```python
import cv2
import scipy
import subprocess
import pandas as pd
import numpy as np
from scipy.io import wavfile
from glob import glob as globlin
import matplotlib.pyplot as plt
import progressbar
from imblearn.over_sampling import SMOTE


def number_of_real_speakers(main_path):
    folder_paths = globlin(main_path)
    sum_files = 0
    for path in folder_paths:
        sum_files += len(globlin(path + '/*'))
    return sum_files


def convert_files_to_wav(mp3_paths, dest_path):
    counter = 0
    for i in range(len(mp3_paths)):
        if 'en-US-Wavenet-C' in mp3_paths.iloc[i]:
            counter += 1
            mp3_file_name = mp3_paths.iloc[i].replace('{DS_PATH}', '/Users/ahmadchaiban/Desktop/Guelph/655
            wav_name = dest_path + '/' + mp3_file_name.split('/')[-1].split('.')[0] + '_set_C.wav'
            subprocess.run(f'ffmpeg -i {mp3_file_name} {wav_name}', shell=True, check=True)
            if counter == 13000:
                break


def extract_spectrogram_from_audio(main_path, destination_path):
    paths = globlin(main_path)
    with progressbar.ProgressBar(max_value=len(paths)) as bar:
        for index, path in enumerate(paths):
            output_path = destination_path + path.split('/')[-1].replace('.wav','.jpg')

            FS, data = wavfile.read(path)  # read wav file

            fig,ax = plt.subplots(1)
            fig.subplots_adjust(left=0,right=1,bottom=0,top=1)
            ax.specgram(data, Fs=FS, NFFT=128, noverlap=0)  # plot
            ax.axis('tight')
            ax.axis('off')

            plt.savefig(output_path)
            plt.close()
            bar.update(index)


def load_images_as_dataframe(main_path, category_binary_value):
    paths = globlin(main_path)
    main_array = []
    classes = []
    with progressbar.ProgressBar(max_value=len(paths)) as bar:
        for index, path in enumerate(paths):
            image = cv2.resize(cv2.imread(path), (75, 75))
            im_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            main_array.append(im_rgb)
            classes.append(category_binary_value)

            bar.update(index)
    return np.array(main_array), np.array(classes)


def adjust_classes(classes_dataset):
    new_classes = []
    for value in classes_dataset:
        if value == 1:
            new_classes.append(np.array([value, 0]))
        else:
            new_classes.append(np.array([value, 1]))
    return np.array(new_classes)


def oversample_save(real_images, real_classes, synth_images, synth_classes):
    """
    Oversamples on a subset of real images, then recombines the image data before saving.
    """
    # Data to oversample and their corresponding classes
    images_to_oversample = np.concatenate((synth_images, real_images[0:100]), axis=0)
    corr_classes_dataset = np.concatenate((synth_classes, real_classes[0:100]), axis=0)

    oversampler = SMOTE()
    X_over, y_over = oversampler.fit_resample(images_to_oversample, corr_classes_dataset)

    # Adding rest of real image data
    X_over = np.concatenate((X_over, real_images[100:3000]), axis=0)
    y_over = np.concatenate((y_over, real_classes[100:3000]), axis=0)

    # Saving Data as pickle files
    with open('oversampled_image_features.pickle', 'wb') as f:
        np.save(f, X_over)
    with open('oversampled_classes.pickle', 'wb') as f:
        np.save(f, y_over)


def oversample_load(image_references, class_references):
    """
    Comparison of elements for the demo, but in reality will
    be comparing keys in a dict
    """
    with open('oversampled_image_features.pickle', 'rb') as f:
        X_over = np.load(f)
    with open('oversampled_classes.pickle', 'rb') as f:
        y_over = np.load(f)
    original_images = []
    original_classes = []
    for index, possible_real in enumerate(X_over):
        if possible_real in image_references:
            original_images.append(possible_real)
        if y_over[index] in class_references:
            original_classes.append(y_over[index])
    return np.array(original_images), np.array(original_classes)
```