#2: Exploratory Data Analysis

Data Science - Capstone Project Submission

- Student Name: **James Toop**
- Student Pace: **Self Paced**
- Scheduled project review date/time: **29th October 2021 @ 21:30 BST**
- Instructor name: **Jeff Herman / James Irving**
- Blog URL: **https://toopster.github.io/ (https://toopster.github.io/)**

Table of Contents

---

2. Exploratory Data Analysis ¶

2A. The Datasets

Speech Commands: A dataset for limited-vocabulary speech recognition – (Pete Warden, TensorFlow team at Google)

https://arxiv.org/abs/1804.03209 (https://arxiv.org/abs/1804.03209)

The Speech Commands dataset is an attempt to build a standard training and evaluation dataset for a class of simple speech recognition tasks. Its primary goal is to provide a way to build and test small models that detect when a single word is spoken, from a set of ten or fewer target words, with as few false background noise or

unrelated speech.

Ultrasuite: A collection of ultrasound and acoustic speech data from child speech therapy sessions – (University of Edinburgh, School of Infomatics)

https://ultrasuite.github.io/ (https://ultrasuite.github.io/)

Ultrasuite is a collection of ultrasound and acoustic speech data from child speech therapy sessions. The current release includes three datasets, one from typically developing children and two from speech disordered children:

- **Ultrax Typically Developing (UXTD) (https://ultrasuite.github.io/data/uxtd/)** - A dataset of 58 typically developing children.
- **Ultrax Speech Sound Disorders (UXSSD) (https://ultrasuite.github.io/data/uxssd/)** - A dataset of 8 children with speech sound disorders.
- **UltraPhonix (UPX) (https://ultrasuite.github.io/data/upx/)** - A second dataset of children with speech sound disorders, collected from 20 children.

Source:
Eshky, A., Ribeiro, M. S., Cleland, J., Richmond, K., Roxburgh, Z., Scobbie, J., & Wrench, A. (2018) Ultrasuite: A repository of ultrasound and acoustic data from child speech therapy sessions. Proceedings of INTERSPEECH. Hyderabad, India. [paper (https://ultrasuite.github.io/papers/ultrasuite_IS18.pdf)]

IMPORTANT NOTE:

The datasets and transformed JSON files have not been included in the GitHub repository with this notebook and will need to be downloaded and stored in the local repository for the code to run correctly.

The code below will however, download, store and transform the datasets as required for the models to run. But, to ensure ease of use, it is also possible to download the raw and transformed datasets using the following link.

2B. Data Discovery

This section presents an initial step to investigate, understand and document the available data fields and relationships, highlighting any potential issues / shortcomings within the datasets supplied.

In [22]:

```
# Import required libraries and modules for data preprocessing
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from pydub import AudioSegment
import wave
import soundfile as sf
import librosa, librosa.display
import IPython.display as ipd

import tensorflow as tf
from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras import layers
from tensorflow.keras import models

import os
import pathlib
from pathlib import Path
import shutil
import collections
import nltk
```

Download Speech Commands v0.02 dataset

The Speech Commands dataset contains `105,829` audio samples of `35` spoken keywords in `.wav` format, no longer than 1 second and with a sample rate of `16000`. The dataset has been organised such that each audio sample is stored within a folder that has been named so that is corresponds with it's associated label as per the following example:

```
└── speech_commands_v0.02

    ├── backward

        ├── 0a2b400e_nohash_0.wav
        ├── 0a2b400e_nohash_1.wav
        ├── 0a2b400e_nohash_2.wav
        ├── 0a2b400e_nohash_3.wav
        ├── 0a396ff2_nohash_0.wav
        └── ...

    ├── bed

        ├── 0a7c2a8d_nohash_0.wav
        └── ...
```

In [3]:

```
def download_speech_commands():
    '''
    Downloads and unpacks the speech commands dataset, removing any unnecessary file
    '''
    data_dir = pathlib.Path('data/speech_commands_v0.02')

    # Check to see if data directory already exists, download if not
    if not data_dir.exists():
        tf.keras.utils.get_file(
            'speech_commands_v0.02.zip',
            origin='http://download.tensorflow.org/data/speech_commands_v0.02.tar.gz
            extract=True,
            cache_dir='.',
            cache_subdir='data/speech_commands_v0.02')
    else:
        print('Speech Commands dataset already exists')

    # Remove the _background_noise_ samples as these are not required
    try:
        shutil.rmtree(str(data_dir) + '/_background_noise_')
    except OSError as e:
        print('Error: %s - %s.' % (e.filename, e.strerror), 'Check if directory has


    # Remove the extracted zip file for politeness as this is also not required
    zip_file = str(data_dir) + '/speech_commands_v0.02.zip'
    if os.path.exists(zip_file):
        os.remove(zip_file)
```

In [3]:

```
# Call the function to download the Speech Commands v0.02 dataset
download_speech_commands()
```

```
Speech Commands dataset already exists
Error: data/speech_commands_v0.02/_background_noise_ - No such file or
directory. Check if directory has already been removed.
```

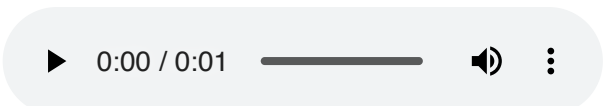
Preview audio samples and waveforms from Speech Commands dataset

In [24]:

```
# Load the audio sample and preview
target_sample = 'data/speech_commands_v0.02/zero/0a2b400e_nohash_0.wav'
sc_sample, sr = librosa.load(target_sample)
print('Audio sample: Zero')
ipd.Audio(sc_sample, rate=sr)
```

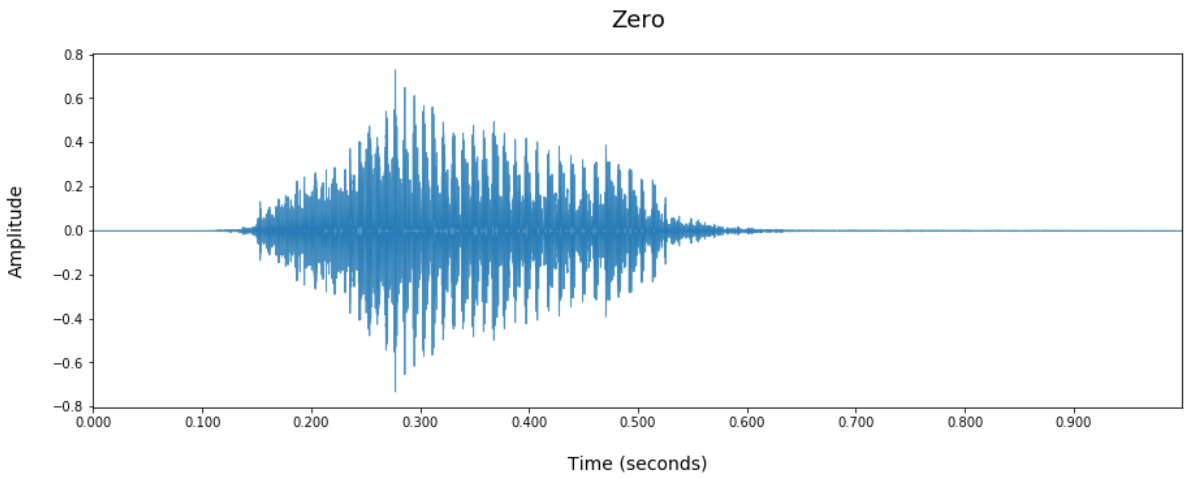
```
Audio sample: Zero
```

Out[24]:



In [25]:

```
# Plot the waveform for the specific audio sample
plt.figure(figsize=(15, 5))
plt.title('Zero', fontsize=18, pad=20)
librosa.display.waveplot(sc_sample, sr, alpha=0.8)
plt.xlabel('Time (seconds)', fontsize=14, labelpad=20)
plt.ylabel('Amplitude', fontsize=14, labelpad=20)
plt.show();
```



In [26]:

```
# Extract the short time Fourier transform and preview the array shape
hop_length = 512
n_fft = 2048

S_sc_sample = librosa.stft(sc_sample, n_fft=n_fft, hop_length=hop_length)
S_sc_sample.shape
```

Out[26]:

```
(1025, 44)
```

In [7]:

```
type(S_sc_sample[0][0])
```

Out[7]:

```
numpy.complex64
```

In [8]:

```
# Calculating the spectrogram
Y_sc_sample = np.abs(S_sc_sample) ** 2
```

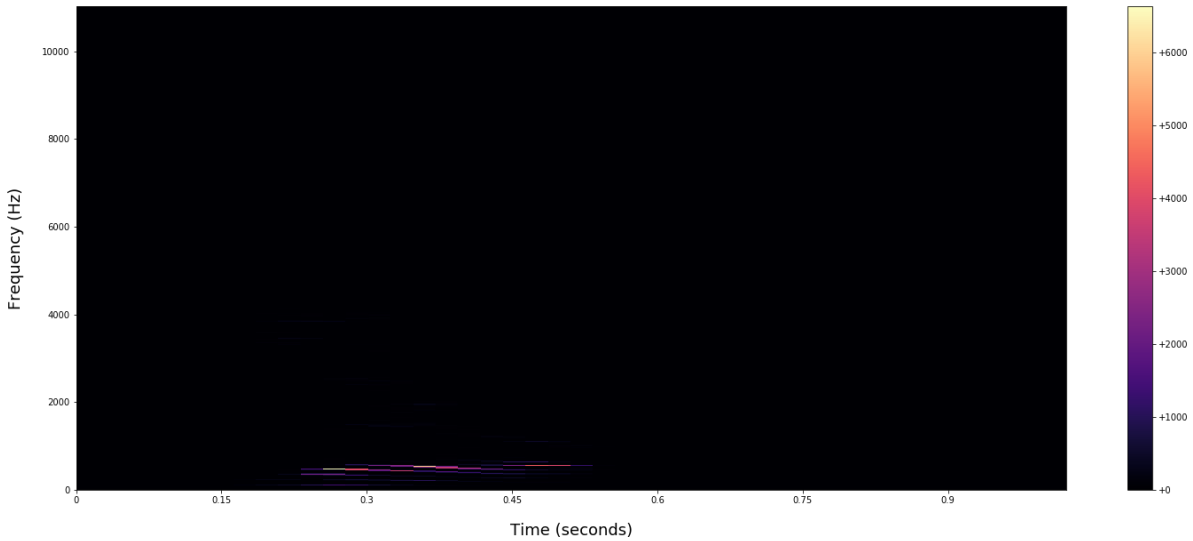
In [37]:

```
def plot_spectrogram(Y, sr, hop_length, y_axis='linear'):
    '''
    Plot and visualise the spectrogram

        Params:
            Y (ndarray): Spectrogram to display
            sr (int): Sample Rate
            hop_length (int): Hop Length
            y_axis (str): Range for the y-axis
    '''
    # Portrait - presentation format
#     fig = plt.figure(figsize=(10, 8))
    # Landscape - Jupyter notebook format
    fig = plt.figure(figsize=(25, 10))
    librosa.display.specshow(Y,
                             sr=sr,
                             hop_length=hop_length,
                             x_axis='time',
                             y_axis=y_axis)
    fig.gca().set_xlabel('Time (seconds)', fontsize=18, labelpad=20)
    fig.gca().set_ylabel('Frequency (Hz)', fontsize=18, labelpad=20)
    plt.colorbar(format='%+2.f')
```

In [10]:

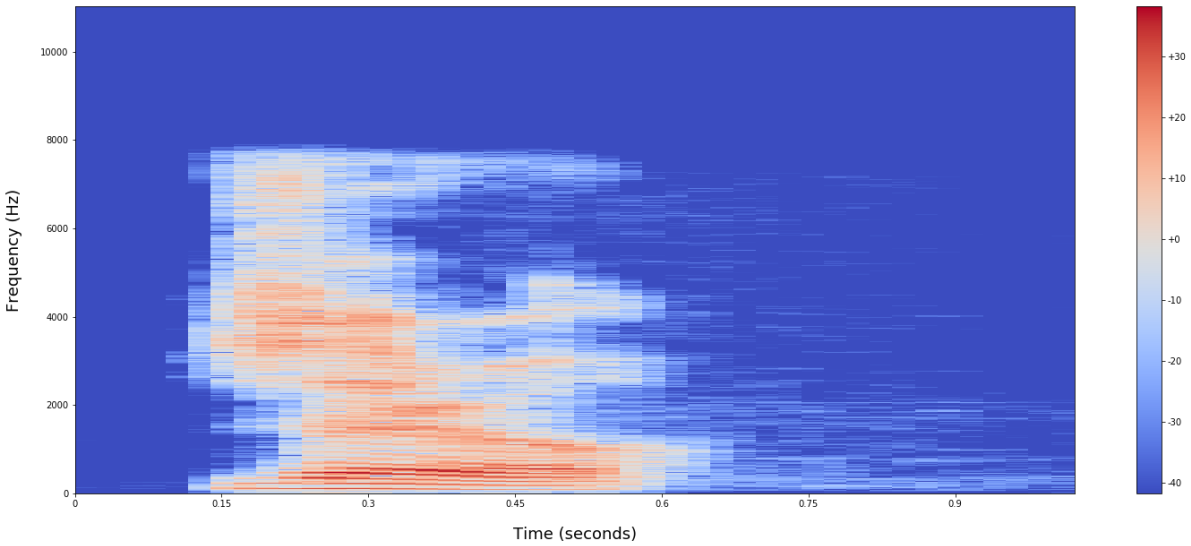
```
# Plotting the spectrogram for our example from the Speech Commands dataset
plot_spectrogram(Y_sc_sample, sr, hop_length)
```



The human perception of sound intensity is logarithmic in nature so we are interested in the log amplitude.

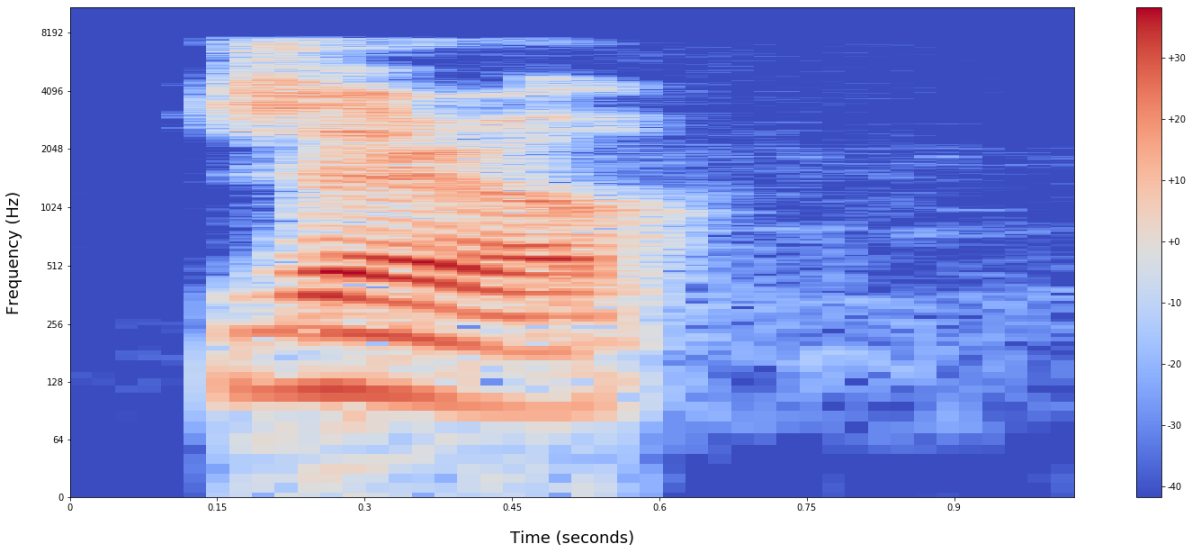
In [11]:

```
# Display spectrogram using log amplitude
Y_log_sc_sample = librosa.power_to_db(Y_sc_sample)
plot_spectrogram(Y_log_sc_sample, sr, hop_length)
```



In [12]:

```
# Display spectrogram using log frequency
plot_spectrogram(Y_log_sc_sample, sr, hop_length, y_axis='log')
```



Download the Ultrasuite dataset

In its raw format (i.e. as downloaded), the Ultrasuite dataset contains `14,286` audio samples from `86` different children from multiple speech therapy sessions in `.wav` format with a sample rate of `22050`. The audio samples are of varying lengths containing spoken words from both the child and, in some cases, the speech therapist.

Each audio sample is given a "grading", A - E that denotes the decreasing quality of the audio sample and capture. As a result, labels are not typically supplied for grades D and E.

In order to prepare the datasets prior to training the deep learning models, we need to download, splice and label each of the audio samples from the Ultrasuite datasets. We also need to transform them into the same structure as the Speech Commands dataset.

Once spliced and cleansed, as detailed below, the transformed Ultrasuite dataset contains `33,800` audio samples with a sample rate of `16000` lasting at least 1 second with `991` different words spoken.

Currently each dataset is organised by a child subject ID, session, and then sample ID, for example:

```
└── ultrasuite
    ├── core-uxssd
        ├── core
            ├── 01M  (Child ID)
                ├── BL1  (Session ID)
                    ├── 001A.wav (Sample ID)
                    ├── 002A.wav
                    ├── 003A.wav
                    ├── 004A.wav
                    ├── 005A.wav
                    └── ...
                ├── Maint1
                    ├── 001A.wav
                    ├── 002A.wav
                    ├── 003A.wav
                    └── ...
```

In [13]:

```
# Function for downloading the Ultrasuite datasets
def download_ultrasuite(datasets):
    '''
    Sets up a remote sync for the Ultrasuite datasets and labels

        Params:
            datasets (list): Specific ultrasuite dataset to sync can be 'upx', 'uxtd

    '''
    orig_loc = Path.cwd()
    data_dir = pathlib.Path('data/ultrasuite')

    # Check to see if data directory already exists, download if not
    if not os.path.isdir(data_dir):
        os.makedirs(data_dir)

        # Change working directory
        os.chdir(data_dir)

        for dataset in datasets:
            os.system('rsync -av --include="*/" --include="*.wav" --exclude="*" ultra
            print(dataset, 'dataset has been downloaded.')

        os.system('rsync -av ultrasuite-rsync.inf.ed.ac.uk::ultrasuite/labels-uxtd-u
        print('The ultrasuite labels have been downloaded.')

        # Change working directory back
        os.chdir(orig_loc)
```

In [14]:

```
# Download the Ultrasuite datasets
download_ultrasuite(['upx', 'uxtd', 'uxssd'])
```

```
upx dataset has been downloaded.
uxtd dataset has been downloaded.
uxssd dataset has been downloaded.
The labels have been downloaded.
```

Preview audio samples, waveforms, spectrograms and labels from Ultrasuite dataset

In [4]:

```
# Function to get the audio sample file statistics based on a target directory
def get_filestats(src_directory):
    '''
    Gets the audio sample file statistics based on a target directory and loads into

        Params:
            src_directory (str): Target directory containing the *.wav files for ger

        Returns:
            filestats_df (pandas.core.frame.DataFrame): Pandas dataframe containing
    '''
    src_files = Path.cwd() / src_directory
    filedata = []

    for src_file in src_files.glob('**/*.wav'):

        if src_file.is_file():
            filedata.append([src_file.parent.parts[-1],
                             src_file.stem + src_file.suffix,
                             librosa.get_duration(filename=src_file),
                             librosa.get_samplerate(src_file)])

    if src_directory.find('_transformed') != -1:
        columns = ['sample_utterance', 'sample_filename', 'sample_duration', 'sample
    else:
        columns = ['sample_speaker', 'sample_filename', 'sample_duration', 'sample_s

    filestats_df = pd.DataFrame(data=filedata, columns=columns)

    return filestats_df
```

In [6]:

```
# Load the Ultrasuite raw file information into a dataframe
raw_ultrasuite_filestats = get_filestats('data/ultrasuite')
raw_ultrasuite_filestats.head()
```

Out[6]:

	sample_speaker	sample_filename	sample_duration	sample_samplerate
0	39F	037D.wav	21.733878	22050
1	39F	021D.wav	3.622313	22050
2	39F	017D.wav	2.414875	22050
3	39F	040D.wav	9.659501	22050
4	39F	041D.wav	3.018594	22050

In [7]:

```
# Preview the number of audio samples in the raw Ultrasuite dataset
len(raw_ultrasuite_filestats)
```

Out[7]:

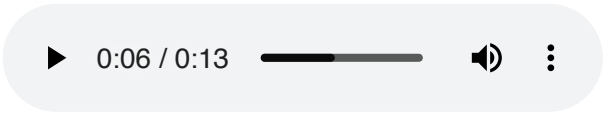
```
14286
```

In [26]:

```
# Load the audio sample and preview
target_sample = 'data/ultrasuite/core-uxssd/core/06M/BL1/002A.wav'
us_sample, sr = librosa.load(target_sample)
print('Audio sample: Car | Girl | Moon | Knife')
ipd.Audio(us_sample, rate=sr)
```

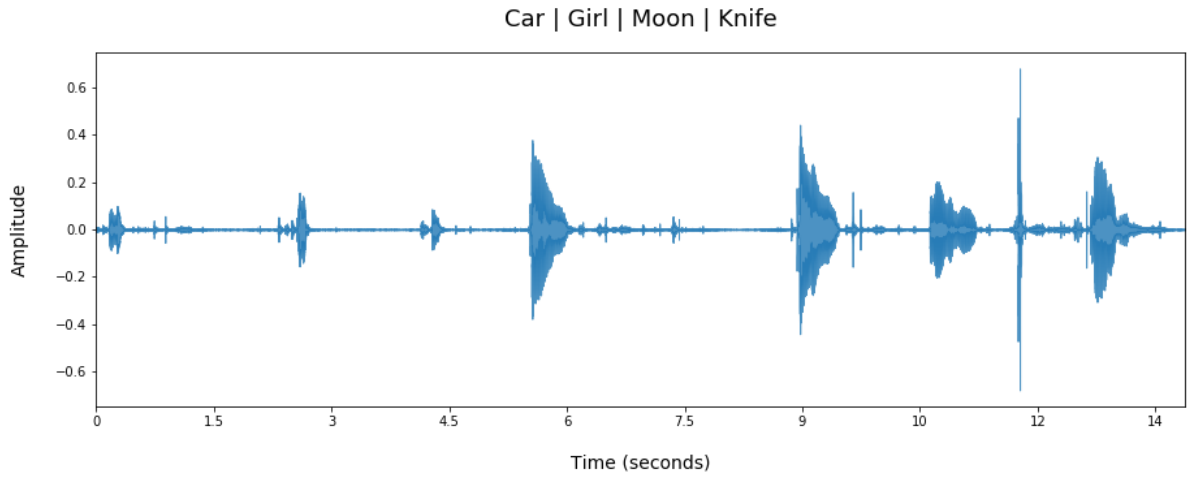
```
Audio sample: Car | Girl | Moon | Knife
```

Out[26]:



In [27]:

```
# Plot the waveform for the specific audio sample
plt.figure(figsize=(15, 5))
plt.title('Car | Girl | Moon | Knife', fontsize=18, pad=20)
librosa.display.waveplot(us_sample, sr, alpha=0.8)
plt.xlabel('Time (seconds)', fontsize=14, labelpad=20)
plt.ylabel('Amplitude', fontsize=14, labelpad=20)
plt.show();
```



In [28]:

```
def all_ultrasuite_word_labels(src_directory, src_dataset):
    '''
    Extracts and combines the labels from all *.lab files into a single DataFrame

        Params:
            src_directory (str): Target directory containing the *.wav files for ger

        Returns:
            all_labels_df (pandas.core.frame.DataFrame): DataFrame containing all la
    '''
    directory = src_directory + src_dataset + '/word_labels/lab/'
    columns = ['start_time', 'end_time', 'utterance']
    all_labels_df = pd.DataFrame()

    for filename in os.listdir(directory):

        filepath = directory + filename

        labels_df = pd.read_csv(filepath, sep=" ", header=None, names=columns)

        # Extract the speaker, session and speech data from the filename and add to
        labels_df['dataset'] = src_dataset
        labels_df['speaker'] = filename[0:3]
        if len(filename[4:-9]) == 0:
            labels_df['session'] = None
        else:
            labels_df['session'] = filename[4:-9]
        labels_df['speech_waveform'] = filename[-8:-4]

        # Tidy up data formatting and correct time based units
        labels_df['utterance'] = labels_df['utterance'].str.lower()
        labels_df['start_time'] = pd.to_timedelta(labels_df['start_time'] * 100)
        labels_df['end_time'] = pd.to_timedelta(labels_df['end_time'] * 100)

        # Append incoming labels to existing dataframe
        all_labels_df = all_labels_df.append(labels_df, ignore_index=True)

    return all_labels_df
```

In [29]:

```
# Load the labels for the Ultrax Speech Sound Disorders dataset
uxssd_df = all_ultrasuite_word_labels('data/ultrasuite/labels-uxtd-uxssd-upx/', 'uxs
```

In [30]:

```
# Preview the data
uxssd_df.head()
```

Out[30]:

	start_time	end_time	utterance	dataset	speaker	session	speech_waveform
0	00:00:01.340000	00:00:02.040000	th	uxssd	02M	BL1	069B
1	00:00:02.460000	00:00:03.350000	atha	uxssd	02M	BL1	069B
2	00:00:03.790000	00:00:04.650000	eethee	uxssd	02M	BL1	069B
3	00:00:05.210000	00:00:06.110000	otho	uxssd	02M	BL1	069B
4	00:00:00.970000	00:00:01.480000	core	uxssd	04M	Maint1	017A

In [31]:

```
# Load the labels for the Ultrax Typically Developing dataset
uxtd_df = all_ultrasuite_word_labels('data/ultrasuite/labels-uxtd-uxssd-upx/', 'uxtd
```

In [32]:

```
# Preview the data
uxtd_df.head()
```

Out[32]:

	start_time	end_time	utterance	dataset	speaker	session	speech_waveform
0	00:00:07.300000	00:00:08.180000	watch	uxtd	37M	None	001A
1	00:00:08.270000	00:00:09.219999	fishing	uxtd	37M	None	001A
2	00:00:09.539999	00:00:10.500000	gloves	uxtd	37M	None	001A
3	00:00:10.640000	00:00:11.520000	spider	uxtd	37M	None	001A
4	00:00:01.170000	00:00:02.020000	r	uxtd	30F	None	010B

In [33]:

```
# Preview the dataframe info
uxtd_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6094 entries, 0 to 6093
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   start_time       6094 non-null   timedelta64[ns]
 1   end_time         6094 non-null   timedelta64[ns]
 2   utterance        6094 non-null   object
 3   dataset          6094 non-null   object
 4   speaker          6094 non-null   object
 5   session          0 non-null      object
 6   speech_waveform  6094 non-null   object
dtypes: object(5), timedelta64[ns](2)
memory usage: 333.4+ KB
```

In [34]:

```
# Load the labels for the Ultraphonix dataset
upx_df = all_ultrasuite_word_labels('data/ultrasuite/labels-uxtd-uxssd-upx/', 'upx')
```

In [35]:

```
# Preview the data
upx_df.head()
```

Out[35]:

	start_time	end_time	utterance	dataset	speaker	session	speech_waveform
0	00:00:00	00:00:01.400000	sigh	upx	20M	Post	012A
1	00:00:01.639999	00:00:02.910000	sausages	upx	20M	Post	012A
2	00:00:03.140000	00:00:03.970000	snail	upx	20M	Post	012A
3	00:00:04.890000	00:00:05.699999	beige	upx	20M	Post	012A
4	00:00:00.510000	00:00:01.240000	sack	upx	16M	BL3	016A

2C. Preprocessing - Stage One

Transforming the Ultrasuite dataset

Unlike the Speech Commands dataset, each audio sample in it's raw format contains multiple utterances that are spoken by both the Speech Therapist and the child subject.

The labels for each `.wav` file have been stored in a separate `.lab` file together with timestamps for the start and end of each utterance. The following is an example for the audio clip previewed earlier:

```
54700000 60900000 CAR
88800000 94800000 GIRL
109500000 112999999 MOON
126100000 136400000 KNIFE
```

In order to get the audio samples from the Ultrasuite datasets into the appropriate format for the Deep Learning models, we will need to splice the raw audio samples according to these timestamps.

In [61]:

```
def ultrasuite_word_labels(src_dataset, src_file):
    '''
    Extracts the labels from a single *.lab file into a single DataFrame

        Params:
            src_dataset (str): Dataset name of the target *.lab file
            src_file (str): Filename of the target *.lab file

        Returns:
            word_labels_df (pandas.core.frame.DataFrame):
            DataFrame containing the labels (utterances), timestamps, speaker and se
    '''
    filepath = 'data/ultrasuite/labels-uxtd-uxssd-upx/' + src_dataset + '/word_label

    columns = ['start_time', 'end_time', 'utterance']
    word_labels_df = pd.DataFrame()
    word_labels_df = pd.read_csv(filepath, sep=" ", header=None, names=columns)

    # Extract the speaker, session and speech data from the filename and add to the
    word_labels_df['dataset'] = src_dataset
    word_labels_df['speaker'] = src_file[0:3]
    if len(src_file[4:-9]) == 0:
        word_labels_df['session'] = None
    else:
        word_labels_df['session'] = src_file[4:-9]
    word_labels_df['speech_waveform'] = src_file[-8:-4]

    # Tidy up data formatting and correct time based units
    word_labels_df['utterance'] = word_labels_df['utterance'].str.lower()
    word_labels_df['start_time'] = pd.to_timedelta(word_labels_df['start_time'] * 10
    word_labels_df['end_time'] = pd.to_timedelta(word_labels_df['end_time'] * 100)

    return word_labels_df
```

In [62]:

```
# Quick test to check function works for a single labels file
upx_01F_df = ultrasuite_word_labels('upx', '01F-BL1-005A.lab')
upx_01F_df.head()
```

Out[62]:

	start_time	end_time	utterance	dataset	speaker	session	speech_waveform
0	00:00:00	00:00:00.620000	teeth	upx	01F	BL1	005A
1	00:00:03.860000	00:00:04.650000	watch	upx	01F	BL1	005A
2	00:00:06.160000	00:00:06.780000	orange	upx	01F	BL1	005A
3	00:00:09.050000	00:00:09.980000	school	upx	01F	BL1	005A

In [38]:

```
def extract_segments(y, sr, segments, dataset):
    '''
    Extracts audio segments from the source *.wav file based on timestamps containe

        Params:
            y (str): Path to input file
            sr (int): Sample Rate
            segments (DataFrame): DataFrame containing timestamps, labels, speaker a
            dataset (str): Specific ultrasuite dataset to process can be 'upx', 'uxt
    '''
    # Compute segment regions in number of samples
    starts = np.floor(segments.start_time.dt.total_seconds() * sr).astype(int)
    ends = np.ceil(segments.end_time.dt.total_seconds() * sr).astype(int)

    isolated_directory = 'data/ultrasuite_isolated/' + dataset + '/'

    if not os.path.isdir(isolated_directory):
        os.makedirs(isolated_directory.strip('/'))

    i = 0
    # Slice the audio into segments
    for start, end in zip(starts, ends):
        audio_seg = y[start:end]
        print('extracting audio segment:', len(audio_seg), 'samples')

        # Set the file path for the spliced audio file
        file_path = isolated_directory + str(segments.speaker[i]) + '/'
        if segments.session[i] != None:
            file_path = file_path + str(segments.session[i]) + '/'
        file_path = file_path + str(segments.speech_waveform[i]) + '/'

        if not os.path.isdir(file_path):
            os.makedirs(file_path.strip('/'))

        file_name = file_path + str(segments.utterance[i]) + '.wav'

        sf.write(file_name, audio_seg, sr)
        i += 1
```

In [39]:

```
def process_ultrasuite_wav_files(src_dataset, src_speaker, src_session):
    '''
    Processes and extracts audio segments for all Ultrasuite *.wav files

        Params:
            src_dataset (str): Ultrasuite dataset to process can be 'upx', 'uxtd' or
            src_speaker (str): Speaker to process
            src_session (str): Session to process

    '''
    directory = 'data/ultrasuite/core-' + src_dataset + '/core/' + src_speaker + '/

    # Set the target directory based on session if available
    if src_session != False:
         directory = directory + src_session + '/'

    # Loop through files in the directory, splice and rename files based on labels
    for filename in os.listdir(directory):

        if not filename[-5:-4] == 'E' or filename[-5:-4] == 'D':
            # Fetch the corresponding word labels and load into a DataFrame
            # Handle errors for when no labels exist
            # Files are graded on basis of quality and labels only available for hig
            try:
                if src_session != False:
                    labels_filename = src_speaker + '-' + src_session + '-' + filena
                else:
                    labels_filename = src_speaker + '-' + filename[-8:-4] + '.lab'

                labels_df = ultrasuite_word_labels(src_dataset, labels_filename)

                wav_path = directory + filename
                y, sr = librosa.load(wav_path, sr=16000)
                extract_segments(y, sr, labels_df, src_dataset)

            except IOError:
                if src_session != False:
                    print('\n' + src_speaker + '-' + src_session + '-' + filename[-8
                else:
                    print('\n' + src_speaker + '-' + filename[-8:-4] + '.lab not fou
```

In [40]:

```
def process_all_wav_files(datasets):
    '''
    Processes and extracts audio segments for all Ultrasuite *.wav files

        Params:
            datasets (list): Ultrasuite dataset to process can be any or all of 'upx
    '''
    # Loop through the datasets
    for dataset in datasets:
        current_dataset_dir = 'data/ultrasuite/core-' + dataset + '/core/'
        speakers = os.listdir(current_dataset_dir)

        # Loop through the speakers
        for speaker in speakers:
            current_speaker_dir = 'data/ultrasuite/core-' + dataset + '/core/' + spe
            sessions = os.listdir(current_speaker_dir)

            # If there are multiple therapy sessions, loop through the sessions and
            for session in sessions:
                if os.path.isdir(os.path.join(current_speaker_dir, session)):
                    process_ultrasuite_wav_files(dataset, speaker, session)
                else:
                    process_ultrasuite_wav_files(dataset, speaker, False)
```

In [ ]:

```
# Splice all *.wav files for all datasets
# NOTE: This takes a long time to run
process_datasets = ['upx', 'uxssd', 'uxtd']
process_all_wav_files(process_datasets)
```

In [14]:

```
def pad_silence(target_length, input_filepath, output_filepath):
    '''
    Pad the spliced audio samples with silence so that they are all at least 1 secor

        Params:
            target_length (int): Target length of final audio sample in milliseconds
            input_filepath (str): File path to input / original *.wav file
            output_filepath (str): File path to output / padded *.wav file
    '''
    target_length = target_length
    audio = AudioSegment.from_wav(input_filepath)
    if len(audio) > target_length:
        print(str(input_filepath) , 'is longer that 1 second, no padding required.')
        silence = AudioSegment.silent(duration=0)
    else:
        silence = AudioSegment.silent(duration=target_length - len(audio) + 1)

    padded = audio + silence
    padded.export(output_filepath, format='wav')
```

In [42]:

```
def standardise_filing(datasets):
    '''
    Standardise filing structure for isolated samples, padding and renaming files ir

        Params:
            datasets (list): Ultrasuite dataset to process can be any or all of 'upx
    '''
    # Loop through the datasets
    for dataset in datasets:

        isolated_files = Path.cwd() / 'data/ultrasuite_isolated' / dataset

        for isolated_file in isolated_files.glob('**/*'):

            if isolated_file.is_file():

                filename = isolated_file.stem
                extension = isolated_file.suffix
                sourcedata = dataset
                sourcefile = isolated_file.parent.parts[-1]

                # Rename the file but don't lose the original references handling tl
                if dataset == 'uxtd':
                    speaker = isolated_file.parent.parts[-2]
                    new_filename = f'{filename}_{dataset}-{speaker}-{sourcefile}{ext

                else:
                    session = isolated_file.parent.parts[-2]
                    speaker = isolated_file.parent.parts[-3]
                    new_filename = f'{filename}_{dataset}-{speaker}-{session}-{sourc

                # Define the new file path and create directory if it doesn't exist
                new_path = Path.cwd() / 'data/ultrasuite_transformed' / filename

                if not new_path.exists():
                    new_path.mkdir(parents=True, exist_ok=True)

                new_file_path = new_path.joinpath(new_filename)

                # Pad audio sample if required and move to new location
                if extension == '.wav':
                    pad_silence(1000, str(isolated_file), str(new_file_path))
```

In [ ]:

```
# Run the function to standardise the filing for all Ultrasuite datasets
standardise_filing(['upx', 'uxssd', 'uxtd'])
```

Cleanse the Ultrasuite dataset

1. Only keep audio samples of actual words using NLTK WordNet as a source corpus
2. Remove audio samples of simple phonetic letters (from the `manual_remove` list)
3. Only keep audio samples that have more than 5 different samples

In [43]:

```
# Check to see if the WordNet corpus is available, download if not and import
try:
    nltk.data.find('corpora/wordnet')
except LookupError:
    nltk.download('wordnet')

from nltk.corpus import wordnet as wn

def remove_invalid_samples():
    '''
    Function to remove all 'invalid' audio samples based on predetermined criteria
    '''
    transformed_files = 'data/ultrasuite_transformed/'

    manual_remove = ['a']

    for name in sorted(os.listdir(transformed_files)):

        path = os.path.join(transformed_files, name)

        if os.path.isdir(path):
            num_samples = len(os.listdir(path))

            # Remove audio samples of words not listed in NLTK WordNet corpus
            if not wn.synsets(name) or len(name)==1:
                print(name, 'is NOT a valid word, removing', num_samples, 'samples')
                shutil.rmtree(path)
            # Remove audio samples where there are 5 or less samples
            elif num_samples <= 5:
                print(name, 'does NOT have enough samples, removing', num_samples,
                shutil.rmtree(path)
            # Remove audio samples based on our manually constructed list above
            elif name in manual_remove:
                print(name, 'is being manually removed', num_samples, 'samples')
                shutil.rmtree(path)
            else:
                print('---')
                print(name, 'is a valid word and there are', num_samples, 'samples:
```

In [ ]:

```
# Remove invalid audio samples from the transformed dataset
remove_invalid_samples()
```

In [8]:

```
# Get the audio sample file information for the Ultrasuite dataset
ultrasuite_filestats = get_filestats('data/ultrasuite_transformed')
ultrasuite_filestats.head()
```

Out[8]:

	sample_utterance	sample_filename	sample_duration	sample_samplerate
0	parch	parch_upx-05M-BL2-017A.wav	1.000938	16000
1	parch	parch_upx-05M-Mid-016A.wav	1.001000	16000
2	parch	parch_upx-05M-BL3-016A.wav	1.000875	16000
3	parch	parch_upx-05M-BL4-016A.wav	1.000875	16000
4	parch	parch_upx-05M-Maint-016A.wav	1.000875	16000

In [45]:

```
# Get the file information the audio samples for 'book'
ultrasuite_book = ultrasuite_filestats[(ultrasuite_filestats['sample_utterance'] ==
len(ultrasuite_book)
```

Out[45]:

```
114
```

In [46]:

```
# Preview the dataframe
ultrasuite_book.head(20)
```

Out[46]:

	sample_utterance	sample_filename	sample_duration	sample_samplerate
22389	book	book_uxssd-07F-Post-015A.wav	1.001000	16000
22390	book	book_uxtd-13F-045A.wav	1.000938	16000
22391	book	book_uxssd-06M-Mid-011A.wav	1.000938	16000
22392	book	book_uxssd-02M-Maint2-013A.wav	1.000938	16000
22393	book	book_uxssd-06M-Post-047A.wav	1.001000	16000
22394	book	book_uxssd-05M-Post-020A.wav	1.000938	16000
22395	book	book_upx-08M-Suit-014A.wav	1.000875	16000
22396	book	book_uxtd-16F-046A.wav	1.000938	16000
22397	book	book_upx-03F-Therapy_04-011A.wav	1.000938	16000
22398	book	book_uxssd-02M-BL2-027A.wav	1.001000	16000
22399	book	book_uxssd-04M-Mid-013A.wav	1.000875	16000
22400	book	book_uxssd-04M-BL2-013A.wav	1.001000	16000
22401	book	book_upx-15M-Post-052A.wav	1.000875	16000
22402	book	book_uxtd-23F-046A.wav	1.000938	16000
22403	book	book_uxssd-03F-BL1-027A.wav	1.000875	16000
22404	book	book_uxssd-07F-BL2-014A.wav	1.000938	16000
22405	book	book_uxssd-07F-Mid-014A.wav	1.001000	16000
22406	book	book_uxtd-25M-046A.wav	1.000938	16000
22407	book	book_upx-07M-Suit-015A.wav	1.000875	16000
22408	book	book_uxssd-06M-Maint1-041A.wav	1.000938	16000

In [9]:

```
# Check the total number of samples in the Ultrasuite dataset after preprocessing
len(ultrasuite_filestats)
```

Out[9]:

```
33800
```

In [10]:

```
# Check how many samples that are longer than 1 second in duration are in the datase
us_long_samples = ultrasuite_filestats[(ultrasuite_filestats['sample_duration'] > 1.
len(us_long_samples)
```

Out[10]:

```
33800
```

In [11]:

```
# Summarise the number of samples for each utterance
us_summary = (ultrasuite_filestats.groupby(['sample_utterance'])
                                  .size()
                                  .reset_index(name='count')
                                  .sort_values('count', ascending=False))
us_summary.head(35)
```

Out[11]:

	sample_utterance	count
366	helicopter	292
700	say	290
961	watch	235
249	elephant	233
322	got	229
705	scissors	222
946	umbrella	222
274	fishing	222
814	spider	217
397	in	211
314	gloves	210
892	thank	204
84	bridge	198
290	frog	178
958	was	171
744	sheep	166
980	yellow	163
323	gown	162
237	ear	159
538	on	154
75	boy	148
282	four	146
412	ken	143
542	or	142
704	school	142
984	zebra	141
908	times	135
505	monkey	135
906	tiger	133
548	pack	132

	sample_utterance	count
275	five	130
884	teeth	128
905	tie	123
106	cab	123
176	crab	122

In [17]:

```
len(us_summary)
```

Out[17]:

```
991
```

In [12]:

```
# Get the top 35 words with the largest number of samples
us_top35 = us_summary.head(35)
us_top35.sort_values('sample_utterance', ascending=True)
```

Out[12]:

	sample_utterance	count
75	boy	148
84	bridge	198
106	cab	123
176	crab	122
237	ear	159
249	elephant	233
274	fishing	222
275	five	130
282	four	146
290	frog	178
314	gloves	210
322	got	229
323	gown	162
366	helicopter	292
397	in	211
412	ken	143
505	monkey	135
538	on	154
542	or	142
548	pack	132
700	say	290
704	school	142
705	scissors	222
744	sheep	166
814	spider	217
884	teeth	128
892	thank	204
905	tie	123
906	tiger	133
908	times	135
946	umbrella	222
958	was	171

	sample_utterance	count
961	watch	235
980	yellow	163
984	zebra	141

In [51]:

```
# Get the audio sample file information for the Speech Commands dataset
speechcommands_filestats = get_filestats('data/speech_commands_v0.02')
speechcommands_filestats.head()
```

Out[51]:

	sample_speaker	sample_filename	sample_duration	sample_samplerate
0	right	8e523821_nohash_2.wav	1.000000	16000
1	right	bb05582b_nohash_3.wav	1.000000	16000
2	right	988e2f9a_nohash_0.wav	1.000000	16000
3	right	a69b9b3e_nohash_0.wav	0.938625	16000
4	right	1eddce1d_nohash_3.wav	1.000000	16000

In [52]:

```
len(speechcommands_filestats)
```

Out[52]:

```
105829
```

In [53]:

```
# Check how many samples that are longer than 1 second in duration are in the datase
sc_long_samples = speechcommands_filestats[(speechcommands_filestats['sample_duratio
len(sc_long_samples)
```

Out[53]:

```
0
```

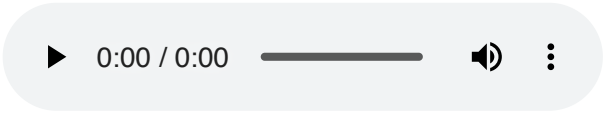
Preview audio samples, waveforms, spectrograms and labels from Ultrasuite dataset post transformation

In [19]:

```
# Load the audio sample and preview post transformation
target_sample_isolated = 'data/ultrasuite_isolated/uxssd/06M/BL1/002A/girl.wav'
us_sample_isolated, sr = librosa.load(target_sample_isolated)
print('Audio sample: Girl')
ipd.Audio(us_sample_isolated, rate=sr)
```

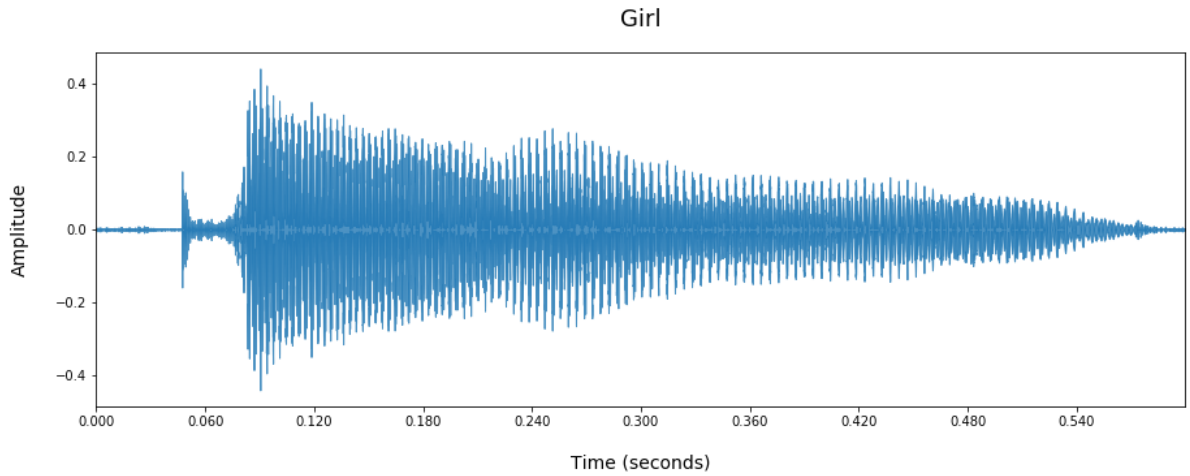
```
Audio sample: Girl
```

Out[19]:



In [20]:

```
# Plot the isolated waveform for the specific audio sample
plt.figure(figsize=(15, 5))
plt.title('Girl', fontsize=18, pad=20)
librosa.display.waveplot(us_sample_isolated, sr, alpha=0.8)
plt.xlabel('Time (seconds)', fontsize=14, labelpad=20)
plt.ylabel('Amplitude', fontsize=14, labelpad=20)
plt.show();
```

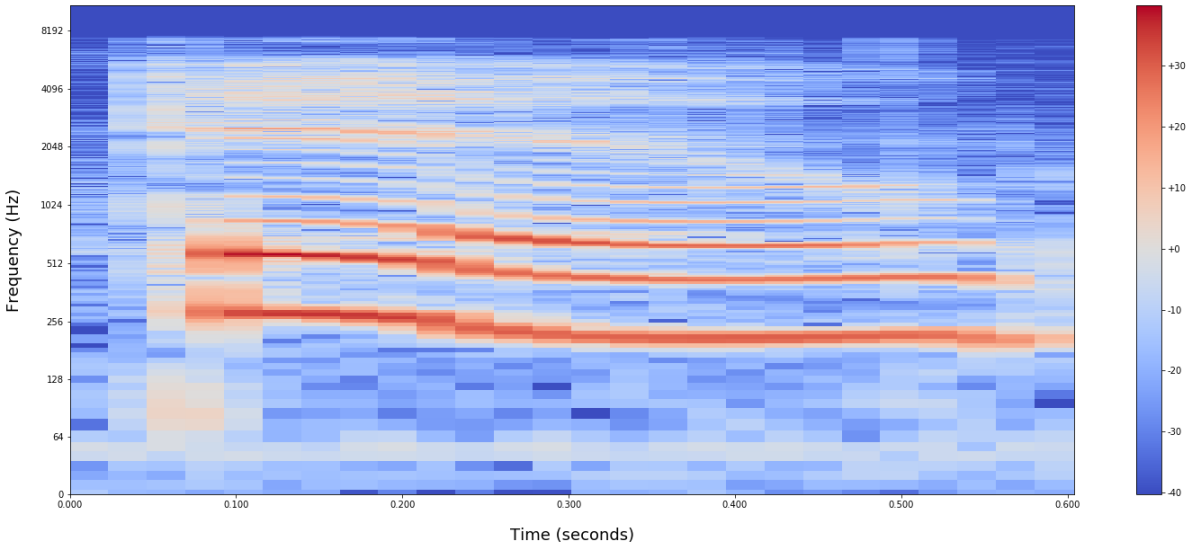


In [38]:

```
S_us_sample_iso = librosa.stft(us_sample_isolated, n_fft=n_fft, hop_length=hop_lengt
Y_us_sample_iso = np.abs(S_us_sample_iso) ** 2

Y_log_us_sample_iso = librosa.power_to_db(Y_us_sample_iso)

# Display spectrogram using log frequency
plot_spectrogram(Y_log_us_sample_iso, sr, hop_length, y_axis='log')
```



Sources / Code adapted from:

* Audio Signal Processing for ML - Valerio Velardo - The Sound of AI (https://github.com/musikalkemist/AudioSignalProcessingForML)