Certainly! I'll go through the code step by step and explain each part in detail.

1. Importing Libraries:

```
import os
import librosa
import numpy as np
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
import matplotlib.pyplot as plt
from datasets import load_dataset
```

-> **os:** Provides a way to interact with the operating system (e.g., file operations).
-> **librosa**: A library for audio and music analysis.
-> **numpy:** A library for numerical computations.
-> **RandomForestClassifier:** A machine learning algorithm for classification tasks.
-> **pandas:** A library for data manipulation and analysis.
-> **matplotlib.pyplot:** A library for creating visualizations.
-> **load_dataset:** A function from the Hugging Face `datasets` module used for loading datasets directly from the Hugging Face model hub.

2. Function to Extract Audio Features:

```
def extract_audio_features(audio_file):
    y, sr = librosa.load(audio_file)
    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
    return np.mean(mfccs, axis=1)
```

-> This function (extract_audio_features) takes an **audio_file** as input.
-> **librosa.load** is used to load the audio file and obtain the audio waveform (y) and the sample rate (sr).
-> **librosa.feature.mfcc** computes the Mel-frequency cepstral coefficients (MFCCs), which are features commonly used in audio processing.
-> **np.mean(mfccs, axis=1)** computes the mean of the MFCCs along the time axis, resulting in a feature vector.

3. Function to Train Emotion Detection Model:

```
def train_emotion_detection_model(features, labels):
    clf = RandomForestClassifier(n_estimators=100, random_state=42)
    clf.fit(features, labels)
    return clf
```

-> This function (train_emotion_detection_model) takes **features** (audio paths) and corresponding **labels** (emotions) as input.
-> A **RandomForestClassifier** is created and trained on the features (audio paths) and labels (emotions).
-> The trained classifier (**clf**) is returned.

4. Function to Predict Emotion:

```
def predict_emotion(model, audio_file):
    features = extract_audio_features(audio_file)
    emotion_label = model.predict([features])[0]
    return "happy" if emotion_label == 1 else "sad"
```

-> This function (predict_emotion) takes the trained **model** and an **audio_file**.
-> It first extracts audio features from the file.
-> The model predicts the emotion label using these features.
-> The function returns **"happy"** if the label is 1 (assuming 1 represents "happy"), otherwise it returns **"sad"**.

5. Function to Load Common Voice Dataset:

```
def load_common_voice_dataset(language_code, split='train'):
    common_voice_dataset = load_dataset('common_voice', language_code,
split=split)
    return common_voice_dataset
```

-> This function (load_common_voice_dataset) loads the Common Voice dataset for a specified **language_code** and split (default is **train**) using the **load_dataset** function from Hugging Face.

6. Function to Print Results:

```
def print_results(dataset, emotion_detection_model):
    emotion_predictions = []

    for item in dataset:
        audio_file = item['path']
        emotion = predict_emotion(emotion_detection_model, audio_file)
        emotion_predictions.append((audio_file, emotion))

    # Print predicted emotions
    print("Emotion Predictions:")
    print(pd.DataFrame(emotion_predictions, columns=['Audio File',
'Emotion']))
```

-> This function (print_results) takes the loaded **dataset** and the trained **emotion_detection_model.**
-> It iterates over the items in the dataset (which include audio file paths) and predicts the emotion for each.
-> The predictions are stored in the **emotion_predictions** list, which is then printed as a DataFrame.

7. Predicted Talk Speed Table and Generating Line Chart:

```
# Predict talk speed
talk_speeds = calculate_talk_speed([entry[0] for entry in
emotion_predictions])
talk_speed_df = pd.DataFrame({
    'Audio File': [entry[0] for entry in emotion_predictions],
    'Talk Speed (s)': talk_speeds
})

# Print predicted talk speed as a table
print("\nPredicted Talk Speed:")
print(talk_speed_df)

# Generate the line chart
timestamps = range(len(emotion_predictions))
emotions = [entry[1] for entry in emotion_predictions]

emotion_values = [1 if emotion == "happy" else -1 for emotion in
emotions]
```

```
plt.figure(figsize=(10, 6))
plt.plot(timestamps, emotion_values, marker='o', linestyle='-')
plt.xlabel("Time")
plt.ylabel("Emotion")
plt.title("Predicted Emotions Over Time")
plt.yticks([-1, 1], ["sad", "happy"])
plt.show()
```

-> This part of the code calculates the talk speed (which seems to be handled by another function **calculate_talk_speed** not shown in this code snippet). It then creates a DataFrame **talk_speed_df** and prints it as a table. Finally, it generates a line chart showing the predicted emotions over time.

8. Setting Language Code and Loading Common Voice Dataset:

```
language_code = 'cy'
common_voice_train = load_common_voice_dataset(language_code,
split='train')
audio_paths = common_voice_train['path']
```

-> **language_code** is set to **cy** for Welsh, but you can change this to any supported language code.
-> The Common Voice dataset for the specified language and split is loaded, and audio paths are obtained.

9. Training Emotion Detection Model:

```
binary_labels = [1 if "happy" in label else 0 for label in labels]
emotion_detection_model = train_emotion_detection_model(audio_paths,
binary_labels)
```

-> In this example, binary labels are generated for illustration purposes. In a real scenario, you would have actual labels for emotion detection.

10. Printing Results:

```
print_results(common_voice_train, emotion_detection_model)
```

- This line calls the **print_results** function with the loaded dataset and trained emotion detection model.