1. Introduction

Finally, some *cutie cute* competition involving animals, sounds, nature, earth and all that goodness .

This is a very new different challenge for me, and not just because it's mainly based on *audio files*, but because the rules are a bit different than what I'm used to. When I joined, I had (still have) some very big issues in understanding not only the *rules of submission*, but also the *data* and ... what all means?

So, as I go along I will try to bring some clear understanding and also point to some fruitful discussions. Ok, here we go!

Libraries ⬇

```
import os

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib.image as mpimg
from matplotlib.offsetbox import AnnotationBbox, OffsetImage

# Map 1 library
import plotly.express as px

# Map 2 libraries
import descartes
import geopandas as gpd
from shapely.geometry import Point, Polygon

# Librosa Libraries
import librosa
import librosa.display
import IPython.display as ipd

import sklearn

import warnings
warnings.filterwarnings('ignore')
```

2. The .csv files

Note:

- `train.csv` contains information about the audio files available in `train_audio`. It contains 21,375 datapoints in 35 unique columns.
- `test.csv` contains only 3 observations (the rest are available in the *hidden test set*).

Discussions

- Few questions about test data
- Confusion about test set

```
# Import data
train_csv = pd.read_csv("../input/birdsong-recognition/train.csv")
test_csv = pd.read_csv("../input/birdsong-recognition/test.csv")

# Create some time features
train_csv['year'] = train_csv['date'].apply(lambda x: x.split('-')[0])
train_csv['month'] = train_csv['date'].apply(lambda x: x.split('-')
[1])
train_csv['day_of_month'] = train_csv['date'].apply(lambda x:
x.split('-')[2])

print("There are {:,} unique bird species in the
dataset.".format(len(train_csv['species'].unique())))
```

```
There are 264 unique bird species in the dataset.
```

TEST.csv - let's take a look here as well before going further

Note:

- only 3 rows available (rest are in the hidden set)
- `site`: there are 3 sites in total, with first 2 having labeles every 5 seconds, while site_3 has labels at file level.
- `row_id`: this is the unique ID that will be used for the submission
- `seconds`: how long the clip is
- `audio_id`: `row_id` without site

PS: "nocall" can be also one of the labels (hearing no bird).

```
# Inspect text_csv before checking train data
test_csv
```

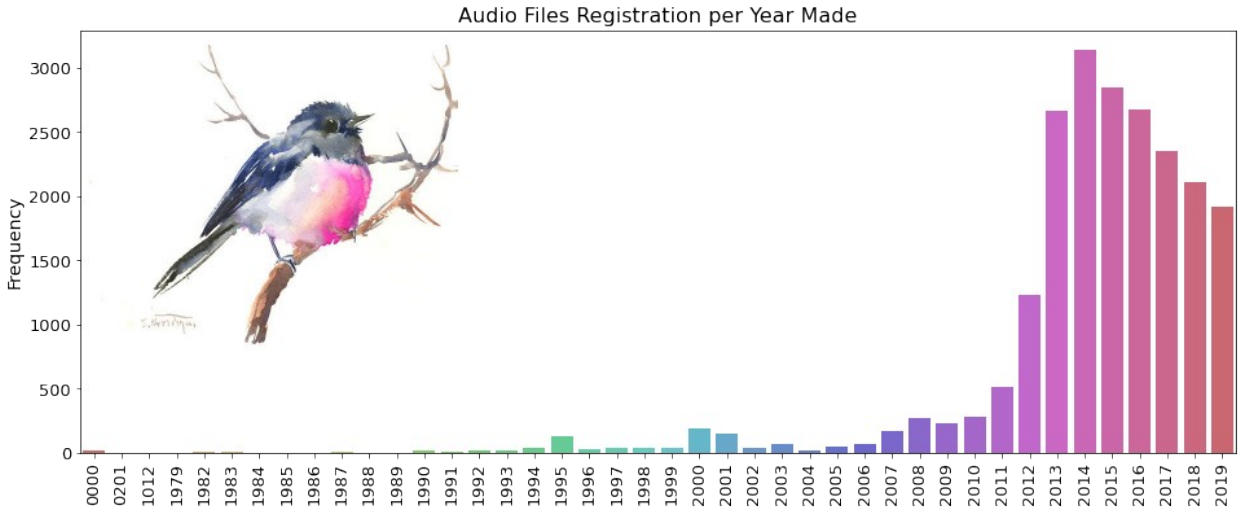
```
     site                                          row_id  seconds  \
0  site_1   site_1_0a997dff022e3ad9744d4e7bbf923288_5        5
1  site_1  site_1_0a997dff022e3ad9744d4e7bbf923288_10       10
2  site_1  site_1_0a997dff022e3ad9744d4e7bbf923288_15       15

                           audio_id
```

```
0  0a997dff022e3ad9744d4e7bbf923288
1  0a997dff022e3ad9744d4e7bbf923288
2  0a997dff022e3ad9744d4e7bbf923288
```

2.1 Time of the Recording

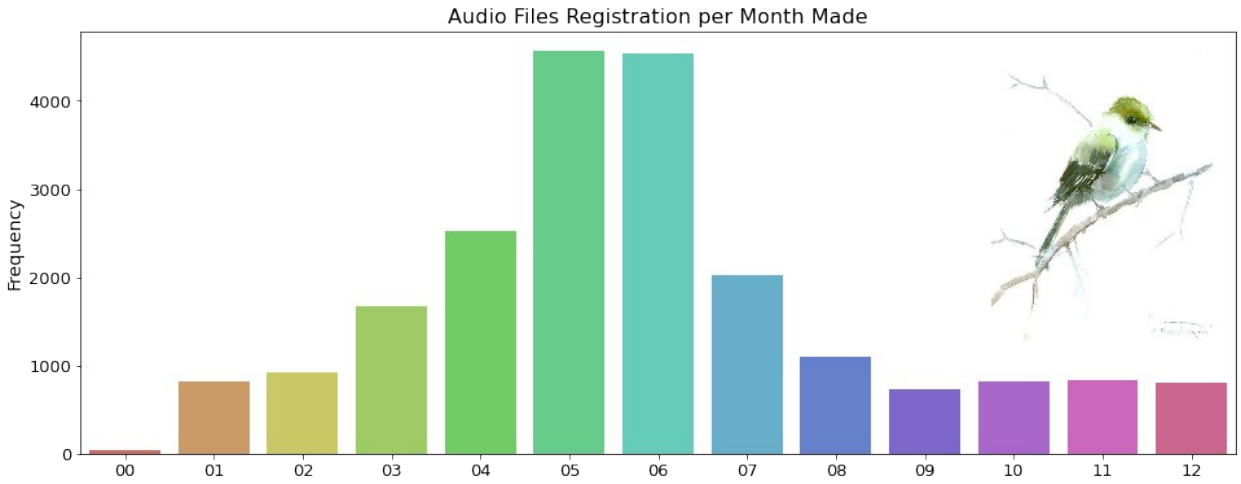
Note: Majority of the data was registered between 2013 and 2019, during Spring and Summer months (`00` is for the dates 0000-00-00, which are unknown).

```
bird = mpimg.imread('../input/birdcall-recognition-data/pink
bird.jpg')
imagebox = OffsetImage(bird, zoom=0.5)
xy = (0.5, 0.7)
ab = AnnotationBbox(imagebox, xy, frameon=False, pad=1, xybox=(6.5,
2000))

plt.figure(figsize=(16, 6))
ax = sns.countplot(train_csv['year'], palette="hls")
ax.add_artist(ab)

plt.title("Audio Files Registration per Year Made", fontsize=16)
plt.xticks(rotation=90, fontsize=13)
plt.yticks(fontsize=13)
plt.ylabel("Frequency", fontsize=14)
plt.xlabel("");
```




```
bird = mpimg.imread('../input/birdcall-recognition-data/green
bird.jpg')
imagebox = OffsetImage(bird, zoom=0.3)
xy = (0.5, 0.7)
ab = AnnotationBbox(imagebox, xy, frameon=False, pad=1, xybox=(11,
3000))
```

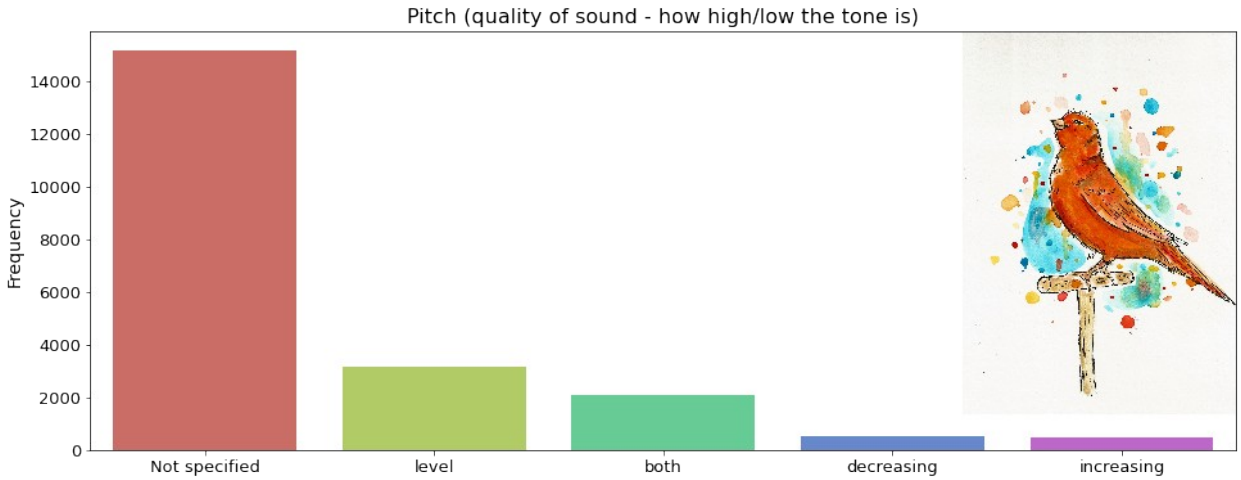
```
plt.figure(figsize=(16, 6))
ax = sns.countplot(train_csv['month'], palette="hls")
ax.add_artist(ab)

plt.title("Audio Files Registration per Month Made", fontsize=16)
plt.xticks(fontsize=13)
plt.yticks(fontsize=13)
plt.ylabel("Frequency", fontsize=14)
plt.xlabel("");
```



2.2 The Songs

Note: Pitch is usually unspecified. This is one of the more *miscellaneous* columns, that we need to be careful how we interpret. Most Song Types are *call, song or flight*.

```
bird =
mpimg.imread('../input/birdcall-recognition-data/orangebird.jpeg')
imagebox = OffsetImage(bird, zoom=0.12)
xy = (0.5, 0.7)
ab = AnnotationBbox(imagebox, xy, frameon=False, pad=1, xybox=(3.9,
8600))

plt.figure(figsize=(16, 6))
ax = sns.countplot(train_csv['pitch'], palette="hls", order =
train_csv['pitch'].value_counts().index)
ax.add_artist(ab)

plt.title("Pitch (quality of sound - how high/low the tone is)",
fontsize=16)
plt.xticks(fontsize=13)
plt.yticks(fontsize=13)
plt.ylabel("Frequency", fontsize=14)
plt.xlabel("");
```



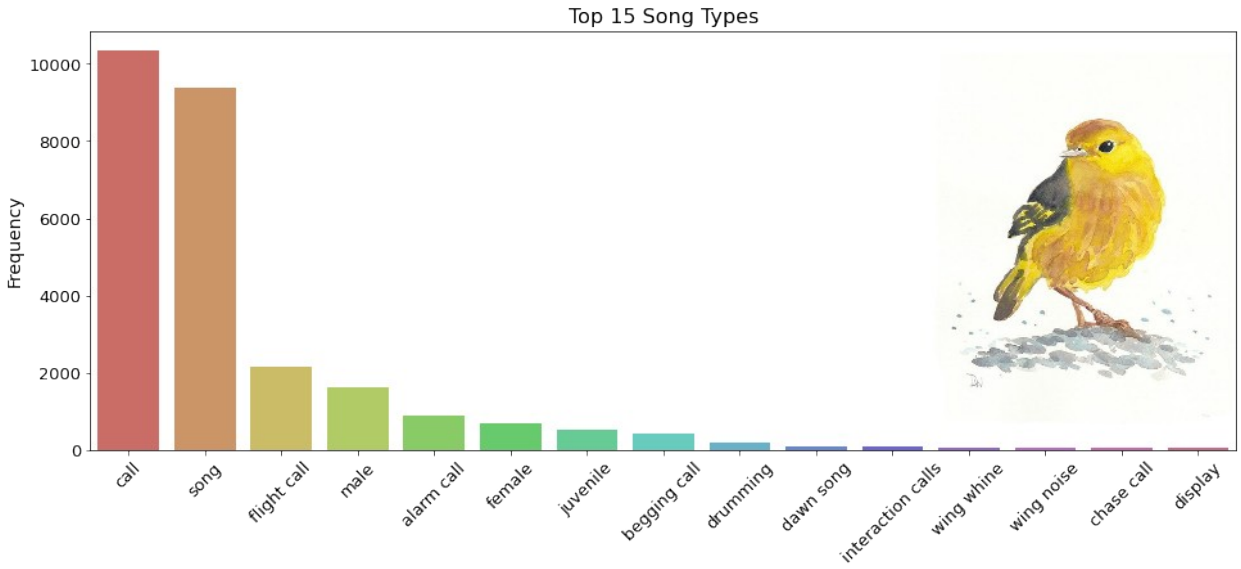
Type Column:

Note: This column is a bit messy, as the same description can be found under multiple names. Also, there can be multiple descriptions for multiple sounds (one bird song can mean a different thing from another one in the same recording). Some examples are:

- **alarm call** is: alarm call | alarm call, call
- **flight call** is: flight call | call, flight call etc.

```
# Create a new variable type by exploding all the values
adjusted_type = train_csv['type'].apply(lambda x:
x.split(',')).reset_index().explode("type")

# Strip of white spaces and convert to lower chars
adjusted_type = adjusted_type['type'].apply(lambda x:
x.strip().lower()).reset_index()
adjusted_type['type'] = adjusted_type['type'].replace('calls', 'call')

# Create Top 15 list with song types
top_15 =
list(adjusted_type['type'].value_counts().head(15).reset_index()
['index'])
data = adjusted_type[adjusted_type['type'].isin(top_15)]

# === PLOT ===
bird = mpimg.imread('../input/birdcall-recognition-data/Eastern
Meadowlark.jpg')
imagebox = OffsetImage(bird, zoom=0.43)
xy = (0.5, 0.7)
ab = AnnotationBbox(imagebox, xy, frameon=False, pad=1, xybox=(12.4,
5700))

plt.figure(figsize=(16, 6))
ax = sns.countplot(data['type'], palette="hls", order =
data['type'].value_counts().index)
```

```
ax.add_artist(ab)

plt.title("Top 15 Song Types", fontsize=16)
plt.ylabel("Frequency", fontsize=14)
plt.yticks(fontsize=13)
plt.xticks(rotation=45, fontsize=13)
plt.xlabel("");
```

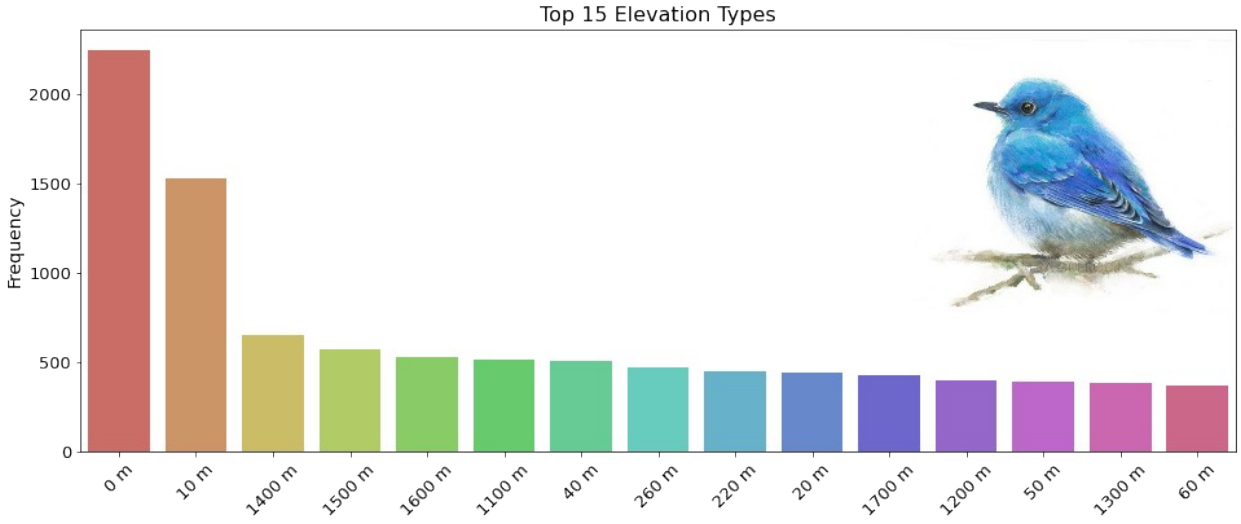


2.3 Where is the bird?

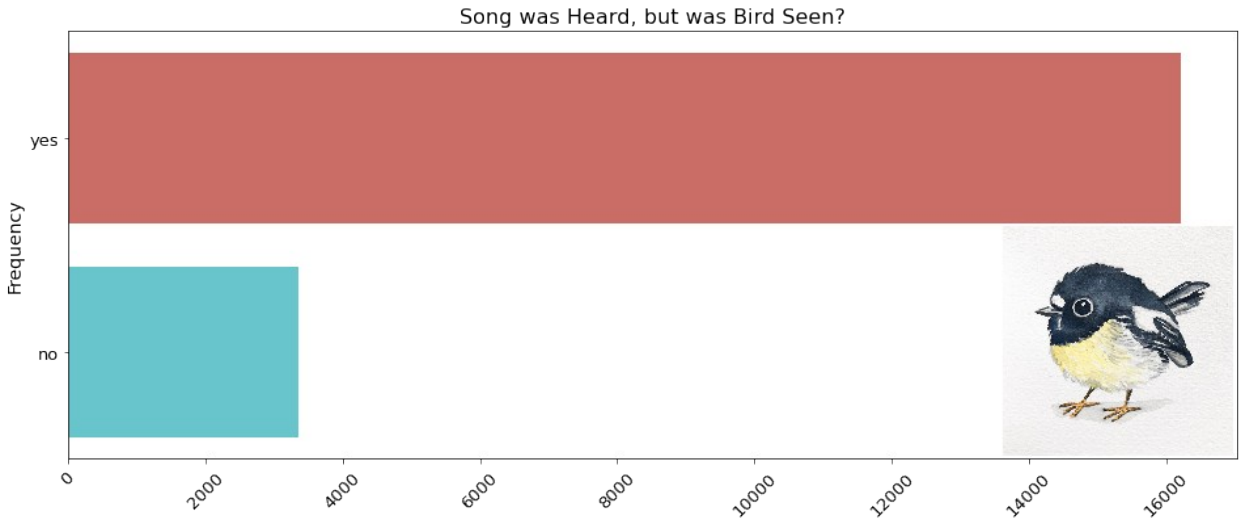
Note: In most recordings the birds were seen, usually at an altitude between 0m and 10m.

```
# Top 15 most common elevations
top_15 =
list(train_csv['elevation'].value_counts().head(15).reset_index()
['index'])
data = train_csv[train_csv['elevation'].isin(top_15)]

# === PLOT ===
bird = mpimg.imread('../input/birdcall-recognition-data/blue
bird.jpg')
imagebox = OffsetImage(bird, zoom=0.43)
xy = (0.5, 0.7)
ab = AnnotationBbox(imagebox, xy, frameon=False, pad=1, xybox=(12.4,
1450))

plt.figure(figsize=(16, 6))
ax = sns.countplot(data['elevation'], palette="hls", order =
data['elevation'].value_counts().index)
ax.add_artist(ab)
```

```
plt.title("Top 15 Elevation Types", fontsize=16)
plt.ylabel("Frequency", fontsize=14)
plt.yticks(fontsize=13)
plt.xticks(rotation=45, fontsize=13)
plt.xlabel("");
```




```
# Create data
data = train_csv['bird_seen'].value_counts().reset_index()

# === PLOT ===
bird = mpimg.imread('../input/birdcall-recognition-data/black
bird.jpg')
imagebox = OffsetImage(bird, zoom=0.22)
xy = (0.5, 0.7)
ab = AnnotationBbox(imagebox, xy, frameon=False, pad=1, xybox=(15300,
0.95))

plt.figure(figsize=(16, 6))
ax = sns.barplot(x = 'bird_seen', y = 'index', data = data,
palette="hls")
ax.add_artist(ab)

plt.title("Song was Heard, but was Bird Seen?", fontsize=16)
plt.ylabel("Frequency", fontsize=14)
plt.yticks(fontsize=13)
plt.xticks(rotation=45, fontsize=13)
plt.xlabel("");
```

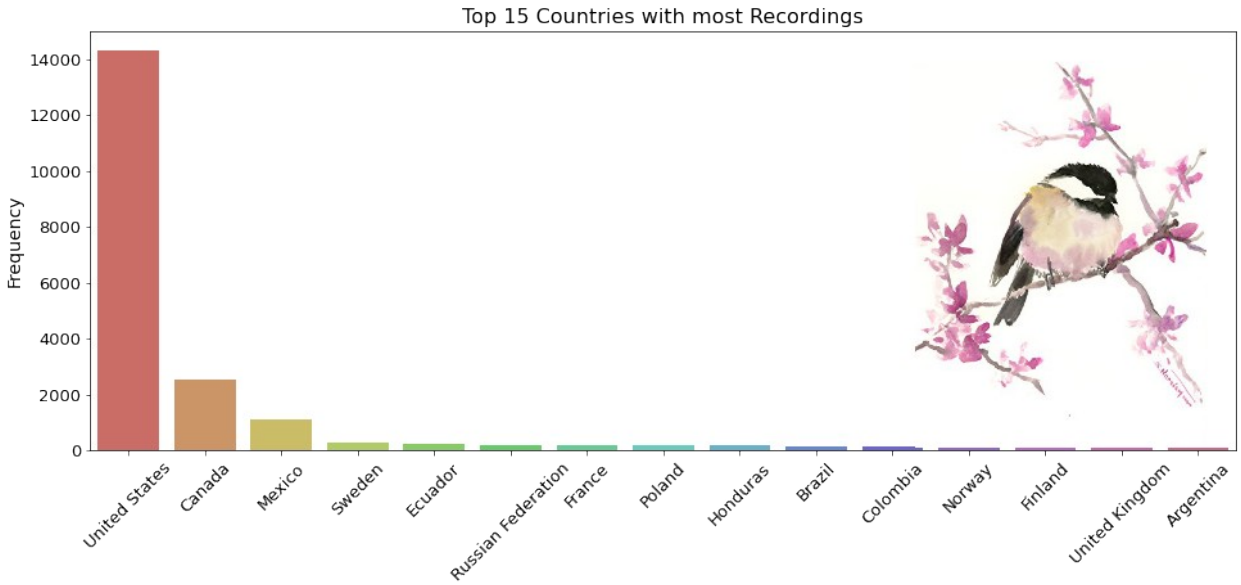


2.4 World View of the Species 🌎🌍

#1. Countries 🌐

🔎**Note**: Let's look at **top 15** countries with most recordings. The majority of recordings are located in the US, followed by Canada and Mexico.

```
# Top 15 most common elevations
top_15 =
list(train_csv['country'].value_counts().head(15).reset_index()
['index'])
data = train_csv[train_csv['country'].isin(top_15)]

# === PLOT ===
bird = mpimg.imread('../input/birdcall-recognition-data/fluff
ball.jpg')
imagebox = OffsetImage(bird, zoom=0.6)
xy = (0.5, 0.7)
ab = AnnotationBbox(imagebox, xy, frameon=False, pad=1, xybox=(12.2,
7000))

plt.figure(figsize=(16, 6))
ax = sns.countplot(data['country'], palette='hls', order =
data['country'].value_counts().index)
ax.add_artist(ab)

plt.title("Top 15 Countries with most Recordings", fontsize=16)
plt.ylabel("Frequency", fontsize=14)
plt.yticks(fontsize=13)
plt.xticks(rotation=45, fontsize=13)
plt.xlabel("");
```



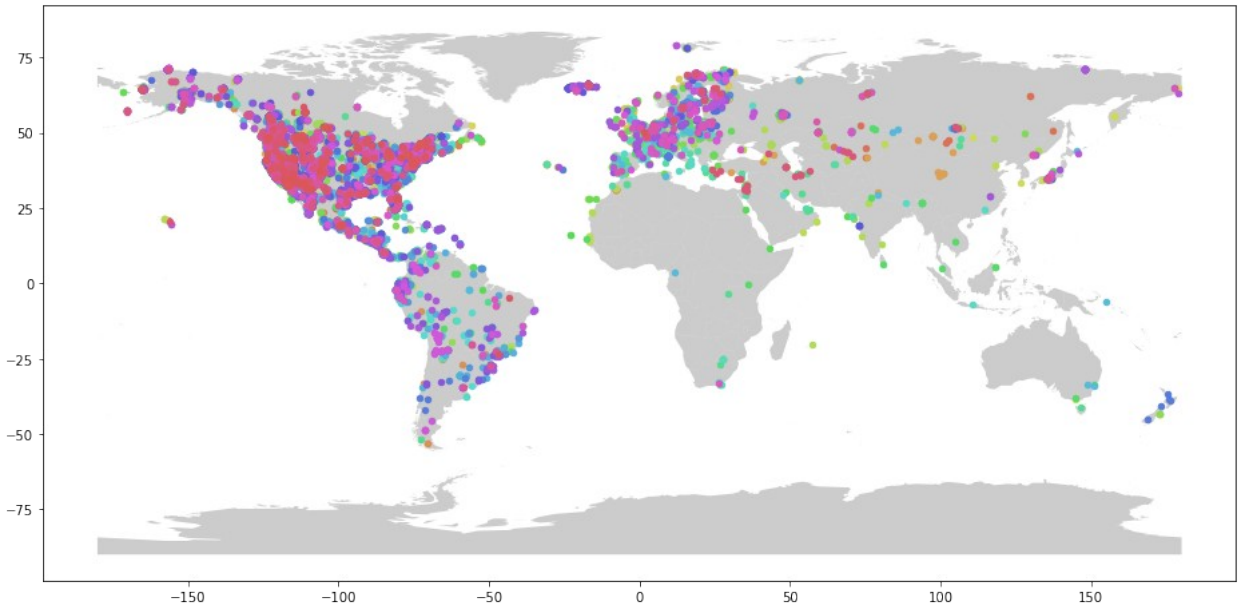
#2. Map View

```
# Import gapminder data, where we have country and iso ALPHA codes
df = px.data.gapminder().query("year==2007")[["country", "iso_alpha"]]

# Merge the tables together (we lose a fiew rows, but not many)
data = pd.merge(left=train_csv, right=df, how="inner", on="country")

# Group by country and count how many species can be found in each
data = data.groupby(by=["country", "iso_alpha"]).count()
["species"].reset_index()

fig = px.choropleth(data, locations="iso_alpha", color="species",
hover_name="country",
                    color_continuous_scale=px.colors.sequential.Teal,
                    title = "World Map: Recordings per Country")
fig.show()
```

#3. Another Map! 😁 Where are our birds?

```
# SHP file
world_map =
gpd.read_file("../input/world-shapefile/world_shapefile.shp")

# Coordinate reference system
crs = {"init" : "epsg:4326"}

# Lat and Long need to be of type float, not object
data = train_csv[train_csv["latitude"] != "Not specified"]
data["latitude"] = data["latitude"].astype(float)
data["longitude"] = data["longitude"].astype(float)
```

```
# Create geometry
geometry = [Point(xy) for xy in zip(data["longitude"],
data["latitude"])]

# Geo Dataframe
geo_df = gpd.GeoDataFrame(data, crs=crs, geometry=geometry)

# Create ID for species
species_id = geo_df["species"].value_counts().reset_index()
species_id.insert(0, 'ID', range(0, 0 + len(species_id)))

species_id.columns = ["ID", "species", "count"]

# Add ID to geo_df
geo_df = pd.merge(geo_df, species_id, how="left", on="species")

# === PLOT ===
fig, ax = plt.subplots(figsize = (16, 10))
world_map.plot(ax=ax, alpha=0.4, color="grey")

palette = iter(sns.hls_palette(len(species_id)))

for i in range(264):
    geo_df[geo_df["ID"] == i].plot(ax=ax, markersize=20,
color=next(palette), marker="o", label = "test");
```



3. The Audio Files

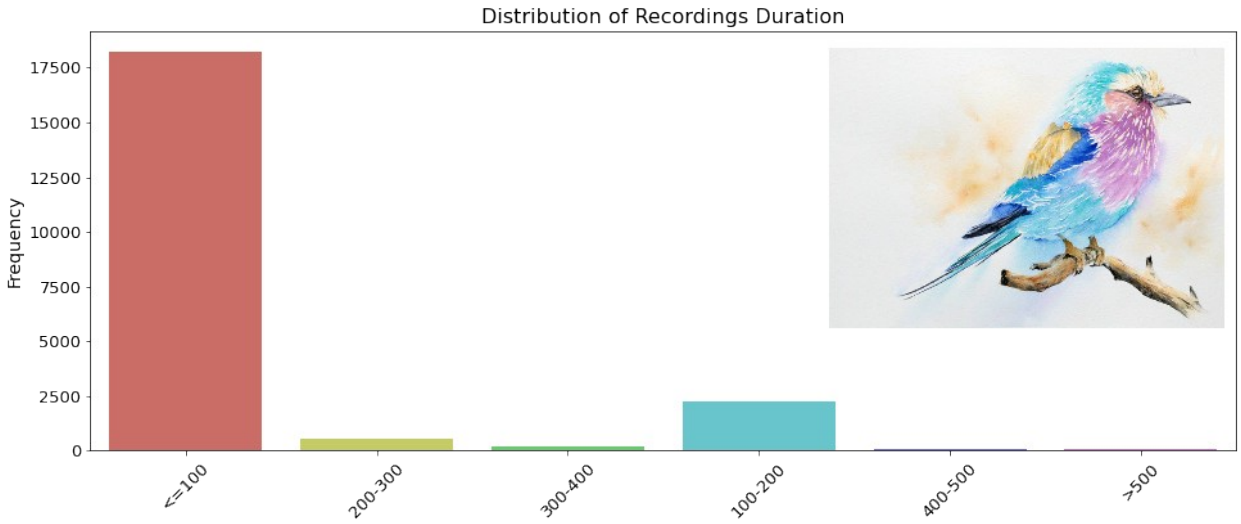
3.1 Description

Note:

- **train_audio**: short recording (majority in mp3 format) of INDIVIDUAL birds.
- **test_audio**: recordings took in 3 locations:
  - Site 1 and Site 2: recordings 10 mins long (mp3) that have labeled a bird every 5 seconds. This is meant to mimic the *real life scenario*, when you would usually have more than 1 bird (or no bird) singing.
  - Site 3: recordings labeled at file level (because it is especially hard to have coders trained to label these kind of files)

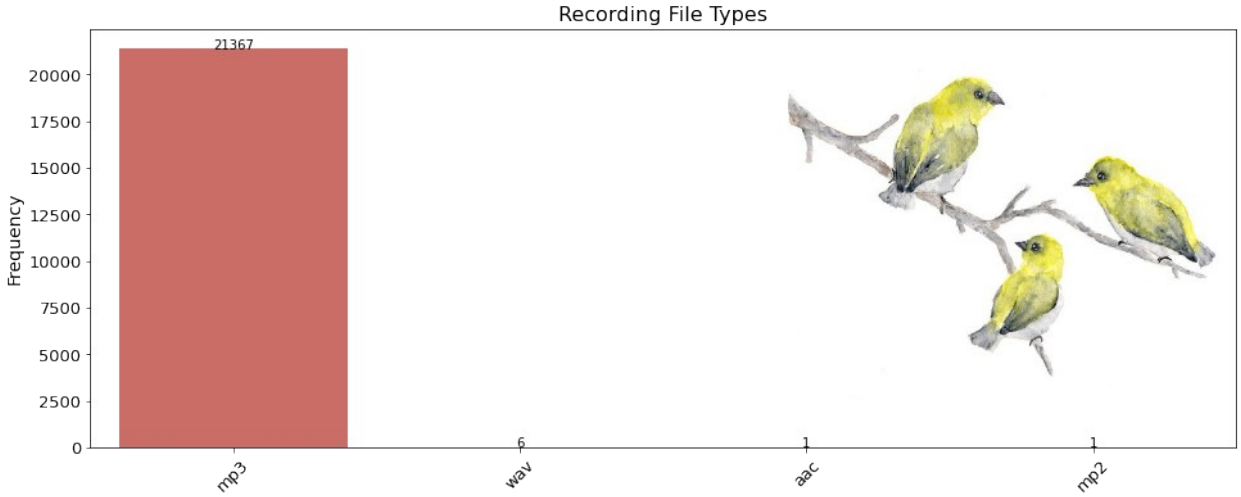
3.2 Duration and File Types

```
# Creating Interval for *duration* variable
train_csv['duration_interval'] = ">500"
train_csv.loc[train_csv['duration'] <= 100, 'duration_interval'] =
"<=100"
train_csv.loc[(train_csv['duration'] > 100) & (train_csv['duration']
<= 200), 'duration_interval'] = "100-200"
train_csv.loc[(train_csv['duration'] > 200) & (train_csv['duration']
<= 300), 'duration_interval'] = "200-300"
train_csv.loc[(train_csv['duration'] > 300) & (train_csv['duration']
<= 400), 'duration_interval'] = "300-400"
train_csv.loc[(train_csv['duration'] > 400) & (train_csv['duration']
<= 500), 'duration_interval'] = "400-500"

bird = mpimg.imread('../input/birdcall-recognition-data/multicolor
bird.jpg')
imagebox = OffsetImage(bird, zoom=0.4)
xy = (0.5, 0.7)
ab = AnnotationBbox(imagebox, xy, frameon=False, pad=1, xybox=(4.4,
12000))

plt.figure(figsize=(16, 6))
ax = sns.countplot(train_csv['duration_interval'], palette="hls")
ax.add_artist(ab)

plt.title("Distribution of Recordings Duration", fontsize=16)
plt.ylabel("Frequency", fontsize=14)
plt.yticks(fontsize=13)
plt.xticks(rotation=45, fontsize=13)
plt.xlabel("");
```




```
def show_values_on_bars(axs, h_v="v", space=0.4):
    def _show_on_single_plot(ax):
        if h_v == "v":
            for p in ax.patches:
                _x = p.get_x() + p.get_width() / 2
                _y = p.get_y() + p.get_height()
                value = int(p.get_height())
                ax.text(_x, _y, value, ha="center")
        elif h_v == "h":
            for p in ax.patches:
                _x = p.get_x() + p.get_width() + float(space)
                _y = p.get_y() + p.get_height()
                value = int(p.get_width())
                ax.text(_x, _y, value, ha="left")

    if isinstance(axs, np.ndarray):
        for idx, ax in np.ndenumerate(axs):
            _show_on_single_plot(ax)
    else:
        _show_on_single_plot(axs)

bird = mpimg.imread('../input/birdcall-recognition-data/yellow
birds.jpg')
imagebox = OffsetImage(bird, zoom=0.6)
xy = (0.5, 0.7)
ab = AnnotationBbox(imagebox, xy, frameon=False, pad=1, xybox=(2.7,
12000))

plt.figure(figsize=(16, 6))
ax = sns.countplot(train_csv['file_type'], palette = "hls", order =
train_csv['file_type'].value_counts().index)
ax.add_artist(ab)

show_values_on_bars(ax, "v", 0)
```

```
plt.title("Recording File Types", fontsize=16)
plt.ylabel("Frequency", fontsize=14)
plt.yticks(fontsize=13)
plt.xticks(rotation=45, fontsize=13)
plt.xlabel("");
```



3.3 Listening to some Recordings

Note: What is sound? In physics, sound is a vibration that propagates as an acoustic wave, through a transmission medium such as a gas, liquid or solid.

```
# Create Full Path so we can access data more easily
base_dir = '../input/birdsong-recognition/train_audio/'
train_csv['full_path'] = base_dir + train_csv['ebird_code'] + '/' +
train_csv['filename']

# Now let's sample a fiew audio files
amered = train_csv[train_csv['ebird_code'] == "amered"].sample(1,
random_state = 33)['full_path'].values[0]
cangoo = train_csv[train_csv['ebird_code'] == "cangoo"].sample(1,
random_state = 33)['full_path'].values[0]
haiwoo = train_csv[train_csv['ebird_code'] == "haiwoo"].sample(1,
random_state = 33)['full_path'].values[0]
pingro = train_csv[train_csv['ebird_code'] == "pingro"].sample(1,
random_state = 33)['full_path'].values[0]
vesspa = train_csv[train_csv['ebird_code'] == "vesspa"].sample(1,
random_state = 33)['full_path'].values[0]

bird_sample_list = ["amered", "cangoo", "haiwoo", "pingro", "vesspa"]
```

Ok, let's hear some songs! 🎵

```
# Amered
ipd.Audio(amered)
```

```
<IPython.lib.display.Audio object>
```

```
# Cangoo
ipd.Audio(cangoo)
```

```
<IPython.lib.display.Audio object>
```

```
# Haiwoo
ipd.Audio(haiwoo)
```

```
<IPython.lib.display.Audio object>
```

```
# Pingro
ipd.Audio(pingro)
```

```
<IPython.lib.display.Audio object>
```

```
# Vesspa
ipd.Audio(vesspa)
```

```
<IPython.lib.display.Audio object>
```

3.4 Extracting Features from Sounds

The audio data is composed by:

1. **Sound**: sequence of vibrations in varying pressure strengths (`y`)
2. **Sample Rate**: (`sr`) is the number of samples of audio carried per second, measured in Hz or kHz

```
# Importing 1 file
y, sr = librosa.load(vesspa)

print('y:', y, '\n')
print('y shape:', np.shape(y), '\n')
print('Sample Rate (KHz):', sr, '\n')

# Verify length of the audio
print('Check Len of Audio:', np.shape(y)[0]/sr)
```

```
y: [-0.00069803 -0.0005046   0.00079464 ...  0.0104561   0.00445888
  0.        ]

y shape: (1544955,)

Sample Rate (KHz): 22050

Check Len of Audio: 70.06598639455783
```

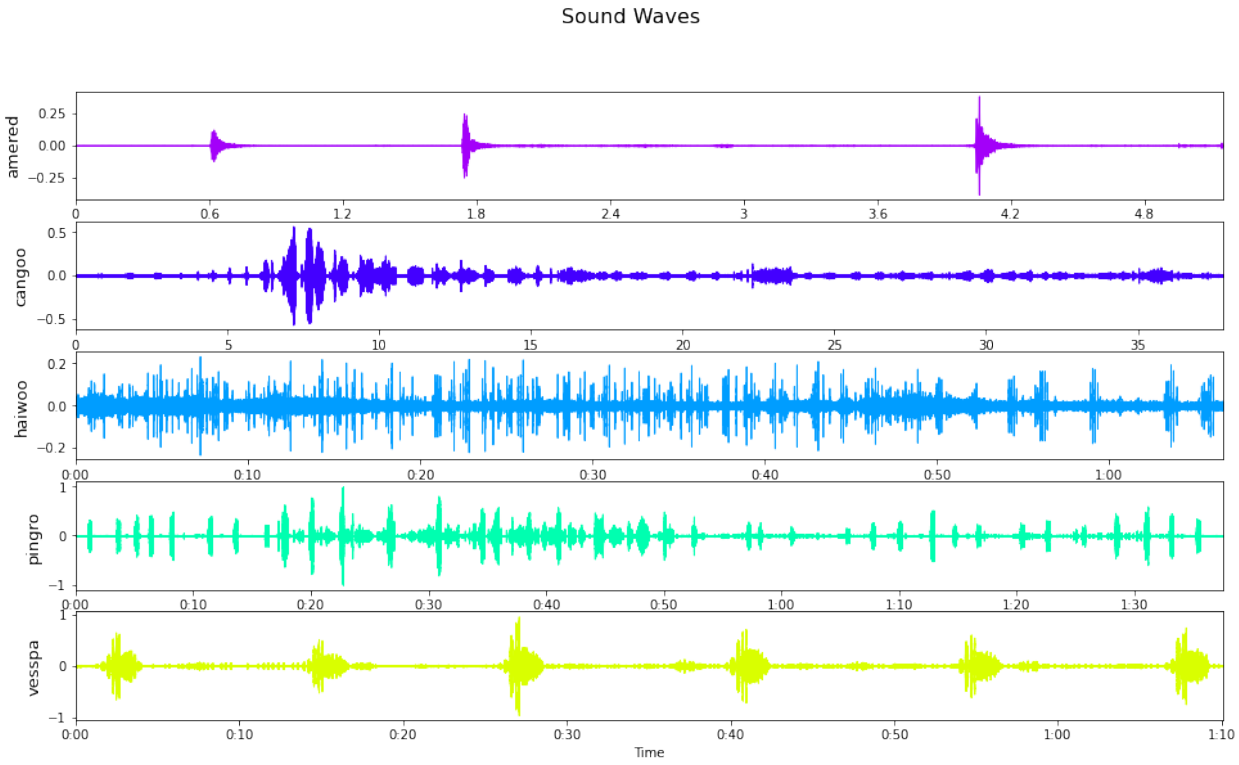
```
# Trim leading and trailing silence from an audio signal (silence
before and after the actual audio)
audio_file, _ = librosa.effects.trim(y)

# the result is an numpy ndarray
print('Audio File:', audio_file, '\n')
print('Audio File shape:', np.shape(audio_file))
```

```
Audio File: [-0.00069803 -0.0005046   0.00079464 ...  0.0104561
0.00445888
  0.        ]

Audio File shape: (1544955,)
```

```
# Importing the 5 files
y_amered, sr_amered = librosa.load(amered)
audio_amered, _ = librosa.effects.trim(y_amered)

y_cangoo, sr_cangoo = librosa.load(cangoo)
audio_cangoo, _ = librosa.effects.trim(y_cangoo)

y_haiwoo, sr_haiwoo = librosa.load(haiwoo)
audio_haiwoo, _ = librosa.effects.trim(y_haiwoo)

y_pingro, sr_pingro = librosa.load(pingro)
audio_pingro, _ = librosa.effects.trim(y_pingro)

y_vesspa, sr_vesspa = librosa.load(vesspa)
audio_vesspa, _ = librosa.effects.trim(y_vesspa)
```

#1. Sound Waves (2D Representation)

```
fig, ax = plt.subplots(5, figsize = (16, 9))
fig.suptitle('Sound Waves', fontsize=16)

librosa.display.waveplot(y = audio_amered, sr = sr_amered, color =
"#A300F9", ax=ax[0])
librosa.display.waveplot(y = audio_cangoo, sr = sr_cangoo, color =
"#4300FF", ax=ax[1])
librosa.display.waveplot(y = audio_haiwoo, sr = sr_haiwoo, color =
"#009DFF", ax=ax[2])
librosa.display.waveplot(y = audio_pingro, sr = sr_pingro, color =
"#00FFB0", ax=ax[3])
librosa.display.waveplot(y = audio_vesspa, sr = sr_vesspa, color =
"#D9FF00", ax=ax[4]);

for i, name in zip(range(5), bird_sample_list):
    ax[i].set_ylabel(name, fontsize=13)
```



#2. Fourier Transform

Note: Function that gets a signal in the time domain as input, and outputs its decomposition into frequencies. Transform both the y-axis (frequency) to log scale, and the “color” axis (amplitude) to Decibels, which is approx. the log scale of amplitudes.

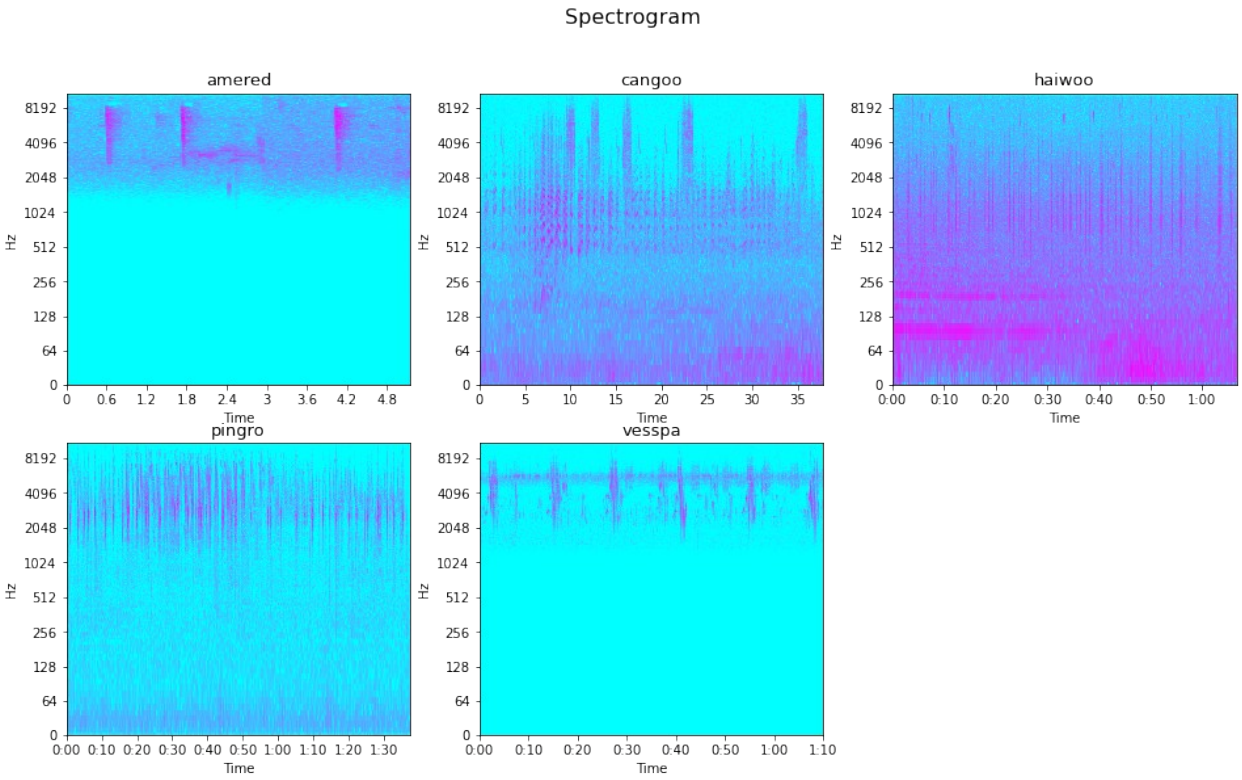
```
# Default FFT window size
n_fft = 2048 # FFT window size
hop_length = 512 # number audio of frames between STFT columns (looks
like a good default)

# Short-time Fourier transform (STFT)
D_amered = np.abs(librosa.stft(audio_amered, n_fft = n_fft, hop_length
= hop_length))
D_cangoo = np.abs(librosa.stft(audio_cangoo, n_fft = n_fft, hop_length
= hop_length))
D_haiwoo = np.abs(librosa.stft(audio_haiwoo, n_fft = n_fft, hop_length
= hop_length))
D_pingro = np.abs(librosa.stft(audio_pingro, n_fft = n_fft, hop_length
= hop_length))
D_vesspa = np.abs(librosa.stft(audio_vesspa, n_fft = n_fft, hop_length
= hop_length))

print('Shape of D object:', np.shape(D_amered))

Shape of D object: (1025, 222)
```

#3. Spectrogram

Note:

- What is a spectrogram? A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time. When applied to an audio signal, spectrograms are sometimes called sonographs, voiceprints, or voicegrams (wiki).
- Here we convert the frequency axis to a logarithmic one.

```
# Convert an amplitude spectrogram to Decibels-scaled spectrogram.
DB_amered = librosa.amplitude_to_db(D_amered, ref = np.max)
DB_cangoo = librosa.amplitude_to_db(D_cangoo, ref = np.max)
DB_haiwoo = librosa.amplitude_to_db(D_haiwoo, ref = np.max)
DB_pingro = librosa.amplitude_to_db(D_pingro, ref = np.max)
DB_vesspa = librosa.amplitude_to_db(D_vesspa, ref = np.max)

# === PLOT ===
fig, ax = plt.subplots(2, 3, figsize=(16, 9))
fig.suptitle('Spectrogram', fontsize=16)
fig.delaxes(ax[1, 2])

librosa.display.specshow(DB_amered, sr = sr_amered, hop_length =
hop_length, x_axis = 'time',
                         y_axis = 'log', cmap = 'cool', ax=ax[0, 0])
librosa.display.specshow(DB_cangoo, sr = sr_cangoo, hop_length =
hop_length, x_axis = 'time',
                         y_axis = 'log', cmap = 'cool', ax=ax[0, 1])
librosa.display.specshow(DB_haiwoo, sr = sr_haiwoo, hop_length =
hop_length, x_axis = 'time',
                         y_axis = 'log', cmap = 'cool', ax=ax[0, 2])
librosa.display.specshow(DB_pingro, sr = sr_pingro, hop_length =
hop_length, x_axis = 'time',
                         y_axis = 'log', cmap = 'cool', ax=ax[1, 0])
librosa.display.specshow(DB_vesspa, sr = sr_vesspa, hop_length =
hop_length, x_axis = 'time',
                         y_axis = 'log', cmap = 'cool', ax=ax[1, 1]);

for i, name in zip(range(0, 2*3), bird_sample_list):
    x = i // 3
    y = i % 3
    ax[x, y].set_title(name, fontsize=13)
```

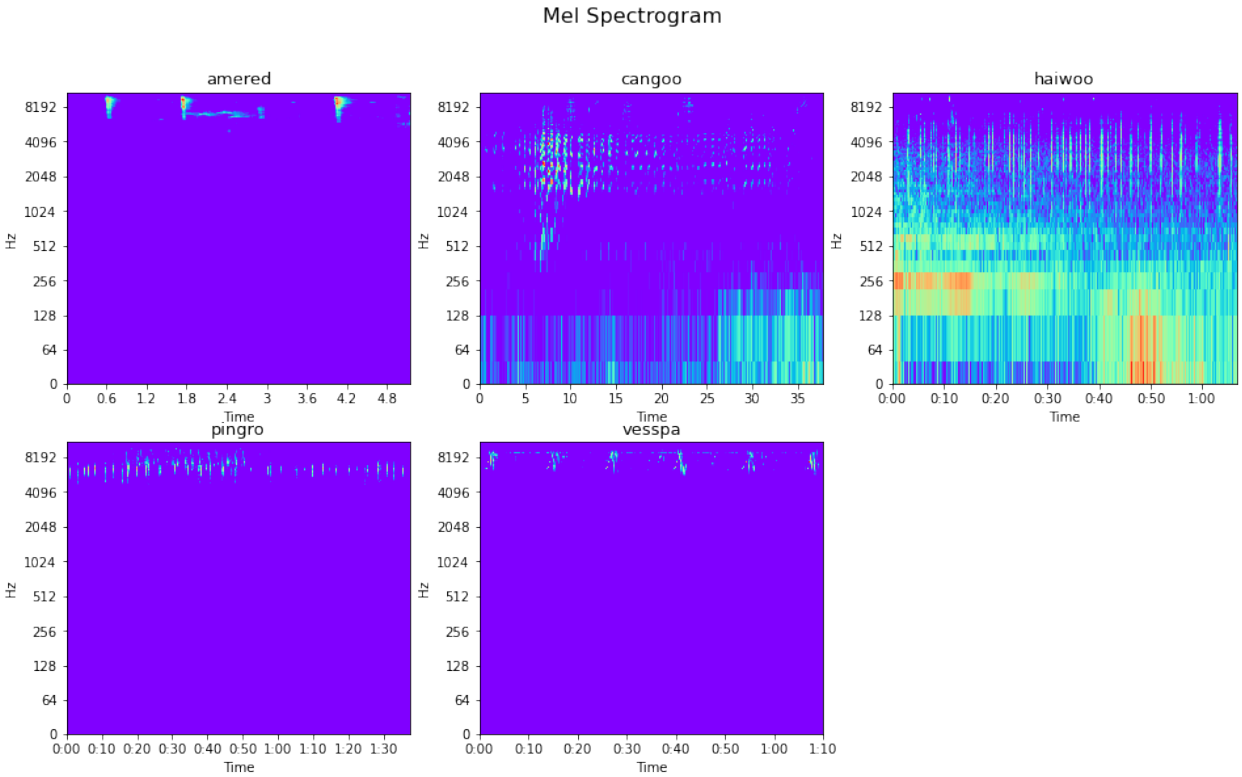


#4. Mel Spectrogram

Note: The Mel Scale, mathematically speaking, is the result of some non-linear transformation of the frequency scale. The Mel Spectrogram is a normal Spectrogram, but with a Mel Scale on the y axis.

```
# Create the Mel Spectrograms
S_amered = librosa.feature.melspectrogram(y_amered, sr=sr_amered)
S_DB_amered = librosa.amplitude_to_db(S_amered, ref=np.max)

S_cangoo = librosa.feature.melspectrogram(y_cangoo, sr=sr_cangoo)
S_DB_cangoo = librosa.amplitude_to_db(S_cangoo, ref=np.max)

S_haiwoo = librosa.feature.melspectrogram(y_haiwoo, sr=sr_haiwoo)
S_DB_haiwoo = librosa.amplitude_to_db(S_haiwoo, ref=np.max)

S_pingro = librosa.feature.melspectrogram(y_pingro, sr=sr_pingro)
S_DB_pingro = librosa.amplitude_to_db(S_pingro, ref=np.max)

S_vesspa = librosa.feature.melspectrogram(y_vesspa, sr=sr_vesspa)
S_DB_vesspa = librosa.amplitude_to_db(S_vesspa, ref=np.max)

# === PLOT ====
fig, ax = plt.subplots(2, 3, figsize=(16, 9))
fig.suptitle('Mel Spectrogram', fontsize=16)
fig.delaxes(ax[1, 2])
```

```
librosa.display.specshow(S_DB_amered, sr = sr_amered, hop_length =
hop_length, x_axis = 'time',
                         y_axis = 'log', cmap = 'rainbow', ax=ax[0,
0])
librosa.display.specshow(S_DB_cangoo, sr = sr_cangoo, hop_length =
hop_length, x_axis = 'time',
                         y_axis = 'log', cmap = 'rainbow', ax=ax[0,
1])
librosa.display.specshow(S_DB_haiwoo, sr = sr_haiwoo, hop_length =
hop_length, x_axis = 'time',
                         y_axis = 'log', cmap = 'rainbow', ax=ax[0,
2])
librosa.display.specshow(S_DB_pingro, sr = sr_pingro, hop_length =
hop_length, x_axis = 'time',
                         y_axis = 'log', cmap = 'rainbow', ax=ax[1,
0])
librosa.display.specshow(S_DB_vesspa, sr = sr_vesspa, hop_length =
hop_length, x_axis = 'time',
                         y_axis = 'log', cmap = 'rainbow', ax=ax[1,
1]);

for i, name in zip(range(0, 2*3), bird_sample_list):
    x = i // 3
    y = i % 3
    ax[x, y].set_title(name, fontsize=13)
```



#5. Zero Crossing Rate

Note: the rate at which the signal changes from positive to negative or back.

```
# Total zero_crossings in our 1 song
zero_amered = librosa.zero_crossings(audio_amered, pad=False)
zero_cangoo = librosa.zero_crossings(audio_cangoo, pad=False)
zero_haiwoo = librosa.zero_crossings(audio_haiwoo, pad=False)
zero_pingro = librosa.zero_crossings(audio_pingro, pad=False)
zero_vesspa = librosa.zero_crossings(audio_vesspa, pad=False)

zero_birds_list = [zero_amered, zero_cangoo, zero_haiwoo, zero_pingro,
zero_vesspa]

for bird, name in zip(zero_birds_list, bird_sample_list):
    print("{} change rate is {:,}".format(name, sum(bird)))
```

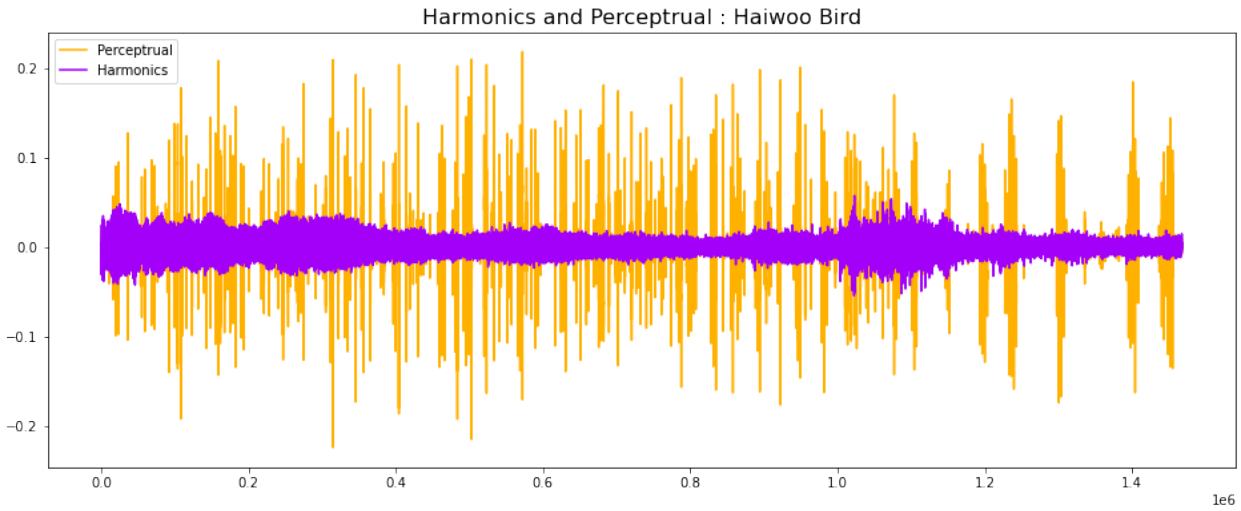
```
amered change rate is 51,379
cangoo change rate is 92,089
haiwoo change rate is 100,198
pingro change rate is 706,094
vesspa change rate is 678,007
```

#6. Harmonics and Perceptrual

Note:

- Harmonics are characteristichs that represent the sound *color*
- Perceptrual shock wave represents the sound *rhythm and emotion*

```
y_harm_haiwoo, y_perc_haiwoo = librosa.effects.hpss(audio_haiwoo)

plt.figure(figsize = (16, 6))
plt.plot(y_perc_haiwoo, color = '#FFB100')
plt.plot(y_harm_haiwoo, color = '#A300F9')
plt.legend(("Perceptrual", "Harmonics"))
plt.title("Harmonics and Perceptrual : Haiwoo Bird", fontsize=16);
```



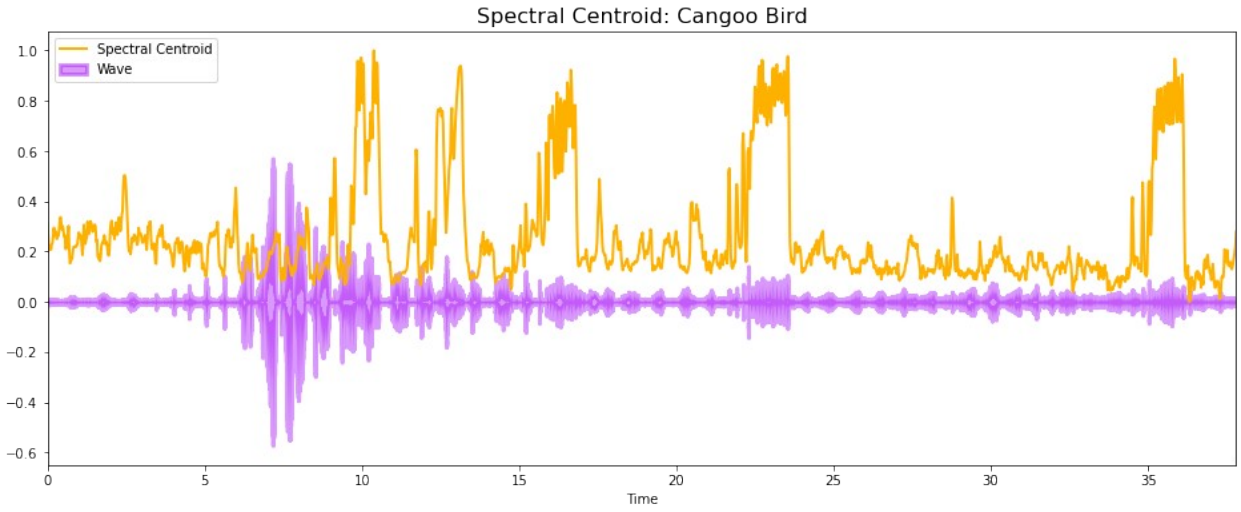
#7. Spectral Centroid

Note: Indicates where the "centre of mass" for a sound is located and is calculated as the weighted mean of the frequencies present in the sound.

```
# Calculate the Spectral Centroids
spectral_centroids = librosa.feature.spectral_centroid(audio_cangoo,
sr=sr)[0]

# Shape is a vector
print('Centroids:', spectral_centroids, '\n')
print('Shape of Spectral Centroids:', spectral_centroids.shape, '\n')

# Computing the time variable for visualization
frames = range(len(spectral_centroids))

# Converts frame counts to time (seconds)
t = librosa.frames_to_time(frames)

print('frames:', frames, '\n')
print('t:', t)

# Function that normalizes the Sound Data
def normalize(x, axis=0):
    return sklearn.preprocessing.minmax_scale(x, axis=axis)
```

```
Centroids: [2262.92623048 2067.01946843 1853.86098094 ...
1752.38203045 1856.01586703
 2122.64436854]

Shape of Spectral Centroids: (1629,)

frames: range(0, 1629)
```

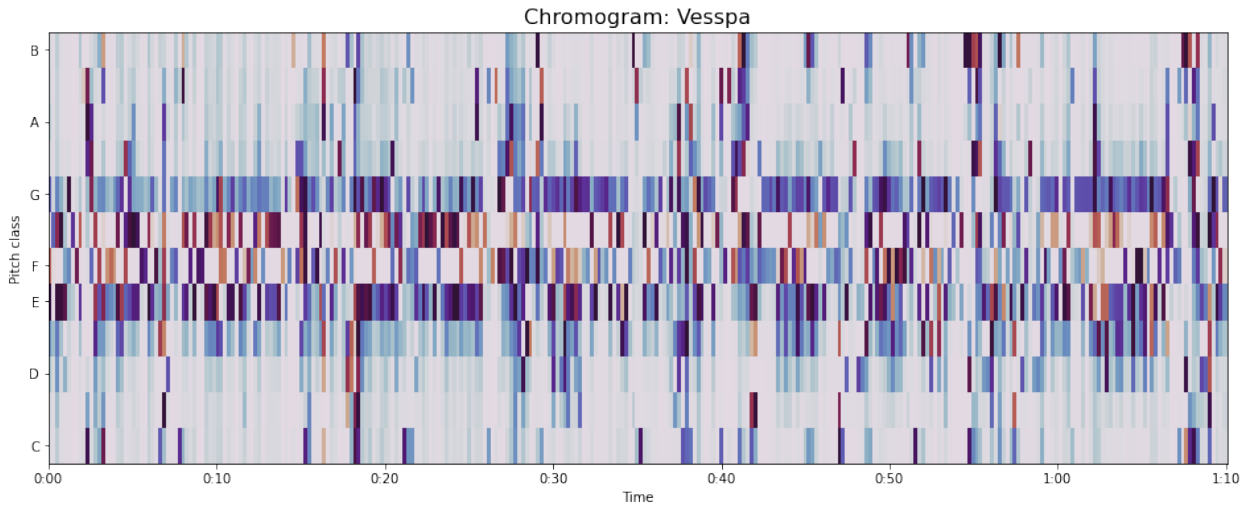
```
t: [0.00000000e+00 2.32199546e-02 4.64399093e-02 ... 3.77556463e+01
 3.77788662e+01 3.78020862e+01]

#Plotting the Spectral Centroid along the waveform
plt.figure(figsize = (16, 6))
librosa.display.waveplot(audio_cangoo, sr=sr, alpha=0.4, color =
'#A300F9', lw=3)
plt.plot(t, normalize(spectral_centroids), color='#FFB100', lw=2)
plt.legend(["Spectral Centroid", "Wave"])
plt.title("Spectral Centroid: Cangoo Bird", fontsize=16);
```



#8. Chroma Frequencies

Note: Chroma features are an interesting and powerful representation for music audio in which the entire spectrum is projected onto 12 bins representing the 12 distinct semitones (or chromas) of the musical octave.

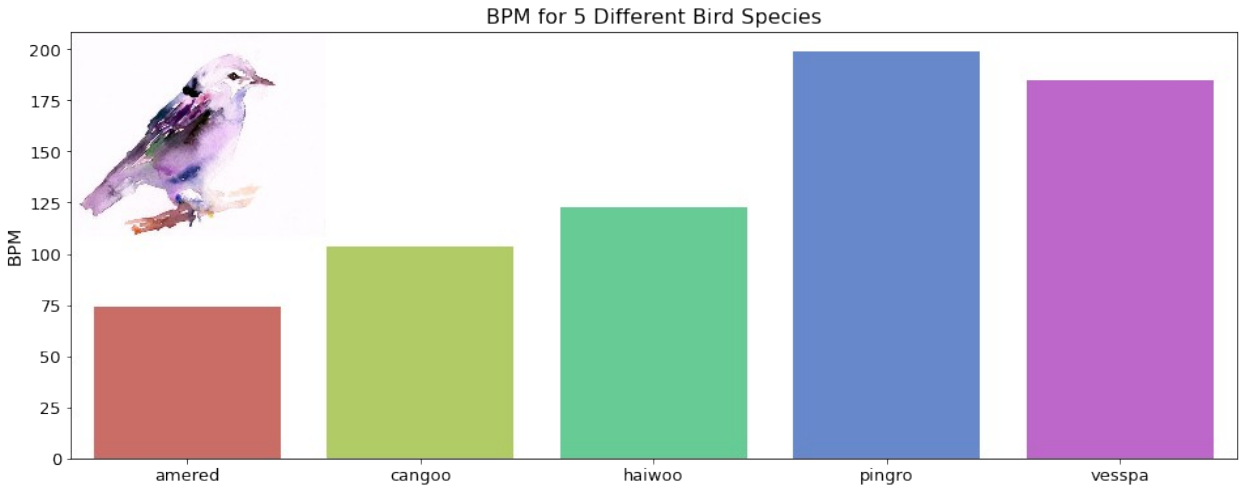
```
# Increase or decrease hop_length to change how granular you want your
data to be
hop_length = 5000

# Chromogram Vesspa
chromagram = librosa.feature.chroma_stft(audio_vesspa, sr=sr_vesspa,
hop_length=hop_length)
print('Chromogram Vesspa shape:', chromagram.shape)

plt.figure(figsize=(16, 6))
librosa.display.specshow(chromagram, x_axis='time', y_axis='chroma',
hop_length=hop_length, cmap='twilight')

plt.title("Chromogram: Vesspa", fontsize=16);

Chromogram Vesspa shape: (12, 309)
```



#9. Tempo BPM (beats per minute)

Note: Dynamic programming beat tracker.

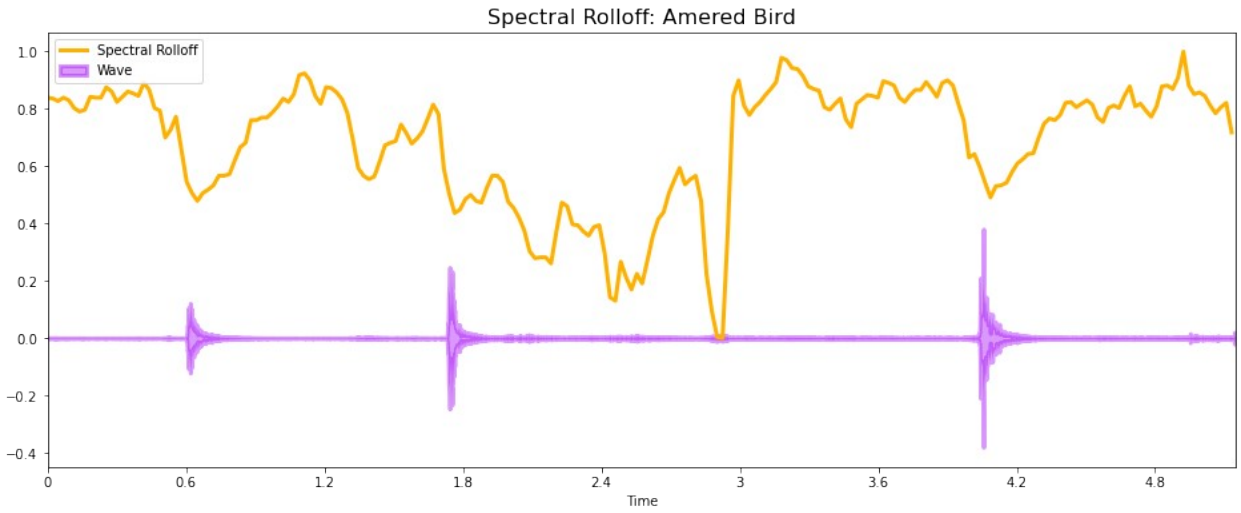
```
# Create Tempo BPM variable
tempo_amered, _ = librosa.beat.beat_track(y_amered, sr = sr_amered)
tempo_cangoo, _ = librosa.beat.beat_track(y_cangoo, sr = sr_cangoo)
tempo_haiwoo, _ = librosa.beat.beat_track(y_haiwoo, sr = sr_haiwoo)
tempo_pingro, _ = librosa.beat.beat_track(y_pingro, sr = sr_pingro)
tempo_vesspa, _ = librosa.beat.beat_track(y_vesspa, sr = sr_vesspa)

data = pd.DataFrame({"Type": bird_sample_list ,
                     "BPM": [tempo_amered, tempo_cangoo, tempo_haiwoo,
tempo_pingro, tempo_vesspa] })

# Image
bird = mpimg.imread('../input/birdcall-recognition-data/violet
bird.jpg')
imagebox = OffsetImage(bird, zoom=0.34)
xy = (0.5, 0.7)
ab = AnnotationBbox(imagebox, xy, frameon=False, pad=1, xybox=(0.05,
158))

# Plot
plt.figure(figsize = (16, 6))
ax = sns.barplot(y = data["BPM"], x = data["Type"], palette="hls")
ax.add_artist(ab)

plt.ylabel("BPM", fontsize=14)
plt.yticks(fontsize=13)
plt.xticks(fontsize=13)
plt.xlabel("")
plt.title("BPM for 5 Different Bird Species", fontsize=16);
```



#10. Spectral Rolloff

Note: Is a measure of the *shape of the signal*. It represents the frequency below which a specified percentage of the total spectral energy (e.g. 85%) lies.

```
# Spectral RollOff Vector
spectral_rolloff = librosa.feature.spectral_rolloff(audio_amered,
sr=sr_amered)[0]

# Computing the time variable for visualization
frames = range(len(spectral_rolloff))
# Converts frame counts to time (seconds)
t = librosa.frames_to_time(frames)

# The plot
plt.figure(figsize = (16, 6))
librosa.display.waveplot(audio_amered, sr=sr_amered, alpha=0.4, color
= '#A300F9', lw=3)
plt.plot(t, normalize(spectral_rolloff), color='#FFB100', lw=3)
plt.legend(["Spectral Rolloff", "Wave"])
plt.title("Spectral Rolloff: Amered Bird", fontsize=16);
```



There are many more features that librosa can extract from sound. Check them all here.

4. Additional Data

Why stop at 100? thread here

- @Vopani kindly scraped the remaining of the available bird audios from Xeno-Canto site.
- The data:
  - doesn't contain already available train audios from competition data
  - only MP3 format
  - same license as the original data
  - no corrupt audio present

However, the Competition Hosts replied with:

- limiting to 100 audio files had the reason to not overload the memory
- only 100 top rated audios were used
- excluded videos that did not alow derivatives

They also recommend the following:

- the upper limit is usually 500 recordings per species
- how many recordings are needed to train? (maybe even less than 100?)

Note: So, should you use the extended data? Should you stick only with original data?
I have no clue, try multiple ideas and see what performs better

```
# Import the .csv files (corresponding with the extended data)
train_extended_A_Z = pd.read_csv("../input/xeno-canto-bird-recordings-extended-a-m/train_extended.csv")

# Create base directory
base_dir_A_M = "../input/xeno-canto-bird-recordings-extended-a-m"
```

```
base_dir_N_Z = "../input/xeno-canto-bird-recordings-extended-n-z"

# Create Full Path column to the audio files
train_extended_A_Z['full_path'] = base_dir_A_M +
train_extended_A_Z['ebird_code'] + '/' +
train_extended_A_Z['filename']
```

Sanity Check: are all the files the same length?

```
def count_files_dir(dir_name = "Default", pref = "Def"):

    birds_names = list(os.listdir(dir_name + "/" + pref))
    total_len = 0

    for bird in birds_names:
        total_len += len(os.listdir(dir_name +"/" + pref + "/" +
bird))

    return total_len

A_M = count_files_dir(base_dir_A_M, pref = "A-M")
N_Z = count_files_dir(base_dir_N_Z, pref = "N-Z")

print("There are {:,} birds in A-Z .csv
file".format(len(train_extended_A_Z)), "\n" +
      "\n" +
      "and there are {:,} audio recs.".format(A_M + N_Z))
```

```
There are 23,041 birds in A-Z .csv file

and there are 23,041 audio recs.
```