric_features[col].values

        # PCA visualization
        if len(viz_df) >= 2 and numeric_features.shape[1] >= 2:
            pca = PCA(n_components=2)
            pca_result = pca.fit_transform(numeric_features)

            plt.figure(figsize=(12, 10))

            # Create scatter plot
            scatter = plt.scatter(pca_result[:, 0], pca_result[:, 1]
```

```
        scatter = plt.scatter(pca_result[:, 0], pca_result[:, 1],
                    c=viz_df['risk_score'], cmap='viridis',
                    s=100 + 100 * viz_df['is_anomaly'],
                    alpha=0.7)

        # Add colorbar
        cbar = plt.colorbar(scatter)
        cbar.set_label('Risk Score')

        # Add labels
        for i, txt in enumerate(viz_df['filename']):
            plt.annotate(txt, (pca_result[i, 0], pca_result[i, 1]),
                         fontsize=9, alpha=0.7)

        plt.title(f"PCA Visualization (explained variance: {sum(pca.explained_
        plt.xlabel(f"PC1 ({pca.explained_variance_ratio_[0]:.2f})")
        plt.ylabel(f"PC2 ({pca.explained_variance_ratio_[1]:.2f})")
        plt.grid(True, linestyle='--', alpha=0.5)
        plt.tight_layout()
        plt.savefig(os.path.join(output_dir, "pca_visualization.png"))
        plt.close()
        print("Created PCA visualization")
except Exception as e:
    print(f"Error creating PCA visualization: {e}")

# 4. t-SNE visualization (for more complex relationships)
try:
    if len(viz_df) >= 2 and numeric_features.shape[1] >= 2:
        # t-SNE requires a bit more data to be meaningful, but let's include
        tsne = TSNE(n_components=2, random_state=42, perplexity=min(30, len(v
        tsne_result = tsne.fit_transform(numeric_features)

        plt.figure(figsize=(12, 10))
        scatter = plt.scatter(tsne_result[:, 0], tsne_result[:, 1],
                    c=viz_df['kmeans_cluster'], cmap='tab10',
                    s=100 + 100 * viz_df['is_anomaly'],
                    alpha=0.7)

        # Add legend for clusters
        legend1 = plt.legend(*scatter.legend_elements(),
                             title="Clusters", loc="upper right")
        plt.gca().add_artist(legend1)

        # Add markers for anomalies
        anomaly_marker = plt.Line2D([], [], color='k', marker='o', linestyle=
                                    markersize=12, label='Anomaly')
        normal_marker = plt.Line2D([], [], color='k', marker='o', linestyle='
                                   markersize=8, label='Normal')
        plt.legend(handles=[anomaly_marker, normal_marker], title="Anomaly St

        # Add labels
        for i, txt in enumerate(viz_df['filename']):
            plt.annotate(txt, (tsne_result[i, 0], tsne_result[i, 1]),
                         fontsize=9, alpha=0.7)

        plt.title("t-SNE Visualization of Audio Samples")
        plt.grid(True, linestyle='--', alpha=0.5)
        plt.tight_layout()
        plt.savefig(os.path.join(output_dir, "tsne_visualization.png"))
        plt.close()
        print("Created t-SNE visualization")
except Exception as e:
    print(f"Error creating t-SNE visualization: {e}")

# 5. Feature importance visualization
try:
    if 'risk_score' in viz_df.columns:
        # Create high and low risk groups
        viz_df['risk_group'] = (viz_df['risk_score'] > viz_df['risk_score'].m

        plt.figure(figsize=(14, 8))

        # Select top features
        top_features = numeric_features.columns[:min(10, len(numeric_features

        # Create boxplots
        melted_df = pd.melt(pd.concat([viz_df[['risk_group']], numeric_featur
                            id_vars=['risk_group'], value_vars=top_features,
                            var_name='Feature', value_name='Value')

        # Standardize values for better visualization
        feature_means = melted_df.groupby('Feature')['Value'].transform('mean
        feature_stds = melted_df.groupby('Feature')['Value'].transform('std')
        melted_df['Standardized_Value'] = (melted_df['Value'] - feature_means

        # Create boxplots
```

```
        sns.boxplot(x='Feature', y='Standardized_Value', hue='risk_group',
                    data=melted_df, palette=['green', 'red'])

        plt.title("Feature Distribution by Risk Group")
        plt.xlabel("Feature")
        plt.ylabel("Standardized Value")
        plt.legend(title="Risk Group", labels=["Low Risk", "High Risk"])
        plt.xticks(rotation=45)
        plt.tight_layout()
        plt.savefig(os.path.join(output_dir, "feature_importance.png"))
        plt.close()
        print("Created feature importance visualization")
except Exception as e:
    print(f"Error creating feature importance visualization: {e}")

# 6. Transcript sentiment analysis
try:
    sentiment_data = []
    for r in results:
        if 'transcript' in r and not r['transcript'].startswith("No transcrip
            if 'sentiment_polarity' in r['features'] and 'vader_compound' in
                sentiment_data.append({
                    'filename': r['filename'],
                    'textblob_sentiment': r['features']['sentiment_polarity']
                    'vader_sentiment': r['features']['vader_compound'],
                    'risk_score': r.get('risk_score', 0)
                })

    if sentiment_data:
        sent_df = pd.DataFrame(sentiment_data)

        plt.figure(figsize=(12, 6))

        plt.subplot(1, 2, 1)
        sns.scatterplot(x='textblob_sentiment', y='vader_sentiment',
                        hue='risk_score', size='risk_score',
                        data=sent_df, palette='viridis', sizes=(50, 200))
        plt.title("Sentiment Analysis Comparison")
        plt.xlabel("TextBlob Sentiment")
        plt.ylabel("VADER Sentiment")
        plt.grid(True, linestyle='--', alpha=0.5)

        plt.subplot(1, 2, 2)
        sent_df_melted = pd.melt(sent_df,
                                 id_vars=['filename', 'risk_score'],
                                 value_vars=['textblob_sentiment', 'vader_sent
                                 var_name='Sentiment Type', value_name='Sentim

        sns.barplot(x='filename', y='Sentiment Value', hue='Sentiment Type',
                    data=sent_df_melted)
        plt.title("Sentiment by Sample")
        plt.xticks(rotation=45)
        plt.tight_layout()

        plt.savefig(os.path.join(output_dir, "sentiment_analysis.png"))
        plt.close()
        print("Created sentiment analysis visualization")
except Exception as e:
    print(f"Error creating sentiment visualization: {e}")

# 7. Audio feature radar chart
try:
    if len(results) > 0:
        # Select key audio features
        audio_features = ['zcr', 'rmse', 'spectral_centroid',
                          'spectral_bandwidth', 'rolloff', 'mfcc_mean_1']

        # Create radar chart for high and low risk samples
        high_risk = [r for r in results if r.get('risk_score', 0) > 0.5]
        low_risk = [r for r in results if r.get('risk_score', 0) <= 0.5]

        if high_risk and low_risk and all(f in numeric_features.columns for f
            # Normalize features
            normalized = (numeric_features[audio_features] - numeric_features
                          (numeric_features[audio_features].max() - numeric_feat

            # Calculate averages for high and low risk groups
            high_risk_indices = [results.index(r) for r in high_risk]
            low_risk_indices = [results.index(r) for r in low_risk]

            high_risk_avg = normalized.iloc[high_risk_indices].mean()
            low_risk_avg = normalized.iloc[low_risk_indices].mean()

            # Create radar chart
            angles = np.linspace(0, 2*np.pi, len(audio_features), endpoint=Fa
            angles += angles[:1]  # Close the loop
```

```
                high_risk_values = high_risk_avg.values.tolist()
                high_risk_values += high_risk_values[:1]

                low_risk_values = low_risk_avg.values.tolist()
                low_risk_values += low_risk_values[:1]

                feature_labels = audio_features.copy()
                feature_labels += feature_labels[:1]

                plt.figure(figsize=(10, 10))
                ax = plt.subplot(111, polar=True)

                # Plot high risk group
                ax.plot(angles, high_risk_values, 'r-', linewidth=2, label='High
                ax.fill(angles, high_risk_values, 'r', alpha=0.25)

                # Plot low risk group
                ax.plot(angles, low_risk_values, 'g-', linewidth=2, label='Low Ri
                ax.fill(angles, low_risk_values, 'g', alpha=0.25)

                # Set labels
                ax.set_thetagrids(np.degrees(angles[:-1]), feature_labels[:-1])
                ax.set_rlabel_position(0)
                plt.xticks(angles[:-1], feature_labels[:-1])

                plt.title("Audio Features Comparison: High vs Low Risk")
                plt.legend(loc='upper right')
                plt.tight_layout()
                plt.savefig(os.path.join(output_dir, "audio_radar_chart.png"))
                plt.close()
                print("Created audio feature radar chart")
    except Exception as e:
        print(f"Error creating radar chart: {e}")

    # 8. Timeline visualization of features
    try:
        # This assumes filenames have some date/time information
        # If not, we'll just use the order they appear in the dataset
        time_df = pd.DataFrame({'filename': [r['filename'] for r in results]})
        for col in ['speech_rate', 'lexical_diversity', 'pause_ratio', 'hesitatio
            if col in numeric_features.columns:
                time_df[col] = numeric_features[col].values

        if not time_df.empty and len(time_df) >= 2:
            plt.figure(figsize=(14, 8))
            time_df_melted = pd.melt(time_df, id_vars=['filename'],
                                     value_name='Value', var_name='Feature')

            sns.lineplot(x='filename', y='Value', hue='Feature', data=time_df_mel
            plt.title("Speech Metrics Over Samples")
            plt.xticks(rotation=45)
            plt.grid(True, linestyle='--', alpha=0.5)
            plt.tight_layout()
            plt.savefig(os.path.join(output_dir, "speech_timeline.png"))
            plt.close()
            print("Created speech timeline visualization")
    except Exception as e:
        print(f"Error creating timeline visualization: {e}")

    print(f"All visualizations saved to {output_dir}")


# Update main execution code to include the new visualizations
def enhanced_run_batch_analysis(audio_folder_path):
    # Run the original batch analysis
    results = run_batch_analysis(audio_folder_path)

    # If results exist, create the enhanced visualizations
    if results:
        # Combine features from results
        try:
            features_df = pd.DataFrame([r['features'] for r in results])

            # Create visualization output directory
            viz_dir = os.path.join(audio_folder_path, "visualizations")
            os.makedirs(viz_dir, exist_ok=True)

            # Generate enhanced visualizations
            generate_visualizations(features_df, results, viz_dir)

            print(f"\nEnhanced visualizations generated and saved to {viz_dir}")
        except Exception as e:
            print(f"Error generating enhanced visualizations: {e}")

    return results
```

```
    return results

# Replace the original batch analysis in the main execution
if __name__ == "__main__":
    # Set folder path
    default_folder_path = "/content/audio_folder"
    folder_path = input(f"Enter audio folder path (default: {default_folder_path}

    # Ensure folder exists
    if not os.path.exists(folder_path):
        print(f"Creating folder: {folder_path}")
        try:
            os.makedirs(folder_path)
        except Exception as e:
            print(f"Error creating folder: {e}")
            sys.exit(1)

    # Check for audio files
    audio_files = [f for f in os.listdir(folder_path)
                   if f.endswith((".wav", ".mp3")) and os.path.isfile(os.path.join

    if not audio_files:
        print(f"No audio files found in {folder_path}. Please add .wav or .mp3 fi
        sys.exit(0)

    # Run analysis
    print(f"\nProcessing {len(audio_files)} audio files...")
    final_results = enhanced_run_batch_analysis(folder_path)

    # Display summary
    print("\n--- Summary ---")
    if final_results:
        for res in sorted(final_results, key=lambda x: x.get('risk_score', 0), re
            anomaly = "Yes" if res.get('iso_label', 0) == -1 or res.get('dbscan_l
            print(f"\nFile: {res['filename']}")
            print(f"Transcript: {res['transcript'][:100]}{'...' if len(res['trans
            print(f"Risk Score: {res.get('risk_score', 0)}")
            print(f"Anomaly: {anomaly}")

        print("\nAnalysis complete. Results saved to 'voice_analysis_report.md'")
        print("Enhanced visualizations saved to the 'visualizations' folder")
    else:
        print("No results to display.")
```

```
Enter audio folder path (default: /content/audio_folder): /content/audio2_folder

Processing 14 audio files...

Processing: 260-123288-0000.mp3
Transcript: the roarings become lost in the distance

Processing: 237-134493-0015.mp3
Transcript: there was something individual about the Great Farm a most unusual tremendous and ca

Processing: 237-134500-0006.mp3
Transcript: just smell the Wild Roses they are always so spicy after a rain

Processing: 260-123288-0003.mp3
Transcript: the electric light can scarcely penetrate through the dense curtain which is dropped

Processing: 260-123288-0002.mp3
Transcript: the atmosphere is charged with vapors pervaded with the electricity generated by the

Processing: 237-134500-0002.mp3
Transcript: a Brisk wind had come up and was driving puffy white clouds across the sky

Processing: 237-134493-0014.mp3
Transcript: they think you're proud because you've been away to school or something

Processing: 237-134500-0004.mp3
Transcript: that invitation decided her

Processing: 237-134500-0005.mp3
Transcript: oh but I'm glad to get this place mold

Processing: 237-134500-0007.mp3
Transcript: we never had so many of them in here before

Processing: 237-134500-0003.mp3
Transcript: the orchard was sparkling and Rippling in the Sun

Processing: 260-123288-0001.mp3
Transcript: the weather  if we may use the term we'll change before long

Processing: 237-134500-0010.mp3
Transcript: it's exciting to see everything growing so fast and to get the grass cut

Processing: 237-134500-0011.mp3
Transcript: aren't you splashed look at the spiderwebs all over the grass

File: 260-123288-0000.mp3
K-Means Cluster: 1
DBSCAN Label: -1
Isolation Forest Label: 1
Similarity Score: 0.50
Risk Score: 1.0

File: 237-134493-0015.mp3
K-Means Cluster: 1
DBSCAN Label: -1
Isolation Forest Label: 1
Similarity Score: 0.08
Risk Score: 1.0

File: 237-134500-0006.mp3
K-Means Cluster: 1
DBSCAN Label: -1
Isolation Forest Label: 1
Similarity Score: 0.18
Risk Score: 1.0

File: 260-123288-0003.mp3
K-Means Cluster: 0
DBSCAN Label: -1
Isolation Forest Label: 1
Similarity Score: 0.24
Risk Score: 1.0

File: 260-123288-0002.mp3
K-Means Cluster: 0
DBSCAN Label: -1
Isolation Forest Label: 1
Similarity Score: 0.10
Risk Score: 1.0

File: 237-134500-0002.mp3
K-Means Cluster: 0
DBSCAN Label: -1
Isolation Forest Label: 1
Similarity Score: -0.76
Risk Score: 1.0

File: 237-134493-0014.mp3
K-Means Cluster: 1
DBSCAN Label: -1
Isolation Forest Label: 1
Similarity Score: 0.27
Risk Score: 1.0
```