workbook1

April 12, 2025

1 Workbook 1 - Music with Python

In this notebook, we: - Work with musical note frequencies - Generate MIDI files - Convert MIDI to WAV using `midi2audio` and `FluidSynth` - Play music with `pygame.midi`

1.1 Install packages

[1]:
```
# Installation process may differ across systems
# apt-get install fluidsynth
# !pip install IPython
# !pip install midiutil
# !pip install midi2audio
# !pip install music21
# !pip install pygame
# !pip install scikit-learn
# !wget https://raw.githubusercontent.com/musescore/MuseScore/master/share/
 ↪sound/FluidR3Mono_GM.sf3
```

[2]:
```
# Import after installing

import numpy as np
import IPython
from IPython.display import Audio, display
import midiutil
import midi2audio
import music21
```

1.2 Define the frequencies of notes

[3]:
```
NOTE_FREQUENCIES = {
    'C4': 261.63, 'D4': 293.66, 'E4': 329.63, 'F4': 349.23,
    'G4': 392.00, 'A4': 440.00, 'B4': 493.88, 'C5': 523.25,
    'F3': 174.61, 'G3': 196.00, 'A3': 220.00, 'B3': 246.94
}
```

1.3 Make a single note sine wave and play

[4]:
```
def create_sine_wave(frequency, duration, sample_rate=44100):
    t = np.linspace(0, duration, int(sample_rate * duration))
    wave = np.sin(2 * np.pi * frequency * t)
    return wave
```

[5]:
```
single_note = create_sine_wave(NOTE_FREQUENCIES['A4'], 1)
#440Hz, 1 second
display(Audio(single_note, rate=44100))
```

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

1.4 Make a single note square wave and play

A square wave is made up of a fundamental frequency plus many odd-numbered harmonics (3x, 5x, etc.) In this example, we will use harmonics 3x, 5x, ..., 19x, based on the following equation.

$\text{square}(f,t) = \frac{4}{\pi} \sum_{k=1,3,5,\ldots,19} \frac{1}{k} \sin(2\pi k f t)$

[6]:
```
def create_square_wave(frequency, duration, sample_rate=44100):
    t = np.linspace(0, duration, int(sample_rate * duration))
    wave = np.zeros_like(t)
    for k in range(1, 21, 2):
        wave += (1 / k) * np.sin(2 * np.pi * k * frequency * t)
    wave *= 4 / np.pi
    return wave
```

[7]:
```
single_note = create_square_wave(NOTE_FREQUENCIES['A4'], 1)
#440Hz, 1 second
display(Audio(single_note, rate=44100))
```

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

1.5 Make a chord and play

[8]:
```
def create_chord(frequencies, duration, sample_rate=44100):
    waves = [create_sine_wave(f, duration, sample_rate) for f in frequencies]
    return np.mean(waves, axis=0)

chord_frequencies = [NOTE_FREQUENCIES['C4'], NOTE_FREQUENCIES['E4'],␣
 ↪NOTE_FREQUENCIES['G4']] #C major chord
chord = create_chord(chord_frequencies, 2)
```

[9]:
```
chord_progression = [
       ([NOTE_FREQUENCIES['C4'], NOTE_FREQUENCIES['E4'],␣
 ↪NOTE_FREQUENCIES['G4']], 2.0),#C Major chord for 2 secs
       ([NOTE_FREQUENCIES['F3'], NOTE_FREQUENCIES['A3'],␣
 ↪NOTE_FREQUENCIES['C4']], 1.0),#F Major chord for 1 sec
```

```
        ([NOTE_FREQUENCIES['C4'], NOTE_FREQUENCIES['E4'],␣
 ↪NOTE_FREQUENCIES['G4']], 1.0),#C Major Chord for 1 sec
        ([NOTE_FREQUENCIES['F3'], NOTE_FREQUENCIES['A3'],␣
 ↪NOTE_FREQUENCIES['C4']], 1.0),#F Major Chord for 1 sec
        ([NOTE_FREQUENCIES['C4'], NOTE_FREQUENCIES['E4'],␣
 ↪NOTE_FREQUENCIES['G4']], 1.0),#C Major Chord for 1 sec
        ([NOTE_FREQUENCIES['G3'], NOTE_FREQUENCIES['B3'],␣
 ↪NOTE_FREQUENCIES['D4']], 1.0),#G Major Chord for 1 sec
        ([NOTE_FREQUENCIES['C4'], NOTE_FREQUENCIES['E4'],␣
 ↪NOTE_FREQUENCIES['G4']], 1.0)]#C Major Chord for 1 sec
chord_waves = []
for frequencies, duration in chord_progression:
    chord_wave = create_chord(frequencies, duration)
    chord_waves.append(chord_wave)
backing_track = np.concatenate(chord_waves)
display(Audio(backing_track, rate=44100))
```

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

1.6 Make a melody line and play

[10]:
```
from IPython.display import Image, display
display(Image(url="https://musescore.com/static/musescore/scoredata/g/
 ↪d08471e13f9aa1a2b2605bb7347943654b012f0e/score_0.svg?no-cache=1715697879"))
```

```
<IPython.core.display.Image object>
```

[11]:
```
melody_notes = [
       NOTE_FREQUENCIES['C4'], NOTE_FREQUENCIES['C4'], NOTE_FREQUENCIES['G4'],␣
 ↪NOTE_FREQUENCIES['G4'],
       NOTE_FREQUENCIES['A4'], NOTE_FREQUENCIES['A4'], NOTE_FREQUENCIES['G4'],
       NOTE_FREQUENCIES['F4'], NOTE_FREQUENCIES['F4'], NOTE_FREQUENCIES['E4'],␣
 ↪NOTE_FREQUENCIES['E4'],
       NOTE_FREQUENCIES['D4'], NOTE_FREQUENCIES['D4'], NOTE_FREQUENCIES['C4']
   ]
melody_durations = [0.5] * 14 #14 notes (0.5 sec for one beat)
melody_durations[6] = 1.0#The 7th note lasts for two beats
melody_durations[-1] = 1.0#The last note lasts for two beats
```

[12]:
```
def create_melody(notes, durations, sample_rate=44100):
   waves = []
   for note, duration in zip(notes, durations):
       wave = create_sine_wave(note, duration, sample_rate)#Generate a sine␣
 ↪wave for each note
       waves.append(wave)#Append to the list
   return np.concatenate(waves)#Concatenate
```

```
melody = create_melody(melody_notes, melody_durations)
display(Audio(melody, rate=44100))
```

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

1.7 Adjust amplitudes, mix and play

[13]:
```
melody = melody * 0.7 #adjust the amplitude
backing_track = backing_track * 0.3#adjust the amplitude
combined = melody + backing_track#combine
combined = combined / np.max(np.abs(combined))
display(Audio(combined, rate=44100))
```

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

1.8 Write a MIDI file (the initial melody line from Twinkle Twinkle Little Star) and save

[14]:
```
from midiutil import MIDIFile # Import library

midi = MIDIFile(1) # Create a MIDI file that consists of 1 track
track = 0 # Set track number
time = 0 # Where is the event placed (at the beginning)
tempo = 120 # The tempo (beats per minute)
midi.addTempo(track, time, tempo) # Add tempo information
```

[15]:
```
help(midi.addTempo)
```

```
Help on method addTempo in module midiutil.MidiFile:

addTempo(track, time, tempo) method of midiutil.MidiFile.MIDIFile instance
    Add notes to the MIDIFile object

    :param track: The track to which the tempo event  is added. Note that
        in a format 1 file this parameter is ignored and the tempo is
        written to the tempo track
    :param time: The time (in beats) at which tempo event is placed
    :param tempo: The tempo, in Beats per Minute. [Integer]
```

[16]:
```
notes = [60, 60, 67, 67, 69, 69, 67, # CCGGAAG
         65, 65, 64, 64, 62, 62, 60] # FFEEDDA
durations = [1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2]
```

[17]:
```
current_time = 0
for pitch, duration in zip(notes, durations):
   midi.addNote(track, 0, pitch, current_time, duration, 100)
   current_time += duration
   print(pitch, duration)
```

```
# add notes to MIDI file
with open("twinkle.mid", "wb") as f:
    midi.writeFile(f) # write MIDI file
```

```
60 1
60 1
67 1
67 1
69 1
69 1
67 2
65 1
65 1
64 1
64 1
62 1
62 1
60 2
```

1.9 Convert the MIDI file into an audio file using downloaded synthesizer

[18]:
```
# Download the FluidR3Mono_GM.sf3 soundfont from MuseScore's GitHub
# !wget https://raw.githubusercontent.com/musescore/MuseScore/master/share/
 ↪sound/FluidR3Mono_GM.sf3
```

[19]:
```
from midi2audio import FluidSynth # Import library
fs = FluidSynth("FluidR3Mono_GM.sf3") # Initialize FluidSynth
```

[20]:
```
# Have to add fluidsynth to your system path (in the directory as this ipynb␣
 ↪file) for this to work...
# See e.g. https://stackoverflow.com/questions/75353745/
 ↪how-do-i-set-up-midi2audio-on-windows
# I had to follow the suggested steps to get this to worka on my tablet
```

[21]:
```
fs.play_midi("twinkle.mid")
```

```
FluidSynth runtime version 2.4.4
Copyright (C) 2000-2025 Peter Hanappe and others.
Distributed under the LGPL license.
SoundFont(R) is a registered trademark of Creative Technology Ltd.
```

[22]:
```
fs.midi_to_audio("twinkle.mid", "twinkle.wav") # convert MIDI to audio
```

```
FluidSynth runtime version 2.4.4
Copyright (C) 2000-2025 Peter Hanappe and others.
Distributed under the LGPL license.
SoundFont(R) is a registered trademark of Creative Technology Ltd.
```

```
Rendering audio to file 'twinkle.wav'..
```

[23]:
```
display(Audio('twinkle.wav'))
```

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

1.10 Write a MusicXML file and save

[24]:
```
from music21 import *
melody = stream.Stream()
melody.append(metadata.Metadata())
melody.metadata.title = 'Twinkle Twinkle Little Star'
melody.metadata.composer = 'Unknown'
melody.append(meter.TimeSignature('4/4'))
melody.append(key.Key('C'))
```

[25]:
```
notes = [
   ('C4', 'quarter'),
   ('C4', 'quarter'),
   ('G4', 'quarter'),
   ('G4', 'quarter'),
   ('A4', 'quarter'),
   ('A4', 'quarter'),
   ('G4', 'half'),
]
for pitch, duration in notes:
   n = note.Note(pitch)
   n.duration.type = duration
   melody.append(n)
```

[26]:
```
melody.write('musicxml', fp='twinkle.musicxml')
```

[26]:
```
PosixPath('/Users/ryoandrewonozuka/Documents/GitHub/notes/ucsd/cse153/wb1/twinkl
e.musicxml')
```

1.11 Generate and receive midi events using pygame

[27]:
```
import pygame
import pygame.midi
from pygame.locals import *
import random
import time
```

```
pygame 2.6.1 (SDL 2.28.4, Python 3.10.16)
Hello from the pygame community. https://www.pygame.org/contribute.html
```

[28]:
```
pygame.init()
pygame.fastevent.init()
pygame.midi.init()
```

[29]:
```
# This will (probably) receive input from an external controller,
# and output to your machine. By changing the ids you could also
# output to a controller. You can delete the parts relating to inputs
# if you don't have a midi controller
input_id = pygame.midi.get_default_input_id()
output_id = pygame.midi.get_default_output_id()
```

[30]:
```
input_id, output_id
```

[30]:
```
(-1, -1)
```

[31]:
```
i = None
player = None

if input_id != -1:
    i = pygame.midi.Input(input_id)
else:
    print("Couldn't find a midi input device")
    print("This isn't required for the course so please ignore this part")

if output_id != -1:
    player = pygame.midi.Output(output_id)
    player.set_instrument(0)
else:
    print("Couldn't find a midi output device")
    print("This isn't required for the course so please ignore this part")
```

```
Couldn't find a midi input device
This isn't required for the course so please ignore this part
Couldn't find a midi output device
This isn't required for the course so please ignore this part
```

[32]:
```
# Play a random note
if player:
    r = random.choice(range(30,90)) # Pitch range
    time.sleep(0.25)
    player.note_on(r, 50)
    time.sleep(0.25)
    player.note_off(r)
```

[33]:
```
going = False
#going = True # Uncomment if you want to run this part...

clock = pygame.time.Clock()
```

```
while going:
    clock.tick(60)
    if i.poll():
        midi_events = i.read(10)
        # print(str(midi_events)) # dump all the info...
        for e in midi_events:
            note_info, t = e
            z,note,velocity,_ = note_info
            if velocity > 0:
                print((note, velocity))
                player.note_on(note, velocity)
            if velocity == 0:
                player.note_off(note)
```

1.12 Classifiying song emotions from BPM and keys using sklearn

[34]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

#List of songs on Billboard's 2024 Year-End Hot 100 chart (from No.1 to No.10)
song_titles = ["Lose Control", "A Bar Song (Tipsy)", "Beautiful Things", "I Had␣
 ↪Some Help", "Lovin on Me", "Not Like Us", "Espresso", "Million Dollar Baby",␣
 ↪"I Remember Everything", "Too Sweet"]
# For example, the BPM of "Lose Control" is 128.
BPMs = [128, 81, 105, 126, 105, 101, 104, 138, 78, 117]
# For example, the key of "Lose Control" is minor, but the key of "Beautiful␣
 ↪Things" is major.
is_major =  [0, 0, 1, 1, 1, 1, 1, 0, 1, 0]
# For example, the emotion of "Lose Control" is negative, but the emotion of "A␣
 ↪Bar Song (Tipsy)" is positive
is_positive = [0, 1, 1, 0, 1, 0, 1, 1, 0, 1]
```

[35]:
```
#Stacking inputs
X = np.column_stack((np.array(BPMs), np.array(is_major)))
#Splitting the dataset (50:50 ratio)
X_train, X_test, Y_train, Y_test = train_test_split(X, is_positive, test_size=0.
 ↪5, random_state=42)
#Train your model
model = LogisticRegression(random_state=42)
model.fit(X_train, Y_train)
```

[35]:
```
LogisticRegression(random_state=42)
```

[36]:
```
#Test your model and make a report
y_pred = model.predict(X_test)
```

```
my_report = classification_report(Y_test, y_pred)
my_report
```

[36]:
```
'              precision    recall  f1-score   support\n\n           0
0.50      0.33      0.40         3\n           1       0.33      0.50      0.40
2\n\n    accuracy                           0.40         5\n   macro avg
0.42      0.42      0.40         5\nweighted avg       0.43      0.40      0.40
5\n'
```

[ ]: