ENGINNERING FACULTY - COMPUTER ENGINEERING DEPARTMENT

Biomedical Signal Analysis and Machine Learning
2023-2024

STUDENT NAME	: Semih Utku POLAT
STUDENT ID	: 191805060
TUDENT NAME	: Burak Tüzel
STUDENT ID	: 191805057

TABLE OF CONTENTS

1 Contents

2 Dataset Description ... 1
3 Audio Recording ... 2
3.1 Record Audio Files For Model Training ... 2
3.2 Record Audio Files For Model Testing With Background Noise ... 2
3.3 Record Audio Files For Model Testing Without Background Noise. ... 3
4 Audio Labeling ... 3
5 Audio Model Training ... 5
6 Audio Model Testing ... 10

INTRODUCTION

Voice or speaker recognition is the ability of a machine or program to receive and interpret dictation or to understand and perform spoken commands. Voice recognition can be defined as a technology that allows a device to understand and analyse a human voice and then transcribe each of the dictated words into usable text. Specifically, the voice is captured via the device's microphone in sound frequencies and then transcribed into written text. Voice recognition can be seen as an alternative to keyboard/handwritten entry and is often praised for being faster and saving time in everyday tasks.

2 Dataset Description

The dataset comprises a collection of sound recordings generated through a scripted series of events. Each recording is designed to capture various audio cues, including specific verbal instructions and non-verbal actions. The events within each recording are as follows:

Wait: A period of silence to capture ambient background noise.
Read "computer": The participant reads the word "computer" aloud.
Wait: Another silent interval.
Read "engineering": The participant reads the word "engineering" aloud.
Wait: Another silent interval.
Say your name: The participant verbally states their first name.
Wait: Another silent interval.
Say your last name: The participant verbally states their last name.
Wait: Another silent interval.
Cough once: The participant coughs audibly.
Wait: Another silent interval.
Clap your hands once: The participant claps their hands.
Wait: Another silent interval.
Snap your fingers once: The participant snaps their fingers.
Wait: Another silent interval.

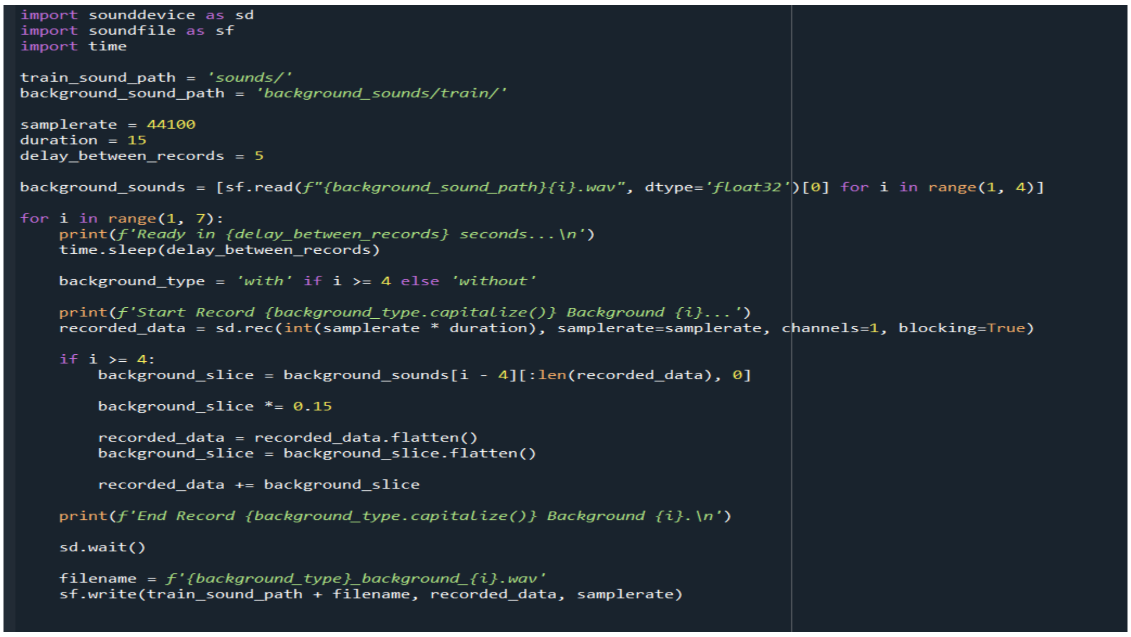
This sequence of events is repeated three times to provide variability and ensure robustness in the dataset. Additionally, each recording is created in two variations: with background sound and without background sound. In this project, the dataset accounts for multiple members, generating a total of 12 recordings for our group (6 with background sound and 6 without background sound).

3 Audio Recording

This project has three different scripts used to generate audio files for training the model, audio files with background noise for testing the model with background noise, and audio files without background noise for testing the model without background noise.

3.1 Record Audio Files For Model Training

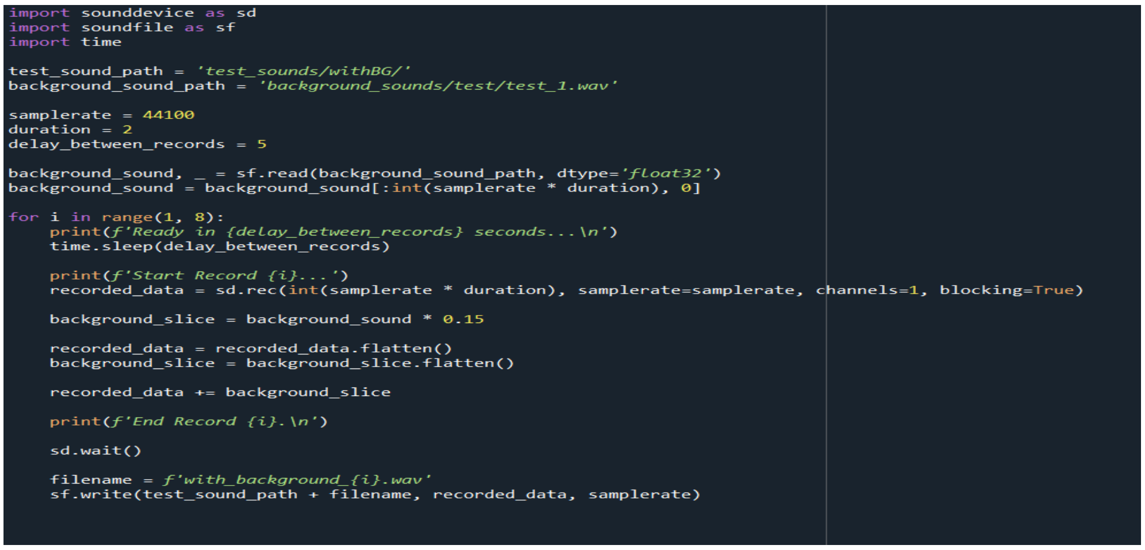
This script used to record 15 sec long audio files for training the model with and without background noise.



```python
import sounddevice as sd
import soundfile as sf
import time

train_sound_path = 'sounds/'
background_sound_path = 'background_sounds/train/'

samplerate = 44100
duration = 15
delay_between_records = 5

background_sounds = [sf.read(f"{background_sound_path}{i}.wav", dtype='float32')[0] for i in range(1, 4)]

for i in range(1, 7):
    print(f'Ready in {delay_between_records} seconds...\n')
    time.sleep(delay_between_records)

    background_type = 'with' if i >= 4 else 'without'

    print(f'Start Record {background_type.capitalize()} Background {i}...')
    recorded_data = sd.rec(int(samplerate * duration), samplerate=samplerate, channels=1, blocking=True)

    if i >= 4:
        background_slice = background_sounds[i - 4][:len(recorded_data), 0]

        background_slice *= 0.15

        recorded_data = recorded_data.flatten()
        background_slice = background_slice.flatten()

        recorded_data += background_slice

    print(f'End Record {background_type.capitalize()} Background {i}.\n')

    sd.wait()

    filename = f'{background_type}_background_{i}.wav'
    sf.write(train_sound_path + filename, recorded_data, samplerate)
```

3.2 Record Audio Files For Model Testing With Background Noise

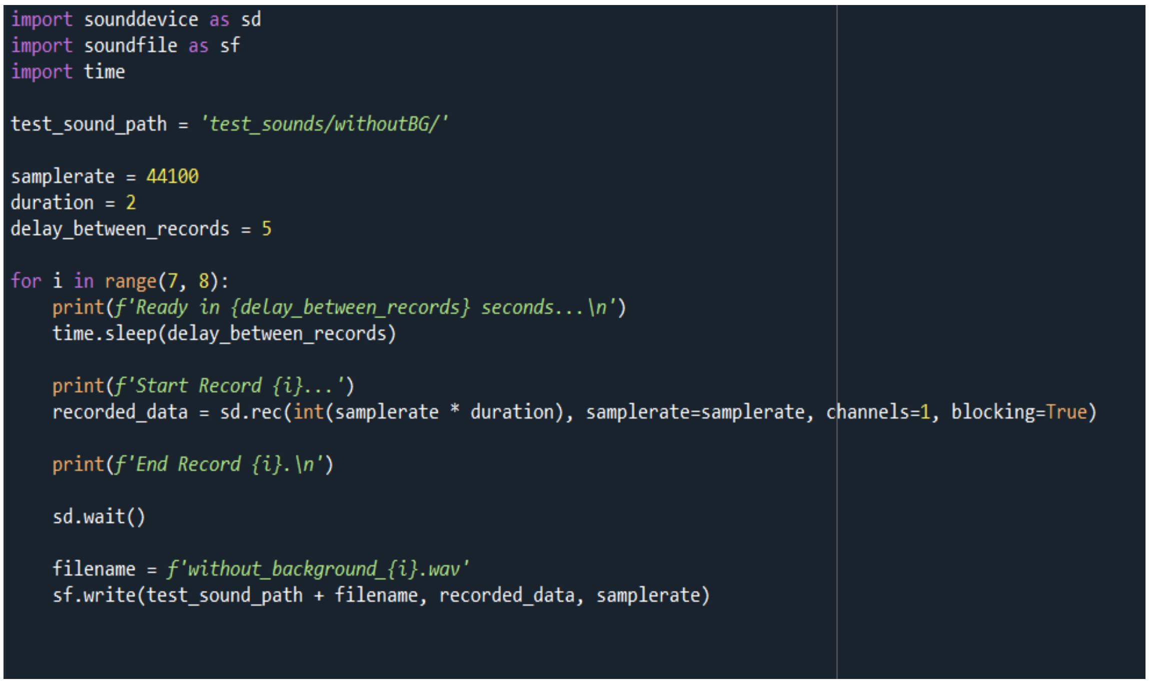
This script used to record 2 sec long audio files for testing the model with background noise.



```python
import sounddevice as sd
import soundfile as sf
import time

test_sound_path = 'test_sounds/withBG/'
background_sound_path = 'background_sounds/test/test_1.wav'

samplerate = 44100
duration = 2
delay_between_records = 5

background_sound, _ = sf.read(background_sound_path, dtype='float32')
background_sound = background_sound[:int(samplerate * duration), 0]

for i in range(1, 8):
    print(f'Ready in {delay_between_records} seconds...\n')
    time.sleep(delay_between_records)

    print(f'Start Record {i}...')
    recorded_data = sd.rec(int(samplerate * duration), samplerate=samplerate, channels=1, blocking=True)

    background_slice = background_sound * 0.15

    recorded_data = recorded_data.flatten()
    background_slice = background_slice.flatten()

    recorded_data += background_slice

    print(f'End Record {i}.\n')

    sd.wait()

    filename = f'with_background_{i}.wav'
    sf.write(test_sound_path + filename, recorded_data, samplerate)
```

3.3 Record Audio Files For Model Testing Without Background Noise.

This script used to record 2 sec long audio files for testing the model without background noise.

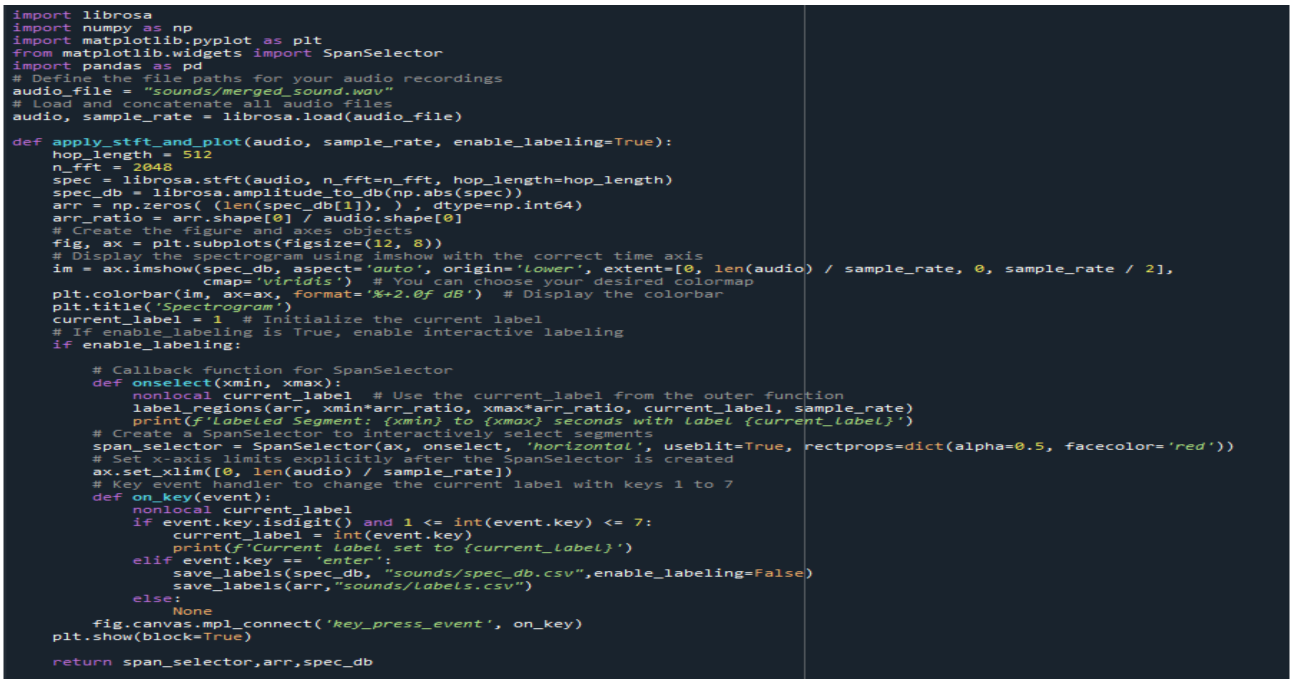


```python
import sounddevice as sd
import soundfile as sf
import time

test_sound_path = 'test_sounds/withoutBG/'

samplerate = 44100
duration = 2
delay_between_records = 5

for i in range(7, 8):
    print(f'Ready in {delay_between_records} seconds...\n')
    time.sleep(delay_between_records)

    print(f'Start Record {i}...')
    recorded_data = sd.rec(int(samplerate * duration), samplerate=samplerate, channels=1, blocking=True)

    print(f'End Record {i}.\n')

    sd.wait()

    filename = f'without_background_{i}.wav'
    sf.write(test_sound_path + filename, recorded_data, samplerate)
```

4 Audio Labeling

In this part, sound_signal_labeling.py script has written to analyze an audio file, specifically applying Short-Time Fourier Transform (STFT) to generate a spectrogram. The script also allows interactive labeling of specific segments of the audio.

Importing necessary libraries and loading the audio file with its sample rate. apply_stft_and_plot function applies STFT to audio file to generate a spectrogram then plots it. The on_select function allows the user to label a spectrogram graph based on the range they choose for an audio file. Finally saves the spectrogram in decibels and the labels array into the sounds folder when the user pressed “Return” key

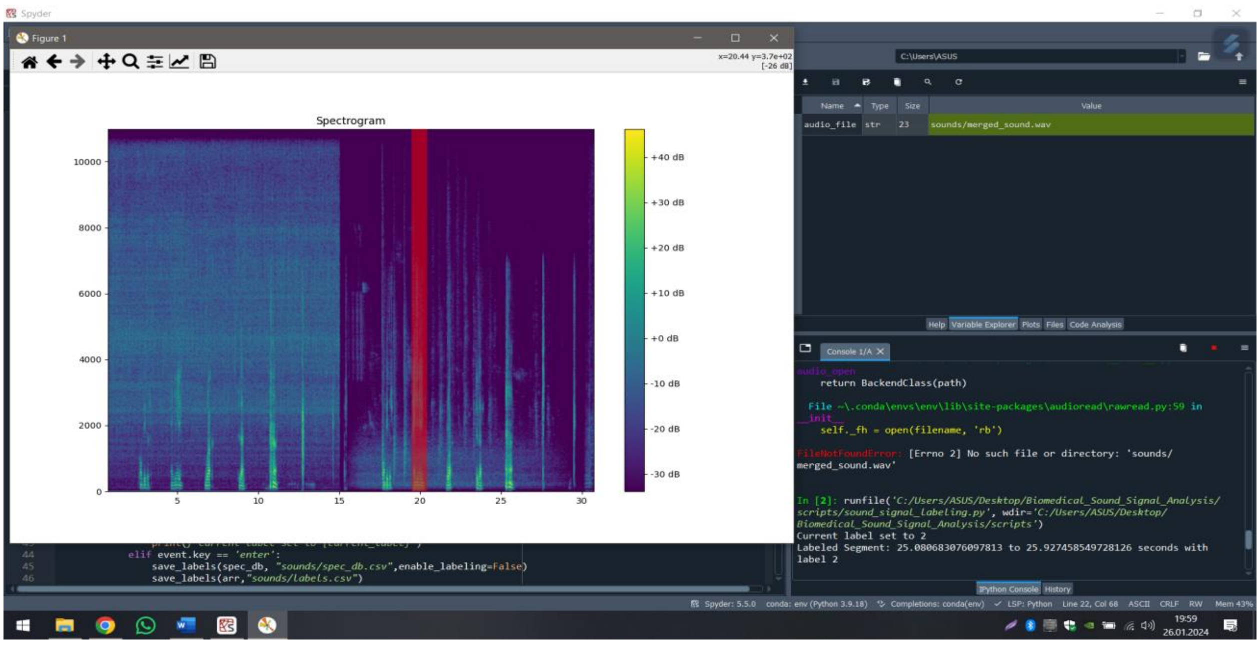


```python
import librosa
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import SpanSelector
import pandas as pd
# Define the file paths for your audio recordings
audio_file = "sounds/merged_sound.wav"
# Load and concatenate all audio files
audio, sample_rate = librosa.load(audio_file)

def apply_stft_and_plot(audio, sample_rate, enable_labeling=True):
    hop_length = 512
    n_fft = 2048
    spec = librosa.stft(audio, n_fft=n_fft, hop_length=hop_length)
    spec_db = librosa.amplitude_to_db(np.abs(spec))
    arr = np.zeros( (len(spec_db[1]), ) , dtype=np.int64)
    arr_ratio = arr.shape[0] / audio.shape[0]
    # Create the figure and axes objects
    fig, ax = plt.subplots(figsize=(12, 8))
    # Display the spectrogram using imshow with the correct time axis
    im = ax.imshow(spec_db, aspect='auto', origin='lower', extent=[0, len(audio) / sample_rate, 0, sample_rate / 2],
                   cmap='viridis')  # You can choose your desired colormap
    plt.colorbar(im, ax=ax, format='%+2.0f dB')  # Display the colorbar
    plt.title('Spectrogram')
    current_label = 1  # Initialize the current label
    # If enable_labeling is True, enable interactive labeling
    if enable_labeling:

        # Callback function for SpanSelector
        def onselect(xmin, xmax):
            nonlocal current_label  # Use the current_label from the outer function
            label_regions(arr, xmin*arr_ratio, xmax*arr_ratio, current_label, sample_rate)
            print(f'Labeled Segment: {xmin} to {xmax} seconds with label {current_label}')
        # Create a SpanSelector to interactively select segments
        span_selector = SpanSelector(ax, onselect, 'horizontal', useblit=True, rectprops=dict(alpha=0.5, facecolor='red'))
        # Set x-axis limits explicitly after the SpanSelector is created
        ax.set_xlim([0, len(audio) / sample_rate])
        # Key event handler to change the current label with keys 1 to 7
        def on_key(event):
            nonlocal current_label
            if event.key.isdigit() and 1 <= int(event.key) <= 7:
                current_label = int(event.key)
                print(f'Current label set to {current_label}')
            elif event.key == 'enter':
                save_labels(spec_db, "sounds/spec_db.csv",enable_labeling=False)
                save_labels(arr,"sounds/labels.csv")
            else:
                None
        fig.canvas.mpl_connect('key_press_event', on_key)
    plt.show(block=True)

    return span_selector,arr,spec_db
```

This is the function that is used in on_select function to label on the spectrogram graph.

```python
# Function to label regions based on criteria
def label_regions(arr, start_time, end_time, label, sample_rate):
    start_idx = int(start_time * sample_rate)
    end_idx = int(end_time * sample_rate)
    arr[start_idx:end_idx] = label
```

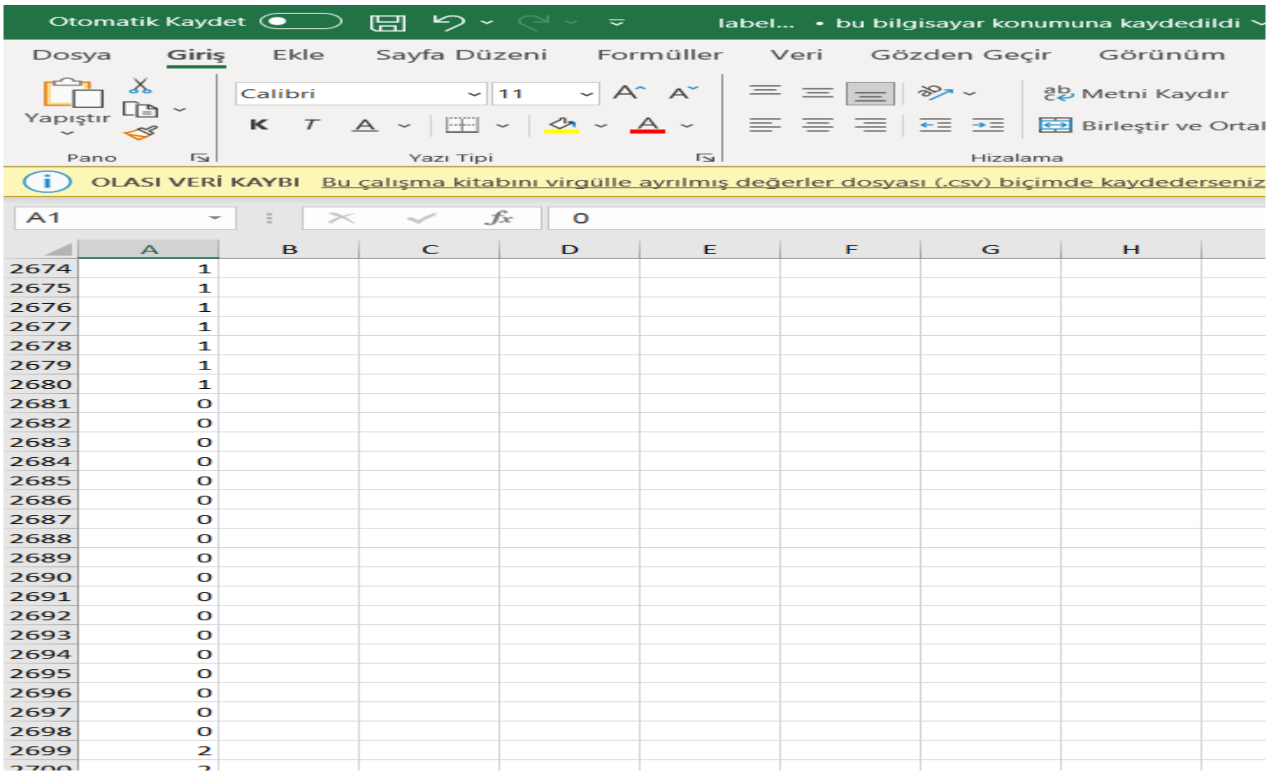
The example of executing:



```
        elif event.key == 'enter':
            save_labels(spec_db, "sounds/spec_db.csv",enable_labeling=False)
            save_labels(arr,"sounds/labels.csv")
```

```
audio_open
    return BackendClass(path)

  File ~\.conda\envs\env\lib\site-packages\audioread\rawread.py:59 in
__init__
    self._fh = open(filename, 'rb')

FileNotFoundError: [Errno 2] No such file or directory: 'sounds/
merged_sound.wav'


In [2]: runfile('C:/Users/ASUS/Desktop/Biomedical_Sound_Signal_Analysis/
scripts/sound_signal_labeling.py', wdir='C:/Users/ASUS/Desktop/
Biomedical_Sound_Signal_Analysis/scripts')
Current label set to 2
Labeled Segment: 25.080683076097813 to 25.927458549728126 seconds with
label 2
```

The label file when pressed “Return” key:

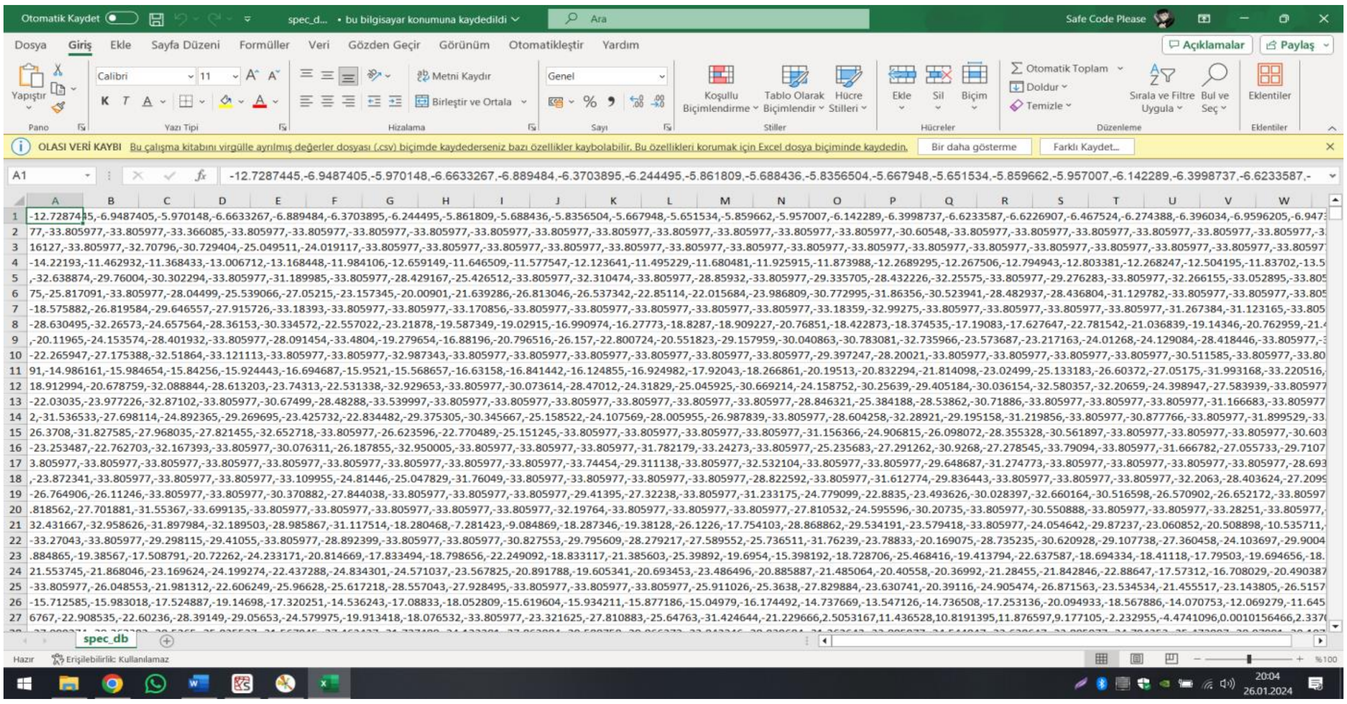


OLASI VERİ KAYBI Bu çalışma kitabını virgülle ayrılmış değerler dosyası (.csv) biçimde kaydederseniz

A1: 0

	A	B	C	D	E	F	G	H
2674	1							
2675	1							
2676	1							
2677	1							
2678	1							
2679	1							
2680	1							
2681	0							
2682	0							
2683	0							
2684	0							
2685	0							
2686	0							
2687	0							
2688	0							
2689	0							
2690	0							
2691	0							
2692	0							
2693	0							
2694	0							
2695	0							
2696	0							
2697	0							
2698	0							
2699	2							

The spec_db file when pressed “Return” key:



OLASI VERİ KAYBI Bu çalışma kitabını virgülle ayrılmış değerler dosyası (.csv) biçimde kaydederseniz bazı özellikler kaybolabilir. Bu özellikleri korumak için Excel dosya biçiminde kaydedin.

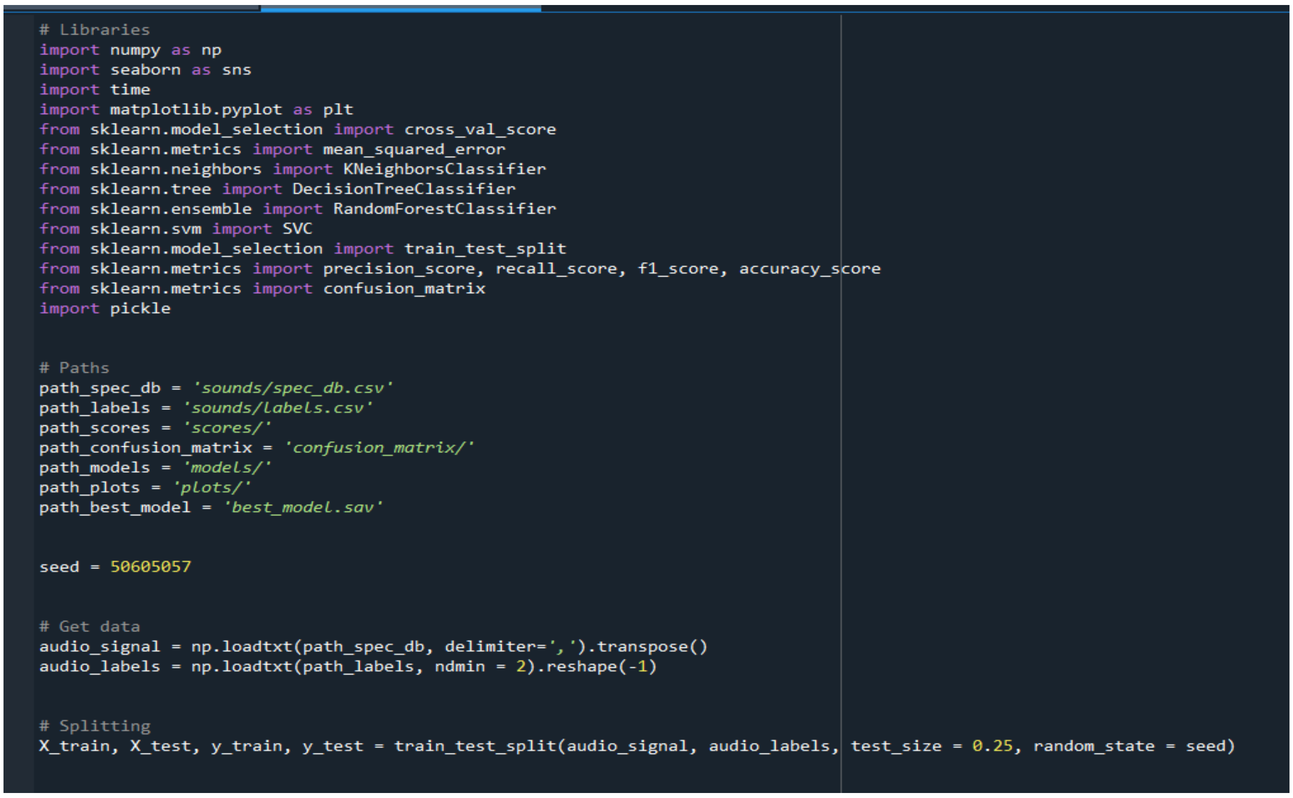
A1 -12.7287445,-6.9487405,-5.970148,-6.6633267,-6.889484,-6.3703895,-6.244495,-5.861809,-5.688436,-5.8356504,-5.667948,-5.651534,-5.859662,-5.957007,-6.142289,-6.3998737,-6.6233587,-

	A
1	-12.7287445,-6.9487405,-5.970148,-6.6633267,-6.889484,-6.3703895,-6.244495,-5.861809,-5.688436,-5.8356504,-5.667948,-5.651534,-5.859662,-5.957007,-6.142289,-6.3998737,-6.6233587,-6.6226907,-6.467524,-6.274388,-6.396034,-6.9596205,-6.947
2	77,-33.805977,-33.805977,-33.366085,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-30.60548,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-3
3	16127,-33.805977,-32.70796,-30.729404,-25.049511,-24.019117,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.80597
4	-14.22193,-11.462932,-11.368433,-13.006712,-13.168448,-11.984106,-12.659149,-11.646509,-11.577547,-12.123641,-11.495229,-11.680481,-11.925915,-11.873988,-12.2689295,-12.267506,-12.794943,-12.803381,-12.268247,-12.504195,-11.83702,-13.5
5	,-32.638874,-29.76004,-30.302294,-33.805977,-31.189985,-33.805977,-28.429167,-25.426512,-33.805977,-32.310474,-33.805977,-28.85932,-33.805977,-29.335705,-28.432226,-32.25575,-33.805977,-29.276283,-33.805977,-32.266155,-33.052895,-33.805
6	75,-25.817091,-33.805977,-28.04499,-25.539066,-27.05215,-23.157345,-20.00901,-21.639286,-26.813046,-26.537342,-22.85114,-22.015684,-23.986809,-30.772995,-31.86356,-30.523941,-28.482937,-28.436804,-31.129782,-33.805977,-33.805977,-33.805
7	-18.575882,-26.819584,-29.646557,-27.915726,-33.18393,-33.805977,-33.805977,-33.170856,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.18359,-32.99275,-33.805977,-33.805977,-33.805977,-33.805977,-31.267384,-31.123165,-33.805
8	-28.630495,-32.26573,-24.657564,-28.36153,-30.334572,-22.557022,-23.21878,-19.587349,-19.02915,-16.990974,-16.27773,-18.8287,-18.909227,-20.76851,-18.422873,-18.374535,-17.19083,-17.627647,-22.781542,-21.036839,-19.14346,-20.762959,-21.4
9	,-20.11965,-24.153574,-28.401932,-33.805977,-28.091454,-33.4804,-19.279654,-16.88196,-20.796516,-26.157,-22.800724,-20.551823,-29.157959,-30.040863,-30.783081,-32.735966,-23.573687,-23.217163,-24.01268,-24.129084,-28.418446,-33.805977,-3
10	-22.265947,-27.175388,-32.51864,-33.121113,-33.805977,-33.805977,-32.987343,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-29.397247,-28.20021,-33.805977,-33.805977,-33.805977,-33.805977,-30.511585,-33.805977,-33.80
11	91,-14.986161,-15.984654,-15.84256,-15.924443,-16.694687,-15.9521,-15.568657,-16.63158,-16.841442,-16.124855,-16.924982,-17.92043,-18.266861,-20.19513,-20.832294,-21.814098,-23.02499,-25.133183,-26.60372,-27.05175,-31.993168,-33.220516,
12	18.912994,-20.678759,-32.088844,-28.613203,-23.74313,-22.531338,-32.929653,-33.805977,-30.073614,-28.47012,-24.31829,-25.045925,-30.669214,-24.158752,-30.25639,-29.405184,-30.036154,-32.580357,-32.20659,-24.398947,-27.583939,-33.805977
13	-22.03035,-23.977226,-32.87102,-33.805977,-30.67499,-28.48288,-33.539997,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-28.846321,-25.384188,-28.53862,-30.71886,-33.805977,-33.805977,-33.805977,-31.166683,-33.805977
14	2,-31.536533,-27.698114,-24.892365,-29.269695,-23.425732,-22.834482,-29.375305,-30.345667,-25.158522,-24.107569,-28.005955,-26.987839,-33.805977,-28.604258,-32.28921,-29.195158,-31.219856,-33.805977,-30.877766,-33.805977,-31.899529,-33.
15	26.3708,-31.827585,-27.968035,-27.821455,-32.652718,-33.805977,-26.623596,-22.770489,-25.151245,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-31.156366,-24.906815,-26.098072,-28.355328,-30.561897,-33.805977,-33.805977,-33.805977,-30.603
16	-23.253487,-22.762703,-32.167393,-33.805977,-30.076311,-26.187855,-32.950005,-33.805977,-33.805977,-33.805977,-31.782179,-33.24273,-33.805977,-25.235683,-27.291262,-30.9268,-27.278545,-33.79094,-33.805977,-31.666782,-27.055733,-29.7107
17	3.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-33.74454,-29.311138,-33.805977,-32.532104,-33.805977,-33.805977,-29.648687,-31.274773,-33.805977,-33.805977,-33.805977,-33.805977,-28.693
18	,-23.872341,-33.805977,-33.805977,-33.805977,-33.109955,-24.81446,-25.047829,-31.76049,-33.805977,-33.805977,-33.805977,-33.805977,-28.822592,-33.805977,-31.612774,-29.836443,-33.805977,-33.805977,-33.805977,-32.2063,-28.403624,-27.2099
19	-26.764906,-26.11246,-33.805977,-33.805977,-30.370882,-27.844038,-33.805977,-33.805977,-33.805977,-29.41395,-27.32238,-33.805977,-31.233175,-24.779099,-22.8835,-23.493626,-30.028397,-32.660164,-30.516598,-26.570902,-26.652172,-33.80597
20	.818562,-27.701881,-31.55367,-33.699135,-33.805977,-33.805977,-33.805977,-33.805977,-33.805977,-32.19764,-33.805977,-33.805977,-33.805977,-27.810532,-24.595596,-30.20735,-33.805977,-30.550888,-33.805977,-33.805977,-33.28251,-33.805977,-
21	32.431667,-32.958626,-31.897984,-32.189503,-28.985867,-31.117514,-18.280468,-7.281423,-9.084869,-18.287346,-19.38128,-26.1226,-17.754103,-28.868862,-29.534191,-23.579418,-33.805977,-24.054642,-29.87237,-23.060852,-20.508898,-10.535711,-
22	-33.27043,-33.805977,-29.298115,-29.41055,-33.805977,-28.892399,-33.805977,-33.805977,-30.827553,-29.795609,-28.279217,-27.589552,-25.736511,-31.76239,-23.78833,-20.169075,-28.735235,-30.620928,-29.107738,-27.360458,-24.103697,-29.9004
23	.884865,-19.38567,-17.508791,-20.72262,-24.233171,-20.814669,-17.833494,-18.798656,-22.249092,-18.833117,-21.385603,-25.39892,-19.6954,-15.398192,-18.728706,-25.468416,-19.413794,-22.637587,-18.694334,-18.41118,-17.79503,-19.694656,-18.
24	21.553745,-21.868046,-23.169624,-24.199274,-22.437288,-24.834301,-24.571037,-23.567825,-20.891788,-19.605341,-20.693453,-23.486496,-20.885887,-21.485064,-20.40558,-20.36992,-21.28455,-21.842846,-22.88647,-17.57312,-16.708029,-20.490387
25	-33.805977,-26.048553,-21.981312,-22.606249,-25.96628,-25.617218,-28.557043,-27.928495,-33.805977,-33.805977,-33.805977,-25.911026,-25.3638,-27.829884,-23.630741,-20.39116,-24.905474,-26.871563,-23.534534,-21.455517,-23.143805,-26.5157
26	-15.712585,-15.983018,-17.524887,-19.14698,-17.320251,-14.536243,-17.08833,-18.052809,-15.619604,-15.934211,-15.877186,-15.04979,-16.174492,-14.737669,-13.547126,-14.736508,-17.253136,-20.094933,-18.567886,-14.070753,-12.069279,-11.645
27	6767,-22.908535,-22.60236,-28.39149,-29.05653,-24.579975,-19.913418,-18.076532,-33.805977,-23.321625,-27.810883,-25.64763,-31.424644,-21.229666,2.5053167,11.436528,10.8191395,11.876597,9.177105,-2.232955,-4.4741096,0.0010156466,2.337

5 Audio Model Training

In this part, sound_signal_model_training.py script has written to train 4 different models named as RandomForestClassifier, KNeighborsClassifier, DecisionTreeClassifier, SVC.

Importing necessary libraries, defining required paths, defining seed to make sure the results are not different from each other when executing, loading audio_signal and labels, lastly splitting audio signal into train and test parts to be used in training phase.



```python
# Libraries
import numpy as np
import seaborn as sns
import time
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from sklearn.metrics import confusion_matrix
import pickle


# Paths
path_spec_db = 'sounds/spec_db.csv'
path_labels = 'sounds/labels.csv'
path_scores = 'scores/'
path_confusion_matrix = 'confusion_matrix/'
path_models = 'models/'
path_plots = 'plots/'
path_best_model = 'best_model.sav'


seed = 50605057


# Get data
audio_signal = np.loadtxt(path_spec_db, delimiter=',').transpose()
audio_labels = np.loadtxt(path_labels, ndmin = 2).reshape(-1)


# Splitting
X_train, X_test, y_train, y_test = train_test_split(audio_signal, audio_labels, test_size = 0.25, random_state = seed)
```

Defining commands recorded, and defining arrays to save the scores after the training.

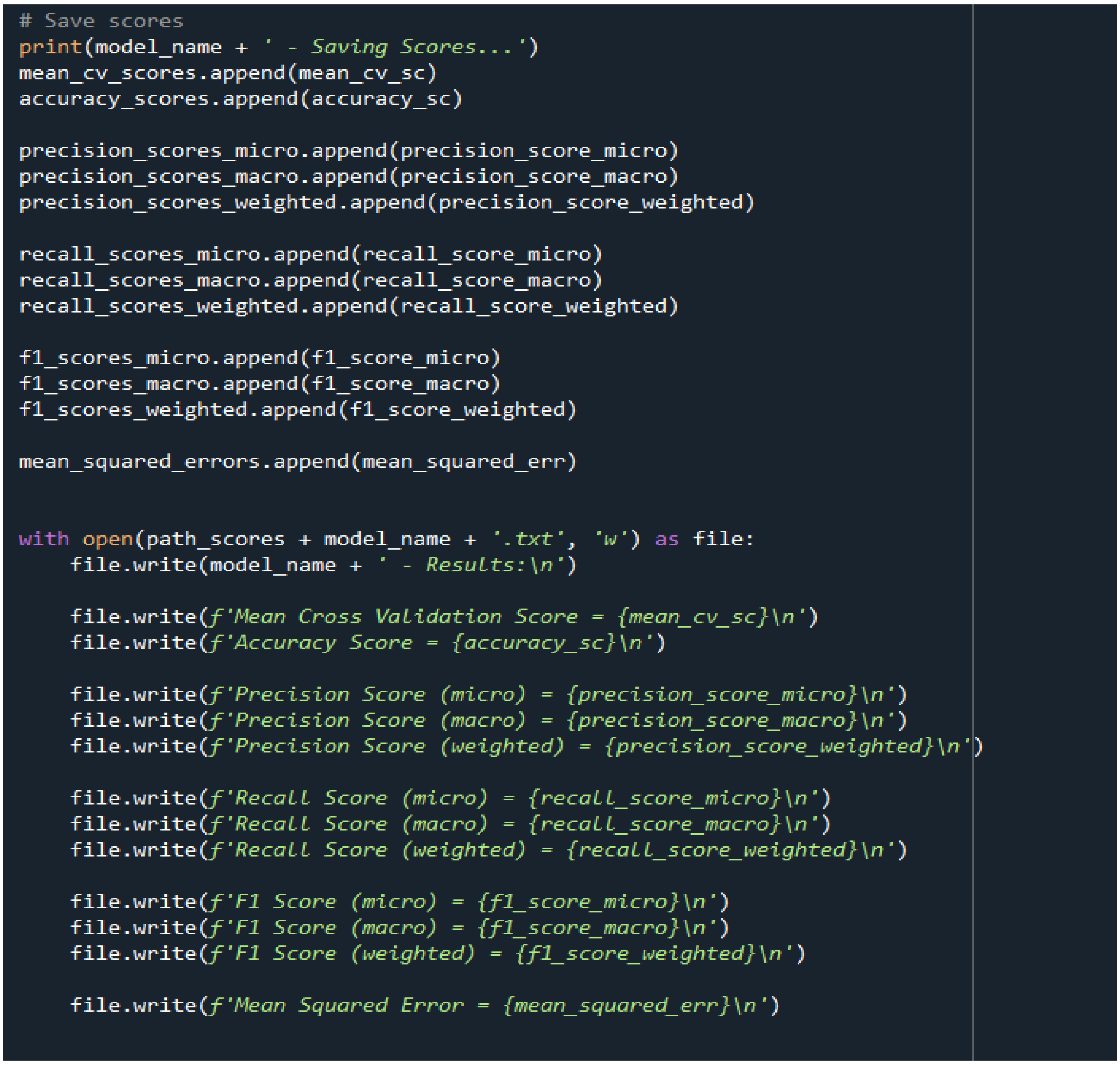
```
# Confusion matrix labels
labels_y = ['wait', 'computer', 'engineering', 'name', 'surname', 'cough', 'clap_hands', 'snap_finger']


# To store outputs
model_names = []

mean_cv_scores = []
accuracy_scores = []

precision_scores_micro = []
precision_scores_macro = []
precision_scores_weighted = []

recall_scores_micro = []
recall_scores_macro = []
recall_scores_weighted = []

f1_scores_micro = []
f1_scores_macro = []
f1_scores_weighted = []

mean_squared_errors = []

times = []
performances = []
```

TRAINING PART

Model training and calculating scores.

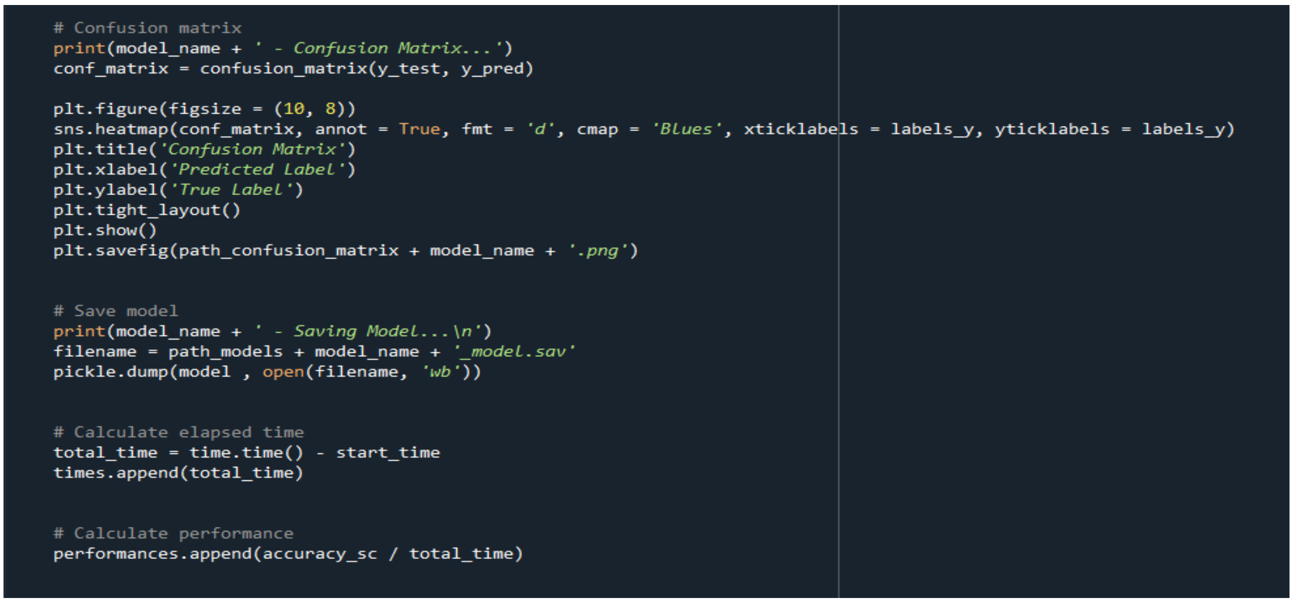
```
for model_name, model in models:
    # Start time
    start_time = time.time()
    model_names.append(model_name)
    # Model training
    print(model_name + ' - Training...')
    model.fit(X_train, y_train)

    # Prediction
    print(model_name + ' - Predicting...')
    y_pred = model.predict(X_test)


    # Calculate cross validation, accuracy, precision, recall, f1, mse scores
    print(model_name + ' - Calculating Scores...')
    cv_scores = cross_val_score(model, X_train, y_train, cv = 10)
    mean_cv_sc = cv_scores.mean()

    accuracy_sc = accuracy_score(y_test, y_pred)

    precision_score_micro = precision_score(y_test, y_pred, average = 'micro')
    precision_score_macro = precision_score(y_test, y_pred, average = 'macro')
    precision_score_weighted = precision_score(y_test, y_pred, average = 'weighted')

    recall_score_micro = recall_score(y_test, y_pred, average = 'micro')
    recall_score_macro = recall_score(y_test, y_pred, average = 'macro')
    recall_score_weighted = recall_score(y_test, y_pred, average = 'weighted')

    f1_score_micro = f1_score(y_test, y_pred, average = 'micro')
    f1_score_macro = f1_score(y_test, y_pred, average = 'macro')
    f1_score_weighted = f1_score(y_test, y_pred, average = 'weighted')

    mean_squared_err = mean_squared_error(y_test, y_pred)
```

Saving scores into *txt.



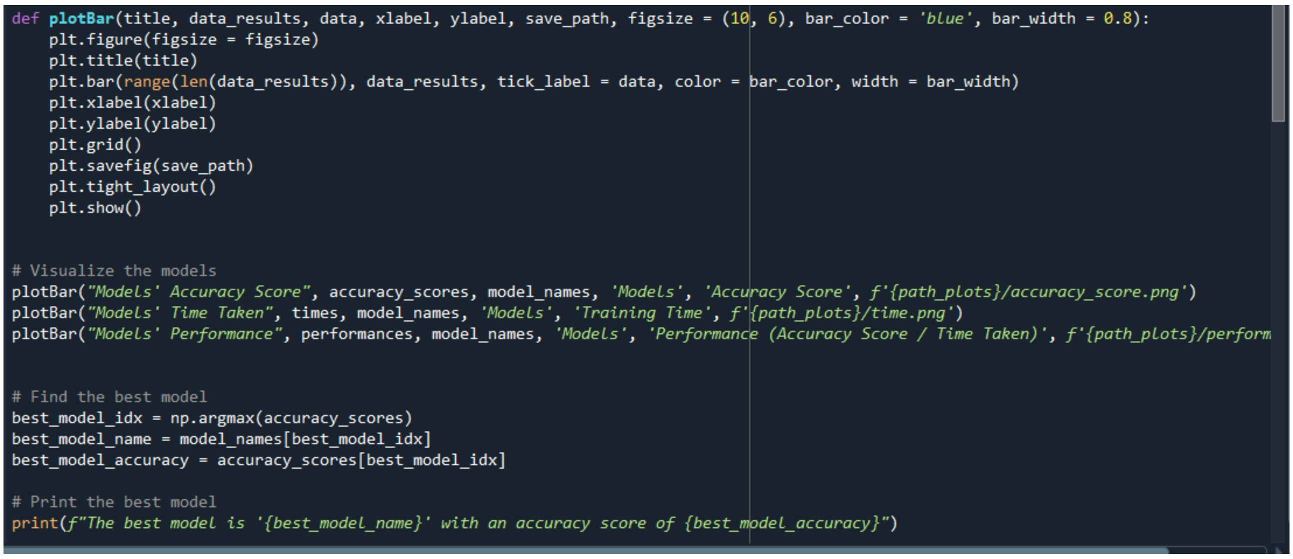
```python
# Save scores
print(model_name + ' - Saving Scores...')
mean_cv_scores.append(mean_cv_sc)
accuracy_scores.append(accuracy_sc)

precision_scores_micro.append(precision_score_micro)
precision_scores_macro.append(precision_score_macro)
precision_scores_weighted.append(precision_score_weighted)

recall_scores_micro.append(recall_score_micro)
recall_scores_macro.append(recall_score_macro)
recall_scores_weighted.append(recall_score_weighted)

f1_scores_micro.append(f1_score_micro)
f1_scores_macro.append(f1_score_macro)
f1_scores_weighted.append(f1_score_weighted)

mean_squared_errors.append(mean_squared_err)


with open(path_scores + model_name + '.txt', 'w') as file:
    file.write(model_name + ' - Results:\n')

    file.write(f'Mean Cross Validation Score = {mean_cv_sc}\n')
    file.write(f'Accuracy Score = {accuracy_sc}\n')

    file.write(f'Precision Score (micro) = {precision_score_micro}\n')
    file.write(f'Precision Score (macro) = {precision_score_macro}\n')
    file.write(f'Precision Score (weighted) = {precision_score_weighted}\n')

    file.write(f'Recall Score (micro) = {recall_score_micro}\n')
    file.write(f'Recall Score (macro) = {recall_score_macro}\n')
    file.write(f'Recall Score (weighted) = {recall_score_weighted}\n')

    file.write(f'F1 Score (micro) = {f1_score_micro}\n')
    file.write(f'F1 Score (macro) = {f1_score_macro}\n')
    file.write(f'F1 Score (weighted) = {f1_score_weighted}\n')

    file.write(f'Mean Squared Error = {mean_squared_err}\n')
```

Plotting Confusion matrix.



```python
    # Confusion matrix
    print(model_name + ' - Confusion Matrix...')
    conf_matrix = confusion_matrix(y_test, y_pred)

    plt.figure(figsize = (10, 8))
    sns.heatmap(conf_matrix, annot = True, fmt = 'd', cmap = 'Blues', xticklabels = labels_y, yticklabels = labels_y)
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.tight_layout()
    plt.show()
    plt.savefig(path_confusion_matrix + model_name + '.png')


    # Save model
    print(model_name + ' - Saving Model...\n')
    filename = path_models + model_name + '_model.sav'
    pickle.dump(model , open(filename, 'wb'))


    # Calculate elapsed time
    total_time = time.time() - start_time
    times.append(total_time)


    # Calculate performance
    performances.append(accuracy_sc / total_time)
```

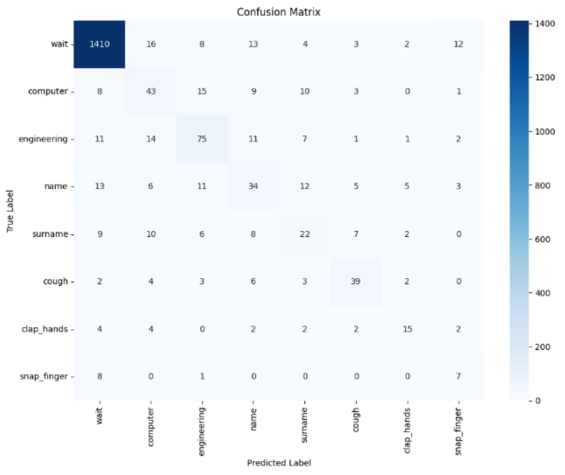
Plotting Time Performance Graphs.



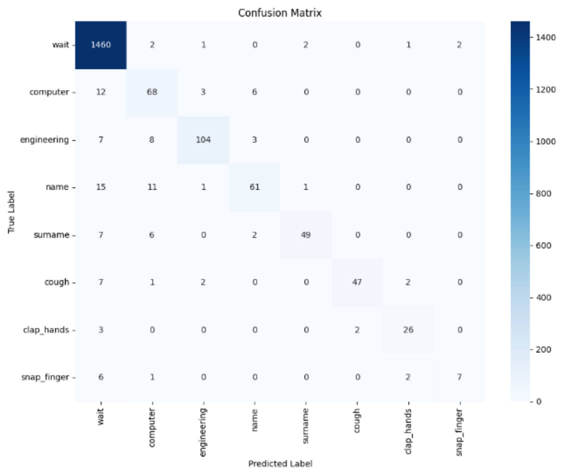
```
def plotBar(title, data_results, data, xlabel, ylabel, save_path, figsize = (10, 6), bar_color = 'blue', bar_width = 0.8):
    plt.figure(figsize = figsize)
    plt.title(title)
    plt.bar(range(len(data_results)), data_results, tick_label = data, color = bar_color, width = bar_width)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.grid()
    plt.savefig(save_path)
    plt.tight_layout()
    plt.show()


# Visualize the models
plotBar("Models' Accuracy Score", accuracy_scores, model_names, 'Models', 'Accuracy Score', f'{path_plots}/accuracy_score.png')
plotBar("Models' Time Taken", times, model_names, 'Models', 'Training Time', f'{path_plots}/time.png')
plotBar("Models' Performance", performances, model_names, 'Models', 'Performance (Accuracy Score / Time Taken)', f'{path_plots}/perform


# Find the best model
best_model_idx = np.argmax(accuracy_scores)
best_model_name = model_names[best_model_idx]
best_model_accuracy = accuracy_scores[best_model_idx]

# Print the best model
print(f"The best model is '{best_model_name}' with an accuracy score of {best_model_accuracy}")
```

Results of execution.

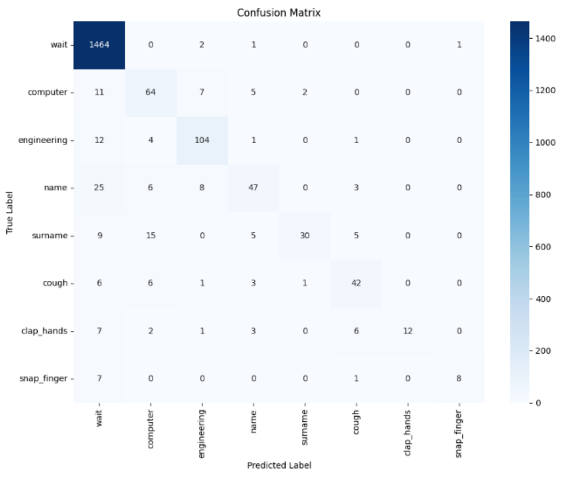
DT



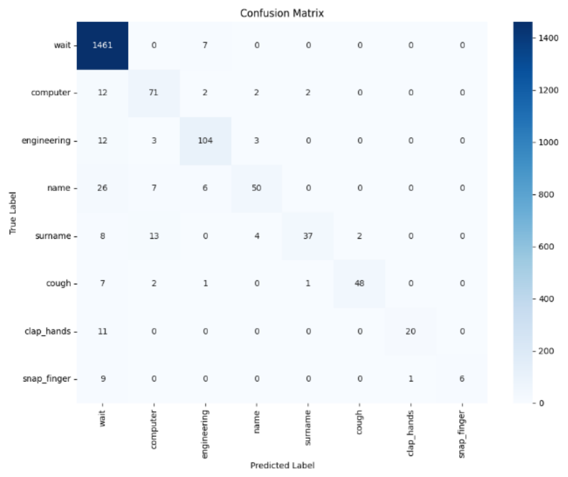
KNN



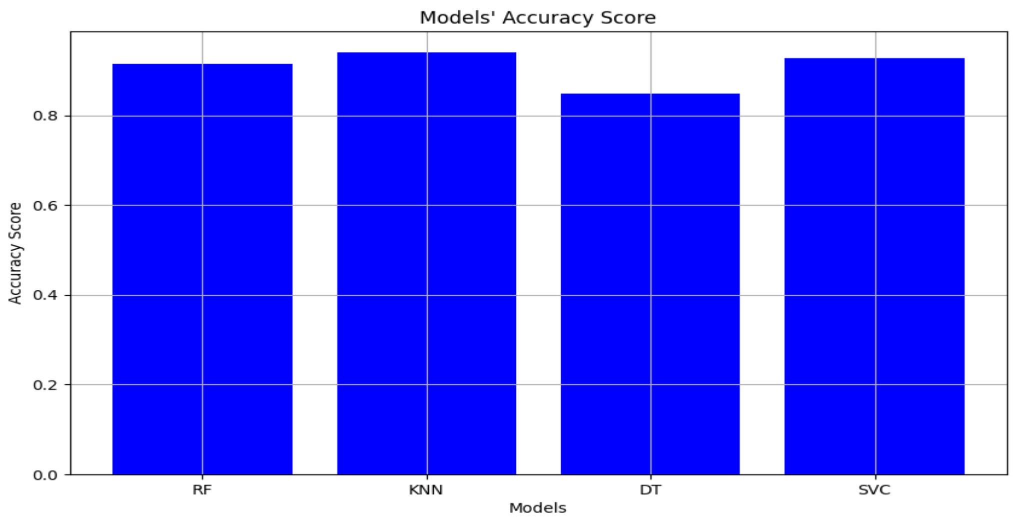
RF



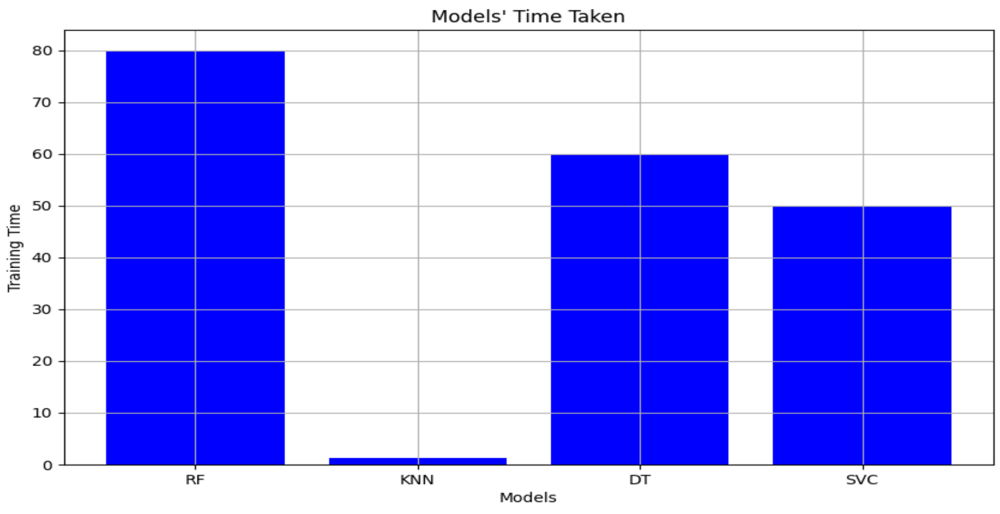
SVC



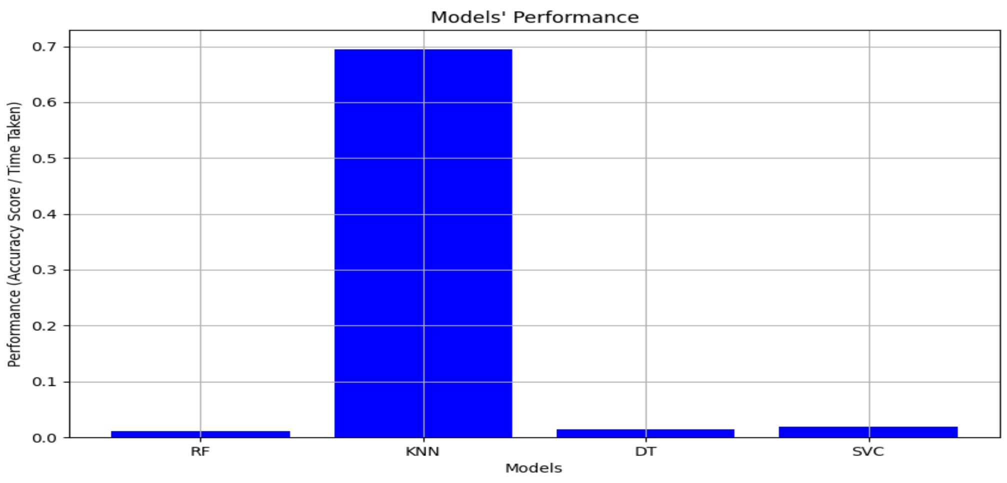
Accuracy Score



Time



Performance

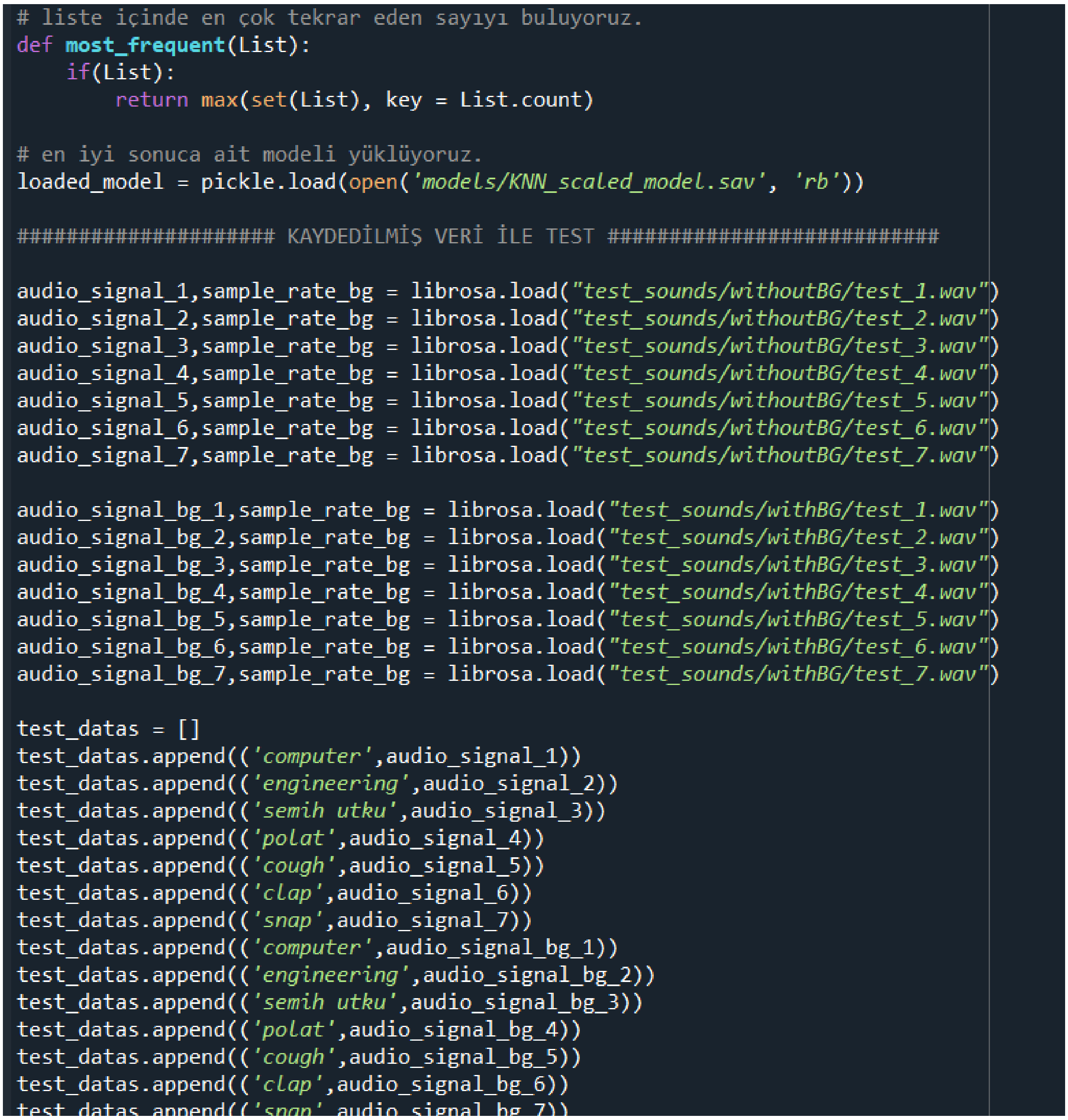


6 Audio Model Testing

In this part, sound_signal_model_testing.py script has written to predict recorded audios. Two types of voice recordings were used in this script: pre-recorded and recorded during the execution of the code.

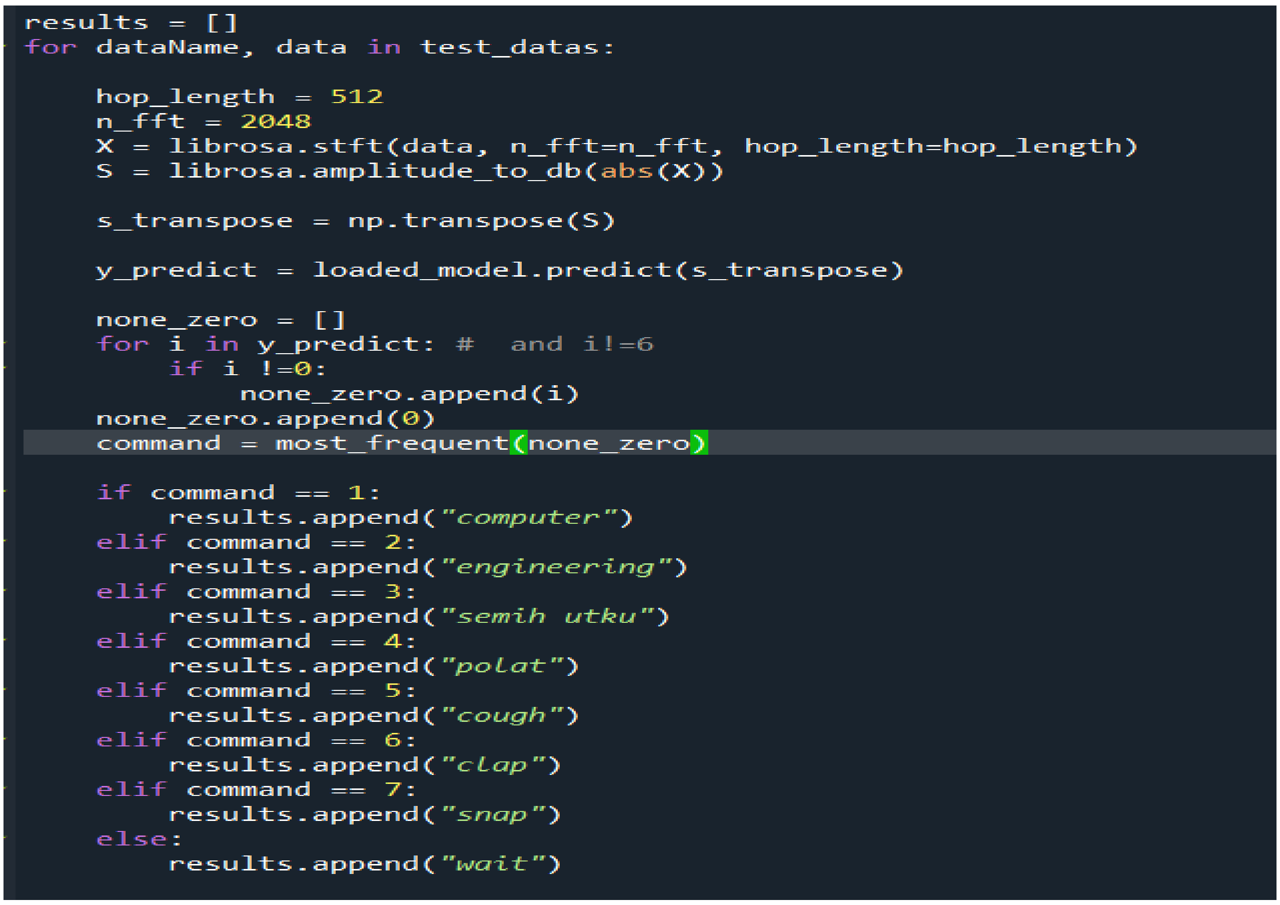
PRE-RECORDED AUDIO FILES

Load pre-recorded audio files with and without background sound. Then, add audio files to the list to use in the loop.



```
# liste içinde en çok tekrar eden sayıyı buluyoruz.
def most_frequent(List):
    if(List):
        return max(set(List), key = List.count)

# en iyi sonuca ait modeli yüklüyoruz.
loaded_model = pickle.load(open('models/KNN_scaled_model.sav', 'rb'))

##################### KAYDEDİLMİŞ VERİ İLE TEST ###########################

audio_signal_1,sample_rate_bg = librosa.load("test_sounds/withoutBG/test_1.wav")
audio_signal_2,sample_rate_bg = librosa.load("test_sounds/withoutBG/test_2.wav")
audio_signal_3,sample_rate_bg = librosa.load("test_sounds/withoutBG/test_3.wav")
audio_signal_4,sample_rate_bg = librosa.load("test_sounds/withoutBG/test_4.wav")
audio_signal_5,sample_rate_bg = librosa.load("test_sounds/withoutBG/test_5.wav")
audio_signal_6,sample_rate_bg = librosa.load("test_sounds/withoutBG/test_6.wav")
audio_signal_7,sample_rate_bg = librosa.load("test_sounds/withoutBG/test_7.wav")

audio_signal_bg_1,sample_rate_bg = librosa.load("test_sounds/withBG/test_1.wav")
audio_signal_bg_2,sample_rate_bg = librosa.load("test_sounds/withBG/test_2.wav")
audio_signal_bg_3,sample_rate_bg = librosa.load("test_sounds/withBG/test_3.wav")
audio_signal_bg_4,sample_rate_bg = librosa.load("test_sounds/withBG/test_4.wav")
audio_signal_bg_5,sample_rate_bg = librosa.load("test_sounds/withBG/test_5.wav")
audio_signal_bg_6,sample_rate_bg = librosa.load("test_sounds/withBG/test_6.wav")
audio_signal_bg_7,sample_rate_bg = librosa.load("test_sounds/withBG/test_7.wav")

test_datas = []
test_datas.append(('computer',audio_signal_1))
test_datas.append(('engineering',audio_signal_2))
test_datas.append(('semih utku',audio_signal_3))
test_datas.append(('polat',audio_signal_4))
test_datas.append(('cough',audio_signal_5))
test_datas.append(('clap',audio_signal_6))
test_datas.append(('snap',audio_signal_7))
test_datas.append(('computer',audio_signal_bg_1))
test_datas.append(('engineering',audio_signal_bg_2))
test_datas.append(('semih utku',audio_signal_bg_3))
test_datas.append(('polat',audio_signal_bg_4))
test_datas.append(('cough',audio_signal_bg_5))
test_datas.append(('clap',audio_signal_bg_6))
test_datas.append(('snap',audio_signal_bg_7))
```

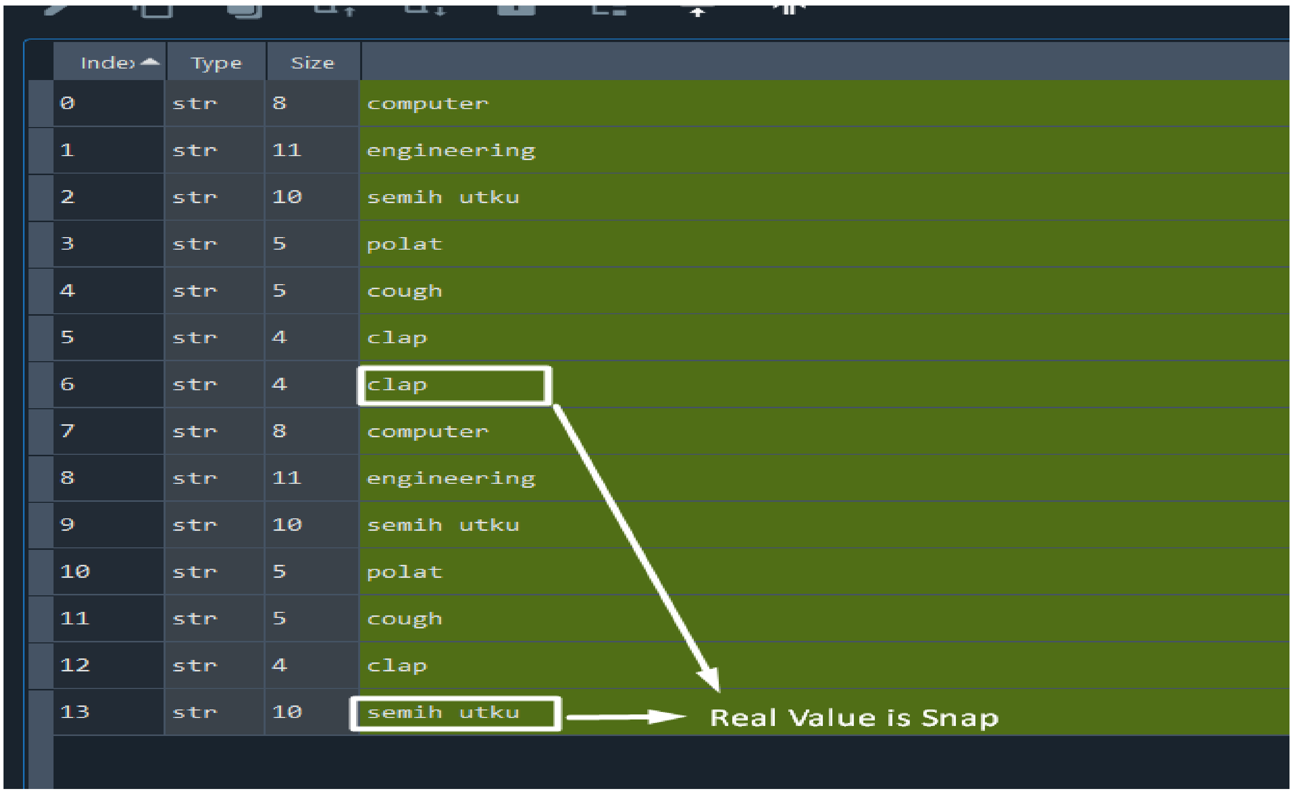
Each sound file was adapted using short term fourier transform, label prediction was made. Then the most guessed label printed.



```
results = []
for dataName, data in test_datas:

    hop_length = 512
    n_fft = 2048
    X = librosa.stft(data, n_fft=n_fft, hop_length=hop_length)
    S = librosa.amplitude_to_db(abs(X))

    s_transpose = np.transpose(S)

    y_predict = loaded_model.predict(s_transpose)

    none_zero = []
    for i in y_predict: #  and i!=6
        if i !=0:
            none_zero.append(i)
    none_zero.append(0)
    command = most_frequent(none_zero)

    if command == 1:
        results.append("computer")
    elif command == 2:
        results.append("engineering")
    elif command == 3:
        results.append("semih utku")
    elif command == 4:
        results.append("polat")
    elif command == 5:
        results.append("cough")
    elif command == 6:
        results.append("clap")
    elif command == 7:
        results.append("snap")
    else:
        results.append("wait")
```

Results of Execution:

10 prediction out of 12 prediction is true..



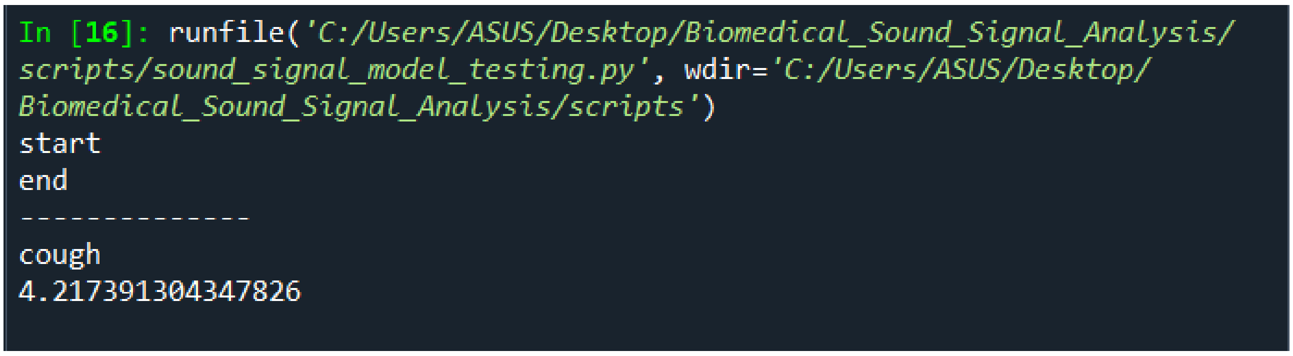
Index	Type	Size	
0	str	8	computer
1	str	11	engineering
2	str	10	semih utku
3	str	5	polat
4	str	5	cough
5	str	4	clap
6	str	4	clap
7	str	8	computer
8	str	11	engineering
9	str	10	semih utku
10	str	5	polat
11	str	5	cough
12	str	4	clap
13	str	10	semih utku

RECORDING DURING THE EXECUTION

In this part, everything is the same except that we record and use it to predict while the code is running.

Results of execution:

Cough record result from recording during the execution



```
In [16]: runfile('C:/Users/ASUS/Desktop/Biomedical_Sound_Signal_Analysis/
scripts/sound_signal_model_testing.py', wdir='C:/Users/ASUS/Desktop/
Biomedical_Sound_Signal_Analysis/scripts')
start
end
--------------
cough
4.217391304347826
```