**Data Science Analysis of Computational Musicology Techniques**

Hovanes Keseyan

Project Design Proposal

ENGR 296V/D

Winter 2018

Advisor: Dr. Thomas Fryer

Table of Contents

[Abstract 4](#_Toc509538880)

[Introduction 6](#_Toc509538881)

[Literature 7](#_Toc509538882)

[Previous research 7](#_Toc509538883)

[Findings & unanswered questions 7](#_Toc509538884)

[Your preliminary work on the project topic 8](#_Toc509538885)

[Remaining questions 8](#_Toc509538886)

[Methodology 9](#_Toc509538887)

[Approach 9](#_Toc509538888)

[Data collection 10](#_Toc509538889)

[*Figure 1: Sample of a MIDI file converted to CSV file in NotePad (left) and Microsoft Excel (right)* 11](#_Toc509538890)

[Analytic methods 13](#_Toc509538891)

[Plans for interpreting results 14](#_Toc509538892)

[Results 15](#_Toc509538893)

[*Figure 2: Effectiveness of CSV Compression in R* 15](#_Toc509538894)

[*Figure 3: R Markdown Song Report for Sample DLL file* 16](#_Toc509538895)

[*Figure 4: R Markdown Song Report for Sample RAG file* 17](#_Toc509538896)

[*Figure 5: Unity program visualization of controller input* 18](#_Toc509538897)

[*Figure 6: Unity program visualization of notes created by controller input* 19](#_Toc509538898)

[Discussion and Conclusion 20](#_Toc509538899)

[References 22](#_Toc509538900)

[Appendix A - Songinfo 25](#_Toc509538901)

[Songinfo – IDL file 25](#_Toc509538902)

[Data Preparation 25](#_Toc509538903)

[Load the songs into R 25](#_Toc509538904)

[Extract data 26](#_Toc509538905)

[Convert note on c to note off c 26](#_Toc509538906)

[Song statistics 27](#_Toc509538907)

[Number of notes: 4272 27](#_Toc509538908)

[Duration (seconds): 166.35 27](#_Toc509538909)

[Derived Tempo: 77.0423805 27](#_Toc509538910)

[Plots of note frequencies 28](#_Toc509538911)

[Songinfo – RAG file 33](#_Toc509538912)

[Data Preparation 33](#_Toc509538913)

[Load the songs into R 33](#_Toc509538914)

[Extract data 34](#_Toc509538915)

[Convert note on c to note off c 34](#_Toc509538916)

[Song statistics 35](#_Toc509538917)

[Number of notes: 393 35](#_Toc509538918)

[Duration (seconds): 4.623 35](#_Toc509538919)

[Derived Tempo: 255.0292018 35](#_Toc509538920)

[Plots of note frequencies 36](#_Toc509538921)

[Appendix B – R Generation Algorithm 38](#_Toc509538922)

[Appendix C – Compression Algorithm 42](#_Toc509538923)

[Appendix D – Unity Code 43](#_Toc509538924)

[Controller Inputs 43](#_Toc509538925)

[Call to Play Note 45](#_Toc509538926)

[CSV Manager 46](#_Toc509538927)

[Appendix E – Python Neural Net 48](#_Toc509538928)

# Abstract

The project will be a study of computational musicology using the data science techniques learned in the UCR MSOL program. Music can be represented through data in many ways, one of which being a time series defining each note by a pitch, tone, and duration. I will be analyzing various sources of musical data, both through publicly available repositories as well as my own creation of musical data through my own experimentation. Computational musicology is the study of music with computer modeling and simulation using this type of data.  
  
The next step after the data is obtained includes organizing and formatting the data into a format where I can properly start data mining. The data mining phase includes analyzing patterns in the musical data and eventually classifying the music into various groups, which could be grouped into things like genre, instrument, tempo, artist, and others.  
  
The next step that I would like to achieve through this project is to see if I can reliably produce music through given parameters. For instance, if I input which genre and artist to imitate, and input which instruments and at what tempo I would like it to imitate, and it outputs a musical track. This will involve research into creative music software that has been developed, along with what else I have learned in class that can help me develop my own version.  
  
This project will help set me up with some more ambitious things that I would like to do with computational musicology, which includes the following:

* Write an algorithm that will detect and write musical notation through audio input
* Create a continuous visual output that is determined by musical input
* Write a program that can separate different instrumental track called “stems” from a complete song
* Much more

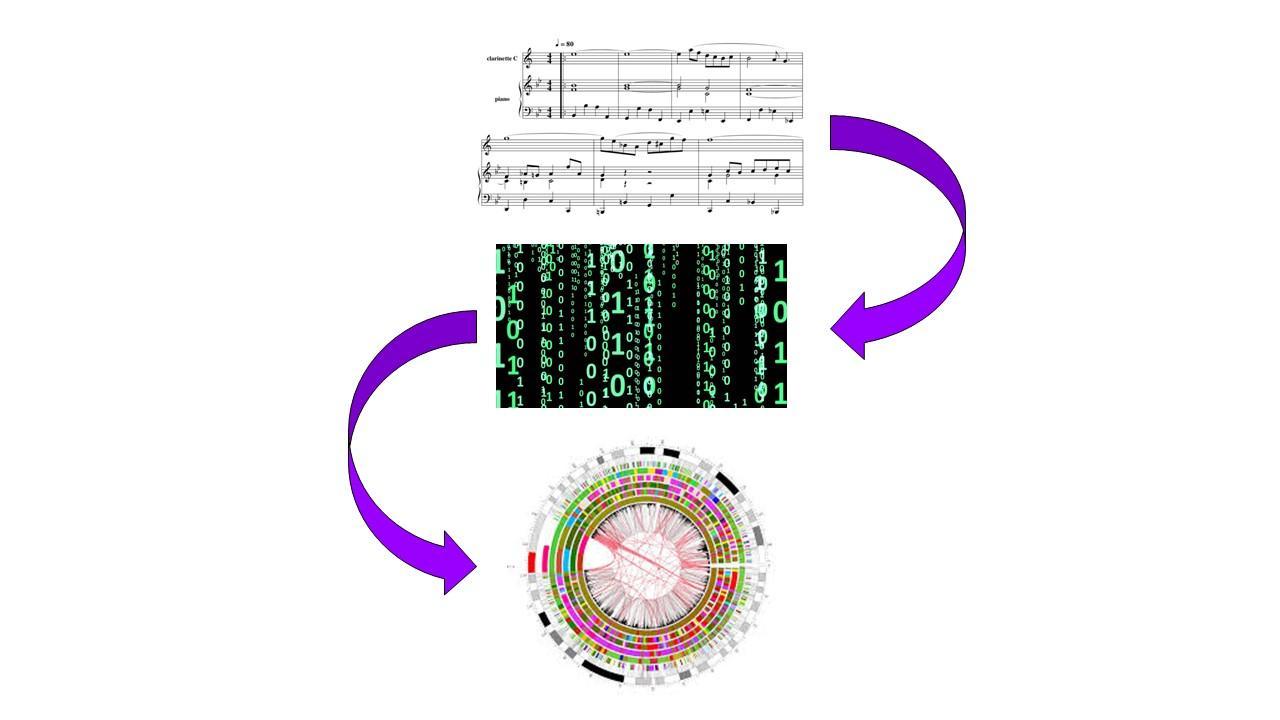
I have found many research papers by and interviews of people exploring this field that have inspired me to produce my own work in this area. There are software packages online that will help me get a head start on this, like Synesthesia, but I will be performing all of the data analysis on applications that we have been using at UCR, which are R and Matlab.  
  
This project will put to practice my knowledge in my major through the following topics:

* Data wrangling
* Data mining
* Classification algorithms
* Pattern recognition
* Statistical modeling
* Machine learning
* Neural networks
* Computer graphics

The project will report on the currently available methods of analyzing musical data and the possible applications of each method.

# Introduction

Computational musicology presents a whole new way of looking at music. Using modern technology, coupled with a good knowledge of data science, and quite a bit of practice, we can do incredible new things with music. This paper will present findings in the field of computational musicology, raise some new questions for what needs can be fulfilled by pursuing work in the field, present ideas for the next steps forward, and provide some analysis of musical data which could provide inspiration for the next aspiring computational musicologist.



# Literature

## Previous research

I have spent some time researching the many aspects of this field. This includes:

* Reading theoretical research papers by leading professors in the field [6, 13]
* Discovering toolsets developed by researchers [2, 3, 4, 5, 11]
* Watching videos on similar projects done by students [7, 8, 9]
* Researching resources that will help obtain and prepare the data [1, 10]
* Keeping updated with commercial applications in the field [12, 14]

## Findings & unanswered questions

I have learned a great deal about current methods in computational methodology. The first group of items I have learned about is the tools that are used to prepare for this type of work.

First, we need to obtain the music file. The easiest file type to start working with is a .midi file, which is output by simple instruments like a generic electronic keyboard, and is an easy file type to find downloadable songs online for, even though it does not have the highest musical range or quality. We will initially keep our scope within .midi files for simplicity, but we hope to evolve into utilizing more complex musical formats. We will start with downloading available .midi files from sources like [10], but we can create our own music later if our work requires it.

Next, the music will need to be converted into a data format that will be easy to perform data science techniques with. We will start by using the tool in reference [1] which converts .midi files to .csv files. We will still need to reformat these files once they are in .csv form, but that will be simple work for a data scientist.

At this point, the brunt of the data science work comes into play. We will be using our programming application of choice (Python and R, with possibly some Matlab script) to work with the algorithms that will do most of the data mining work. The tool sets that I have researched will help speed this process along so that we can obtain new findings from this study outside of what people in the field have already accomplished.

Finally, we will use data mining and machine learning to produce songs of our own, which will create a new data set for analysis.

All of the work up to this point has been performed by others in the field. I will recreate this work, and discover my own questions and findings along the way. As I tackle these problems, I will obtain the experience necessary to achieve my goals in this project, which are to:

* Write an algorithm that will detect and write musical notation through audio input
* Create a continuous visual output that is determined by musical input
* Write a program that can separate different instrumental track called “stems” from a complete song

## Your preliminary work on the project topic

I have researched the methods that others have used to start out with similar projects. I will replicate these efforts in my study as I collect my findings.

## Remaining questions

Questions that I know I will need to answer are:

1. How detailed will the musical data I am working with be, and can it replicate high quality music that will be pleasurable to listen by others?
2. How effective of an algorithm can I make to correctly separate different instruments and voices from a musical file?
3. How reliably can I get the computer to generate music similar to the input without overfitting the data?
4. How can I make sure excess noise does not confuse the computer while it is reading the musical data and trying to create its own?
5. What metrics will I be looking for in my musical analysis?

# Methodology

## Approach

The overall approach taken on this project boils down to the following major capabilities:

* Data collection - I obtained songs in 4 ways
  + Downloaded MIDI files from internet
  + Algorithmically created CSV on R
  + Created Unity program to use controller to play music
    - Created sound files for each note
  + AI Neural Network Output
* Software
  + R
  + Unity
  + Python
  + MIDICSV
  + CReMA
  + LMMS
* Code developed
  + R code for song report
  + R code for song generation
  + R code for compression
  + Unity script for controller input
  + Unity script for visual note output
  + Unity script for sound output
  + Unity script for CSV file export
* Code referenced
  + Python code for AI NN - Rudimentary AI composer [15]
  + Unity script for data output [34]
* Visualizations produced
  + R Markdown Song Report
  + Unity song stream

The plan of action was as follows:

1. Downloaded MIDI files to create MIDI repository
   1. Dave’s MIDI: 6 files, 810 KB
   2. Melody MIDIs: 65703 files 64.1 GB
   3. Rhythm MIDIs: 1907 files, 1.86 GB
2. Downloaded MIDICSV and CSVMIDI, and CReMA
3. Converted MIDI files to CSV using MIDICSV and CReMA
   1. Dave’s MIDI: 6 files, 8.50 MB
   2. Melody MIDIs: 3571 files, 3.58 GB
4. Explore compression
   1. Statistics on how much compression helps
5. Create R Markdown file
   1. Cleaned up CSV files
   2. displays simple statistics
   3. graphs visuals of notes
6. Created a basic song CSV
   1. Use code in R to make some simple rhythm patterns
   2. Random verses
   3. Repeating chorus
   4. Plug into R markdown report
7. Downloaded Python, PyCharm, and Python modules
8. Ran rudimentary ai composer
9. Created Unity program for controller input of sound
   1. Displays key presses
   2. Displays note values and visualizations
   3. Plays sounds
   4. Exports data to CSV file

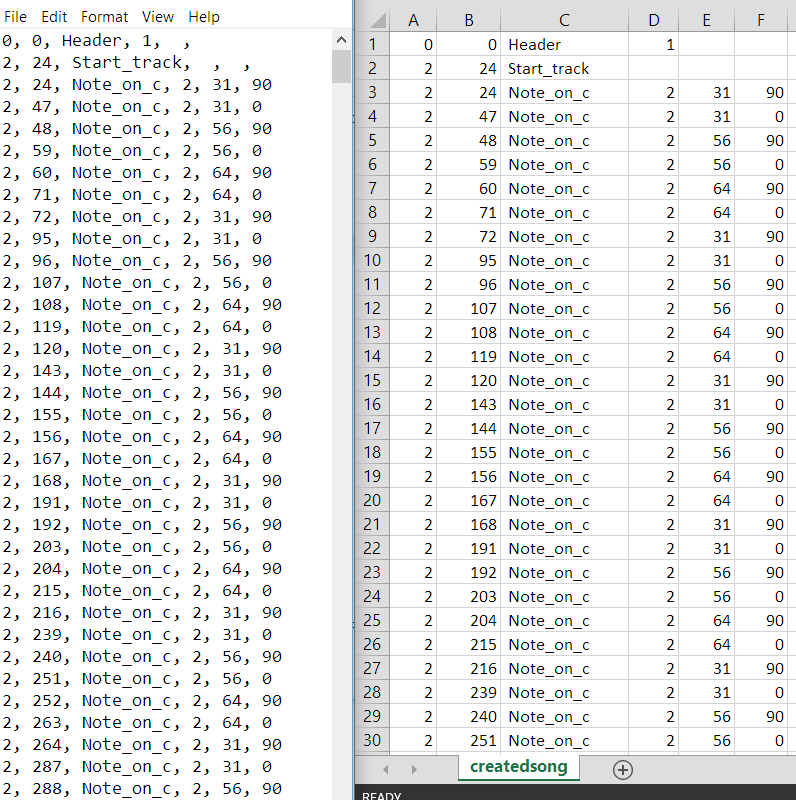
## Data collection

The data for this project is musical data, which will was collected in both MIDI and CSV format.

MIDI (Musical Instrument Digital Interface) is a file format which records signals contained in a musical piece that instruct a MIDI synthesizer on which musical notes to play and when. The synthesizer will output the signals as sounds based on the configuration of the synthesizer. MIDI allows musical structure to be preserved while the actual sounds produced can vary wildly based on the synthesizer configurations. Synthesizers are abundant and vary from the common usage of the term synthesizer used in making electronic music, to the simplest native apps that ship with operating systems (such as Windows Media Player). A MIDI file may also contain metadata about the musical piece, such as track titles, tempo, and more. These properties set the MIDI format apart from other file formats for storing music like WAV, MP3, and OGG which only store information about the sounds that were sonically recorded during the production of the file.

CSV is the simple Comma Separated Values file format which is just text that is separated by commas which indicate a new value to be stored in the next field. This file format can be edited in a text editor like NotePad or displayed and edited easier in a spreadsheet viewer or database viewer such as Microsoft Excel. CSV files are also readable by all major statistical analysis software or computer programming languages like R, Python, C++, Matlab, and more.

*Figure 1: Sample of a MIDI file converted to CSV file in NotePad (left) and Microsoft Excel (right)*



The data used in this project was collected in four ways:

1. Free MIDI files downloaded from the internet [10][30][31] (IDL files)
2. CSV files generated from algorithms written in R code (RAG files)
3. CSV files generated from controller input using the Unity engine (UCG files)
4. MIDI files generated by artificial intelligence in Python (NNG files)

Because of the nature of MIDI files and the way they contain discrete data on the notes that shall be played in a musical piece, it is possible to convert MIDI to a format that is visually readable by humans like CSV. For this project, I used MIDICSV [1], CReMA [16], and the Python library mido to allow me to switch files between these two formats. Each of these tools are free and available open source online.

MIDICSV was the first tool used in this project after downloading the first set of IDL files. This handy tool was able to convert the files to CSV format where I began my data analysis. It also contains a function named CSVMIDI which converts a CSV file back to MIDI. In the early stages of the project, I did some testing on these functions by editing the CSV files in Microsoft Excel and converting to MIDI to hear the output, and how the songs had changed. There were errors in converting the RAG files, however. I am exploring ways of either fixing this bug or utilizing another tool to perform the conversion. A possibility is inputting the file into my Unity program and having it play the output. Until then, my algorithmically generated musical pieces will only be visualized and not heard.

CReMA is a tool much like MIDICSV, but it only converts from MIDI to CSV, and not back from CSV to MIDI. However, it can take a folder as input and convert all MIDI files in that folder at once instead of requiring a command line prompt with the filename of each file you would live to convert one by one, like what MIDICSV offers. I utilized CReMA for mass conversion of my file repository.

Mido is a library that runs on Python which is able to input a MIDI file into Python and allow the user to perform data science techniques on it. The NNG files were generated using this library.

The R-code generated songs (RAG) were developed by me as an attempt to create music algorithmically. I kept it simple here with the typical structure of a pop song with a looping rhythm, repeating chorus, and three verses. It would be interesting to expand on this with other song structures.

Songs were also generated with the Unity program (UCG) that I developed which allowed me to input note values into a USB controller and output the corresponding sound and a visual display of the notes. This required scene generation in Unity, coding of multiple scripts in C# for the logic, and creation of sounds in LMMS (an open-source Digital Audio Workstation).

## Analytic methods

Like any ambitious data science project nowadays, data collection and data preparation were a major piece of this project. The data preparation included not only converting file types, but studying the format of the data generated by the file conversion programs and preparing them for the data analysis. This was a major piece of the R Markdown Song Report code, where the appropriate data had to be extracted from the CSV files generated by MIDICSV. This would not be easily done by working with traditional applications such as NotePad and Microsoft Excel, but an algorithm written in R in this case had made the job much easier.

Methods of further compression were explored in R to record how small the file sizes could get on disk once we exclude all data that needs to be cleaned when the file is brought into R anyway. Fields that were deemed to be irrelevant to our data analysis were removed.

Visualization of the musical data is a major output of this project, which allows for analysis of musical pieces in a format that people are typically not used to. The R Markdown Song Report and Unity program introduced two new methods of viewing what is created with the intent of listening as something to be viewed.

The neural network that generated musical pieces through artificial intelligence is an example of what is now one of the primary areas of focus for many data science in the current era. A neural network is a form of machine learning which, in an abstract way, is modeled after the way the human brain makes decisions.

## Plans for interpreting results

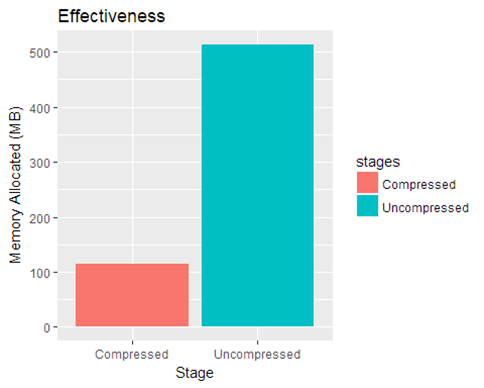
The results will be interpreted in a variety of ways. The most natural and rudimentary way is by hearing the songs produced and using human musical intuition to determine success level. There will be some statistical analysis which is where the computational musicology comes in. Lastly, there will be multiple visualizations which will allow a graphical representation of audio.

# Results

During the course of this project, I learned a great deal about the data acquisition and management constraints in performing large computational musicology projects such as this one. Managing, the downloading, transferring, storage, and conversion of 65 GB of MIDI data requires a non-trivial amount of time and resources. However, it is in the data manipulation and analysis part of a project like this where data science expertise really becomes necessary. Writing algorithms and setting up processes in the proper way becomes crucial to even ensure that your program will run without freezing or crashing.

This was a big motivating piece in my choice to develop a compression algorithm. From the results, it seemed very successful:

*Figure 2: Effectiveness of CSV Compression in R*

**

Number of Songs in Repository: 3571

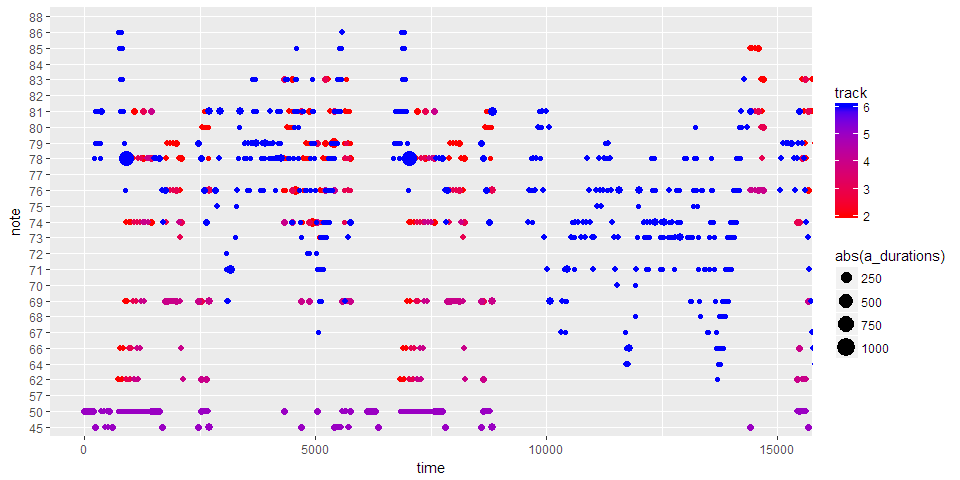
**Uncompressed** memory allocation: **514.2 MB**

**Compressed** memory allocation: **113.8 MB**

The compressed repository is **22.14%** of the size of the uncompressed repository.

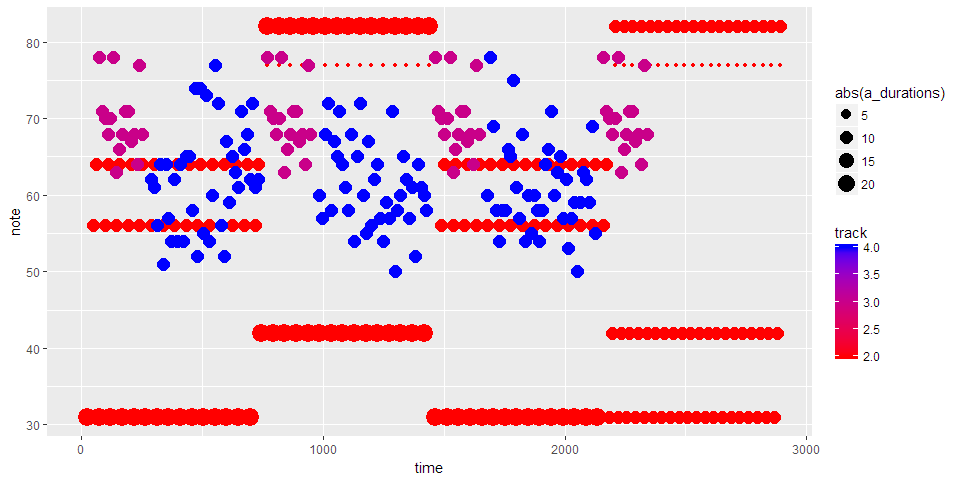
For the first visualization of generated song results, Figure 3 displays a portion of the graph generated by my R Markdown algorithm for plotting notes. The separate tracks of the song (which can indicate different instruments, such as the drums, violins, piano, trumpets, etc) are denoted by different colors; the durations of each note are denoted by size; and the pitch/frequency of the notes are plotted on the y axis with time on the x axis.

*Figure 3: R Markdown Song Report for Sample DLL file*



A visualization of the same sort but on one of the files that was generated by my own R algorithm (RAG) is shown in Figure 4 below. This song has much more contained structure, perhaps because it was generated by an data scientist and not a musician.

*Figure 4: R Markdown Song Report for Sample RAG file*

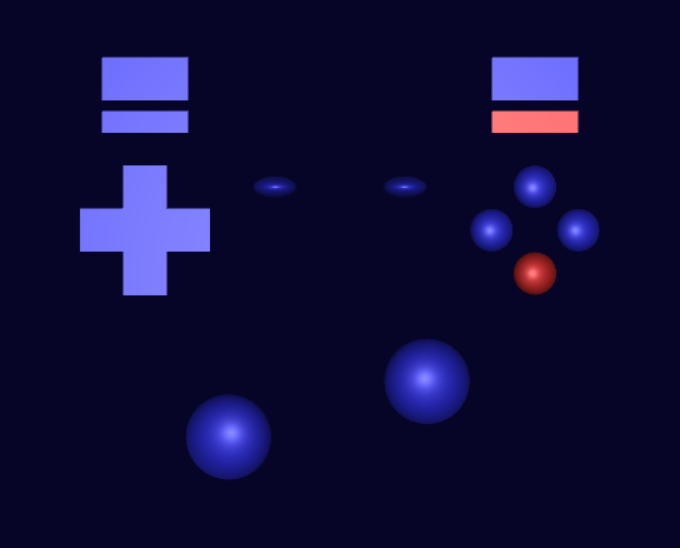


I managed to create a program in the Unity game engine with the generic name of Music Controller which I was able to connect my Logitech USB game controller to in order to generate music. This was a very ambitious task which I had come up with late into the project, and with a large wish-list of capabilities. What I managed to complete was to get mapped utility out of 12 of the 16 inputs possible from the controller. These inputs ultimately determine the pitch/frequency of the note played by mapping electric signals from the controller to a final numerical value for pitch, which is then processed by the game engine as a variable that is displayed on the screen. I created sounds in the open-source Digital Audio Workstation LMMS for a range of 40 pitches (about the size of a small personal keyboard) which I assigned to the pitch values so that a sound is played as that pitch variable changes. I used the scene-building power of Unity to plot dots that spawn on-screen when a note is played and move across the screen to show a moving series of which notes have been played. I managed to also get Unity to print out a log file of notes played and their corresponding times for the history of a game session in CSV format.

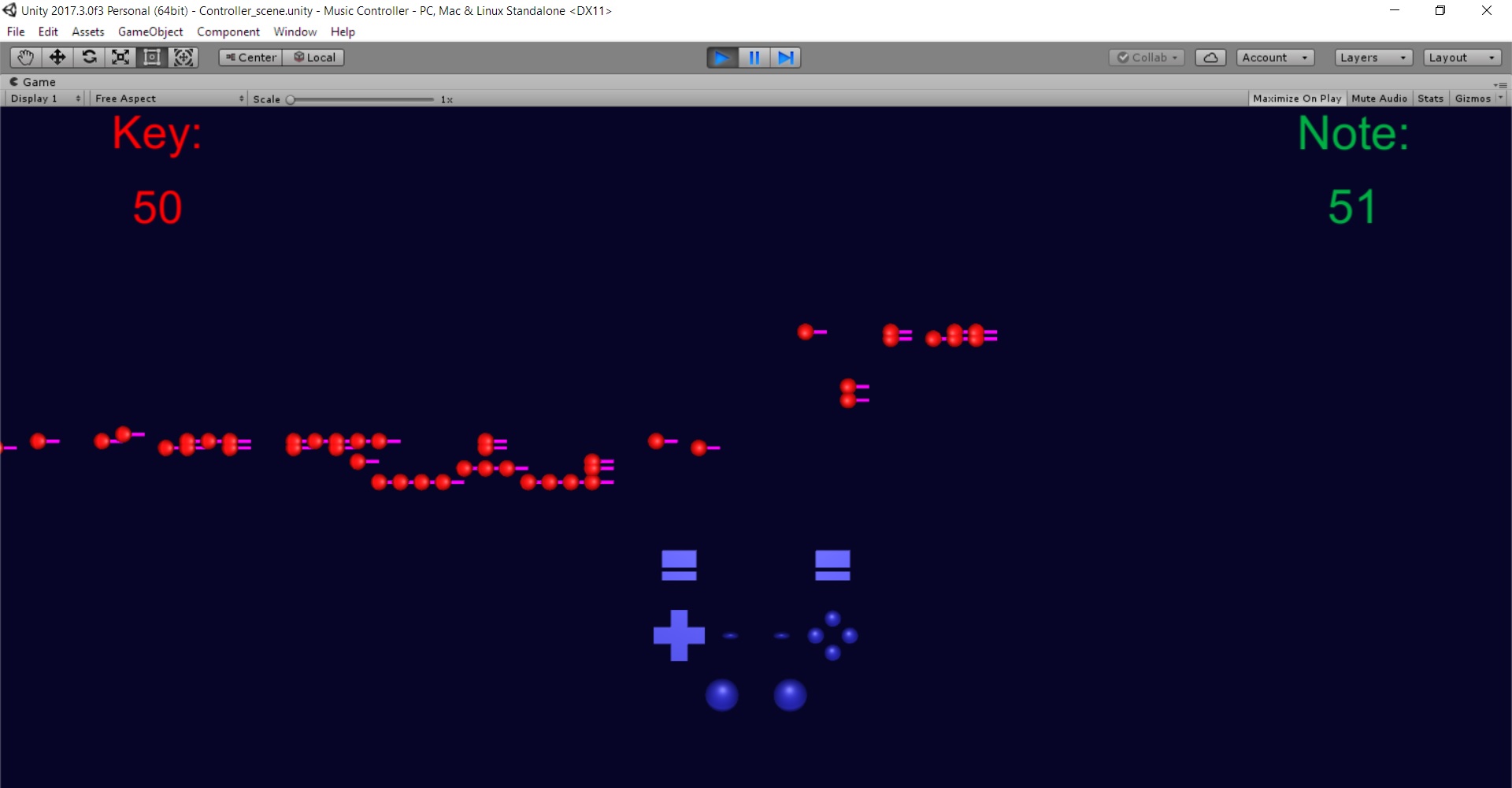
This Unity program involved the programming of many scripts which manipulate data and it was a great practice in data manipulation, data analysis, and data visualization which is uncommon in traditional data science programs. By creating my own such program, I learned a great deal about what goes on behind the scenes of much of the professional software that is used to manipulate, analyze, and visualize music.

Figure 5 below demonstrates the controller input, showing joystick motion as well as button presses in red. Figure 6 is a display of the game screen with this same controller interface as well as a display of the notes on screen.

*Figure 5: Unity program visualization of controller input*

**

*Figure 6: Unity program visualization of notes created by controller input*

**

# 

In this project I also attempted to classify the large library of songs that I acquired. This was a major undertaking and my results boil down to a slightly disappointing answer: it is harder than one may hope. First, the songs generated by the neural network did not resemble the songs they were trained on much. This can be improved with more data, better techniques involving a further understanding of music theory, as well as more iterations or more complex and state-of-the-art data science techniques.

# Discussion and Conclusion

Overall, I am very satisfied with the amount that I learned about the field of computational musicology during this project. The wide breadth of topics that I explored in this project has given me the exposure I was hoping for when I first started this program in order to be able to tackle some of these projects in my own time. My understanding of both data science and music has grown tremendously, in the education of best-practices as well as knowledge of what is on the cutting edge of both of these fields in the current day.

I learned that many of these problems that seem like pipe dreams are very possible because they are being done by data scientists and musicians, and the most important thing is to sit down and start working on it. There is so much available open-source from education materials, use cases, tools, and communication, that it is easy to start doing some level of data science with music. That being said, many things that I thought were easy in theory, ended up teaching me a lot about how technically challenging these problems are.

Classification also had many roadblocks with both computing power as well as unlabeled data which was also not highly diverse. Lessons learned were to dive deeper into cutting-edge research papers on the best techniques to tackle this still-unsolved problem, invest in better computing power, and explore an efficient labeling method on the song library.

MIDICSV - The program is a dated open source small project which has no user support - however, it is still one of the best tools openly available even despite the advancement in data science and technology since its development. This perhaps raises the point that perhaps a new software with more capability should be develop. I may pursue this in my leisure time

I would like to create stems of songs by having neural networks detect different instruments and separate those notes into multiple files, one for each instrument (including vocals). This is a large undertaking but I believe the groundwork I have laid with this project and my education from the UCR program will help me begin such a project.

Performing classification on a massive song library using the methods I had attempted requires a tremendous amount of computing performance, particularly memory. With my above-average personal computing memory at 16 MB RAM and an NVIDIA GeForce 860M graphics card, I was not able to even load a matrix with the dimensions of 3571 songs and 294144 time ticks. Either these methods need to be tried with much higher computing power, or much more efficient algorithms, such as ones that utilize distributed computing like MapReduce, GPU CUDA, or sparse matrices. I would like to pursue these methods in the future as they would add too much to the scope of this project.

There is a great deal of work to do to get the Unity program to a much better state, and I think if I pursue development of the program, I can come up with something valuable. I would like to improve it by refining controller input, adding more user control over sounds, displaying more statistics, and developing better visuals.

# References

1. *Midicsv* tool to convert files between .midi and .csv

<http://www.fourmilab.ch/webtools/midicsv/>

1. WaveNet: A Generative Model for Raw Audio

<https://deepmind.com/blog/wavenet-generative-model-raw-audio/>

1. David Cope’s Creative Music Software

<http://artsites.ucsc.edu/faculty/cope/experiments.htm>

1. Music21 Toolkit for Computer-Aided Musicology

<http://web.mit.edu/music21/>

1. Kolmogorov Computational Musicology Algorithm

<http://ctford.github.io/kolmogorov-music/#/>

1. *The Unreasonable Effectiveness of Recurrent Neural Networks* by Andrej Karpathy

<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

1. *Computer evolves to generate baroque music!* by carykh

<https://youtu.be/SacogDL_4JU>

1. *AI evolves to compose 3 hours of jazz!* by carykh

<https://www.youtube.com/watch?v=nA3YOFUCn4U>

1. *This AI can discern between piano and violin with 96% accuracy!* by carykh

<https://youtu.be/n8NhWnAHCG4>

1. Dave’s J.S. Bach Page

<http://www.jsbach.net/midi/>

1. PixelCNN

<https://github.com/openai/pixel-cnn>

1. Synesthesia Game

<http://www.synthesiagame.com/>

1. Independent Component Analysis by Te-Won Lee

<http://www.springer.com/gp/book/9780792382614>

1. Mycelia by Imogen Heap

<http://myceliaformusic.org/>

1. AI Composer Code

<https://github.com/jiexunsee/rudimentary-ai-composer>

1. CReMA

<http://imerc.org/crema>

1. CRAN documentation on R package dtwclust

<https://cran.r-project.org/web/packages/dtwclust/dtwclust.pdf>

1. Comparing Time-Series Clustering Algorithms in R Using the dtwclust Package. Alexis Sard´a-Espinosa.

<https://cran.r-project.org/web/packages/dtwclust/vignettes/dtwclust.pdf>

1. Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping. Keogh, Eamonn, et. al.

<http://www.cs.ucr.edu/~eamonn/SIGKDD_trillion.pdf>

1. Unity - Joystick (gamepad). Omar Sanitago

<https://www.youtube.com/watch?v=3twMmVHpJqI>

1. How to use Joystick controller input ( Joystick ) [Tutorial][C#] - Unity 3d.   
   N3K EN  
   <https://www.youtube.com/watch?v=384_g0f7K_I>
2. Music Animation Machine: Stephen Malinowski at TEDxZurich. TEDx Talks

<https://www.youtube.com/watch?v=EAWSonBN3Pk>

1. R Markdown Documentation

<https://rmarkdown.rstudio.com/authoring_basics.html>

1. Unity 3D Documentation

<https://docs.unity3d.com/ScriptReference/Input.html>

1. The Routledge Companion to Music, Technology, and Education. Andrew King, Evangelos Himonides, S. Alex Ruthmann

[https://books.google.com/books?id=dw30DQAAQBAJ](https://books.google.com/books?id=dw30DQAAQBAJ&printsec=frontcover#v=onepage&q&f=false)

1. How MIDI Works

<https://entertainment.howstuffworks.com/midi2.htm>

1. Statistical Data Analysis R Documentation

<http://stat.ethz.ch/R-manual/>

1. Stack Overflow

<https://stackoverflow.com/>

1. Stack Exchange

<https://stackexchange.com/>

1. MIDI World

<http://www.midiworld.com/files/>

1. “The Largest MIDI Collection on the Internet, collected and sorted diligently by yours truly.” midi\_man, Reddit

<https://www.reddit.com/r/WeAreTheMusicMakers/comments/3ajwe4/the_largest_midi_collection_on_the_internet/>

1. UCR CS235 Data Mining Course Materials. Keogh, Eamonn

<http://www.cs.ucr.edu/~eamonn/CS235/>

1. Ghost in the MP3. Data Skeptic Podcast

<https://dataskeptic.com/blog/episodes/2015/the-ghost-in-the-mp3>

1. “How to write data to a CSV file in UNITY”. Shushanta Chakraborty

<https://sushanta1991.blogspot.com/2015/02/how-to-write-data-to-csv-file-in-unity.html>

1. “C# Programming Basics for Unity Developers Tutorial 14 - Read Data from Text File”. GameDZine

<https://www.youtube.com/watch?v=4hnItE2gZ40>

# 

# 

# Appendix A - Songinfo

Songinfo

Hovanes Keseyan

December 13, 2017

## Songinfo – IDL file

### Data Preparation

#### Load the songs into R

Asks for song locations and loads into R.

songloader <- function()  
{  
 n <- **readline**("How many songs would you like to load? ")  
 n <- **as.integer**(n)  
 if(**is.na**(n)){  
 n <- **songloader**()  
 }  
 wd <- **readline**("Working Directory: ")  
 *#if(is.na(wd)) getwd()*  
 *#else setwd(wd)*  
 f <- **vector**(length = n)  
 v <- **matrix**(nrow = n, ncol = 7)  
 songs <- **list**()  
 for (i in 1:n) {  
 f[i] <- **readline**("File name: ")  
 v[i] <- **readline**("Set variable name: ")  
 songs[[i]] <- **read.csv**(f[i], header=F, row.names = NULL,  
 col.names = **c**("track", "time", "type", 4:7), stringsAsFactors=FALSE)  
 }  
  
 **return**(songs)  
}  
songloader

## function()  
## {  
## n <- readline("How many songs would you like to load? ")  
## n <- as.integer(n)  
## if(is.na(n)){  
## n <- songloader()  
## }  
## wd <- readline("Working Directory: ")  
## #if(is.na(wd)) getwd()  
## #else setwd(wd)  
## f <- vector(length = n)  
## v <- matrix(nrow = n, ncol = 7)  
## songs <- list()  
## for (i in 1:n) {  
## f[i] <- readline("File name: ")  
## v[i] <- readline("Set variable name: ")  
## songs[[i]] <- read.csv(f[i], header=F, row.names = NULL,  
## col.names = c("track", "time", "type", 4:7), stringsAsFactors=FALSE)  
## }  
##   
## return(songs)  
## }

##ask for directory??  
**setwd**("C:/Users/Hovanes/Desktop/UCR/Project/DavesMIDIfiles")  
##ask for filename??  
a = **read.csv**("a.csv", header=F, row.names = NULL, col.names = **c**("track", "time", "type", 4:7), stringsAsFactors=FALSE)

#### Extract data

Start with only the frequency data for now

a\_notes = a[a$type==" Note\_on\_c",]  
**colnames**(a\_notes)=**c**("track", "time", "type", "channel", "note", "velocity")

#### Convert note on c to note off c

Separate the note on c types that have velocity zero (end note) so it's easier to plot

a\_notes\_on = a\_notes[a\_notes$velocity!="0",]  
a\_notes\_off = a\_notes[a\_notes$velocity=="0",]  
  
a\_notes\_first6=a\_notes\_on[a\_notes\_on$track<=6,]  
  
a\_durations <- a\_notes\_off[a\_notes\_off$track<=6, "time"] - a\_notes\_on[a\_notes\_on$track<=6,"time"]  
  
**head**(a\_notes)

## track time type channel note velocity NA  
## 229 2 720 Note\_on\_c 0 62 90 NA  
## 230 2 749 Note\_on\_c 0 62 0 NA  
## 231 2 750 Note\_on\_c 0 62 90 NA  
## 232 2 764 Note\_on\_c 0 62 0 NA  
## 233 2 765 Note\_on\_c 0 62 90 NA  
## 234 2 779 Note\_on\_c 0 62 0 NA

### Song statistics

##### Number of notes: 4272

##### Duration (seconds): 166.35

Number of tracks analyzed: 5

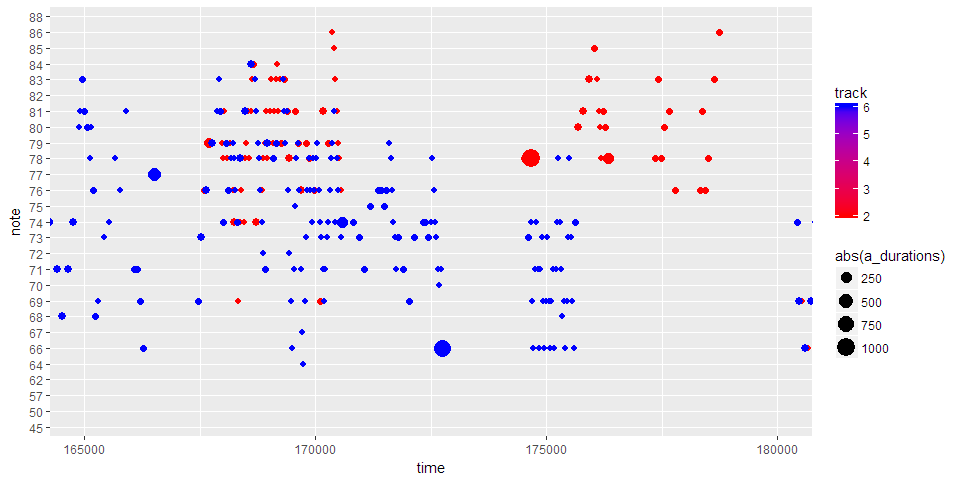
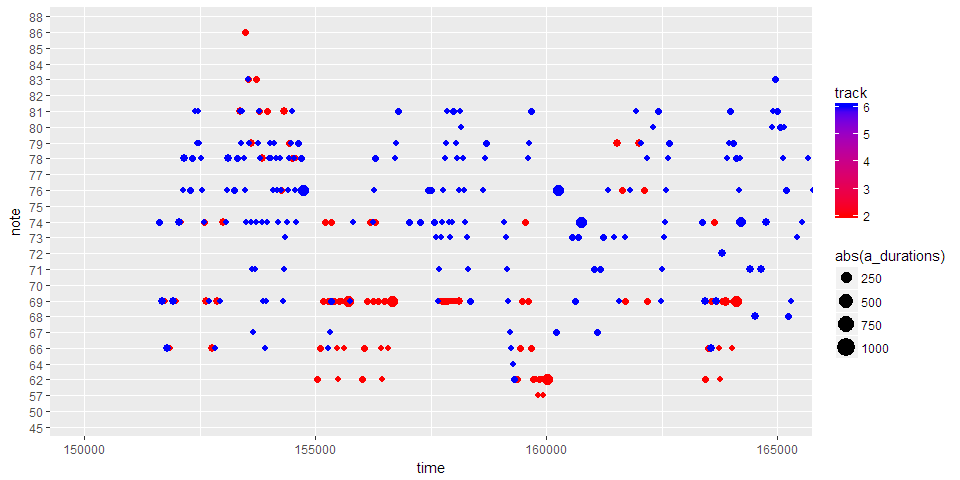
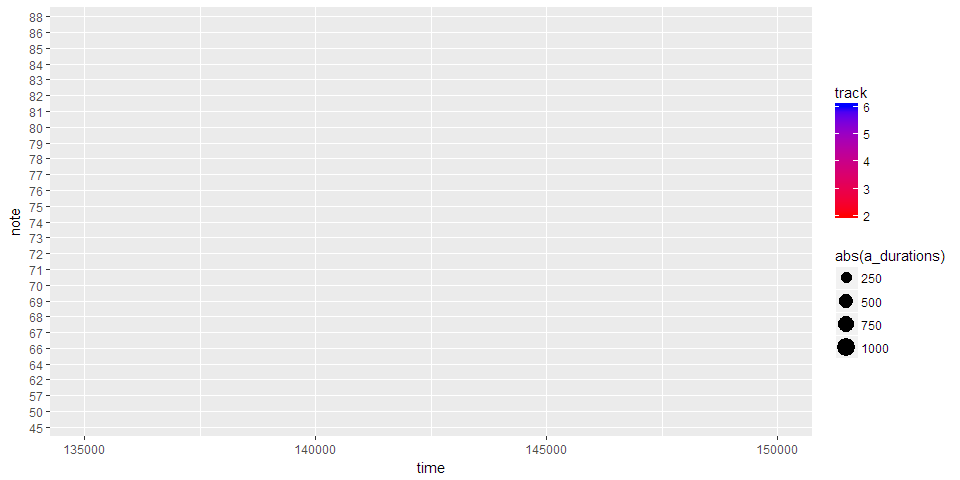
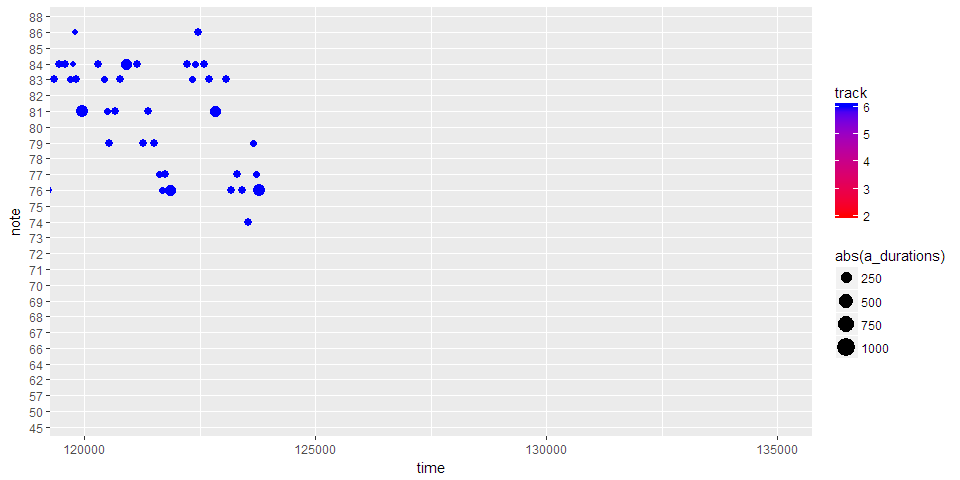
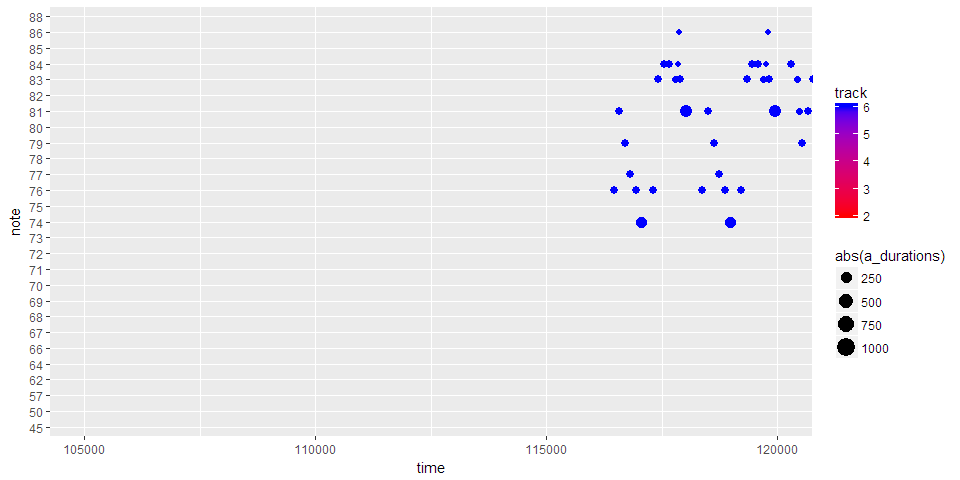
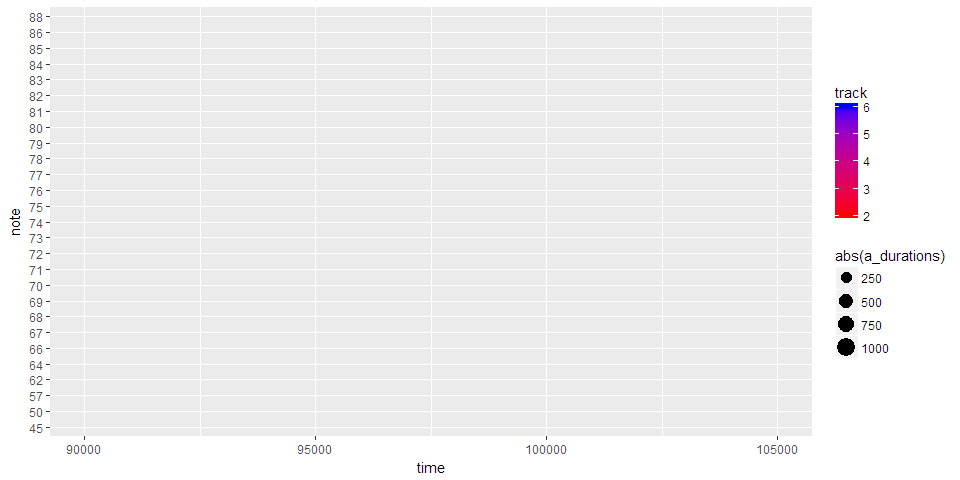
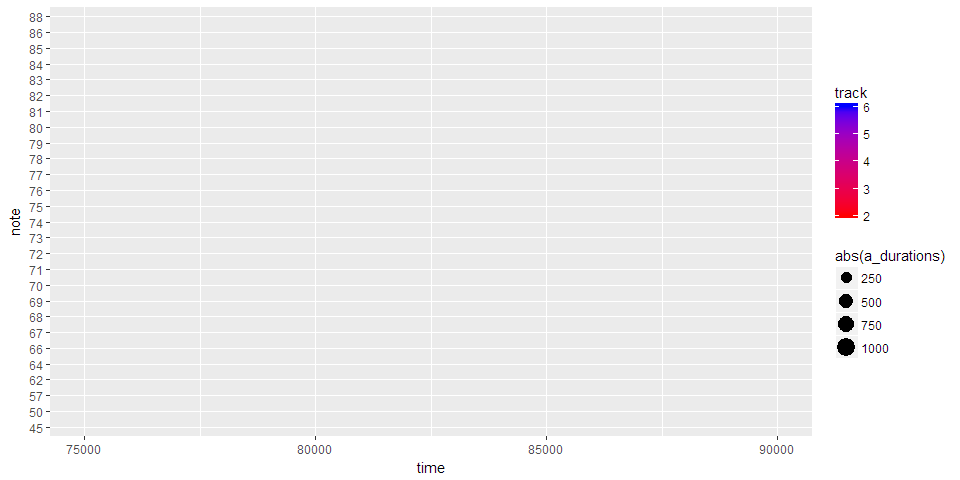
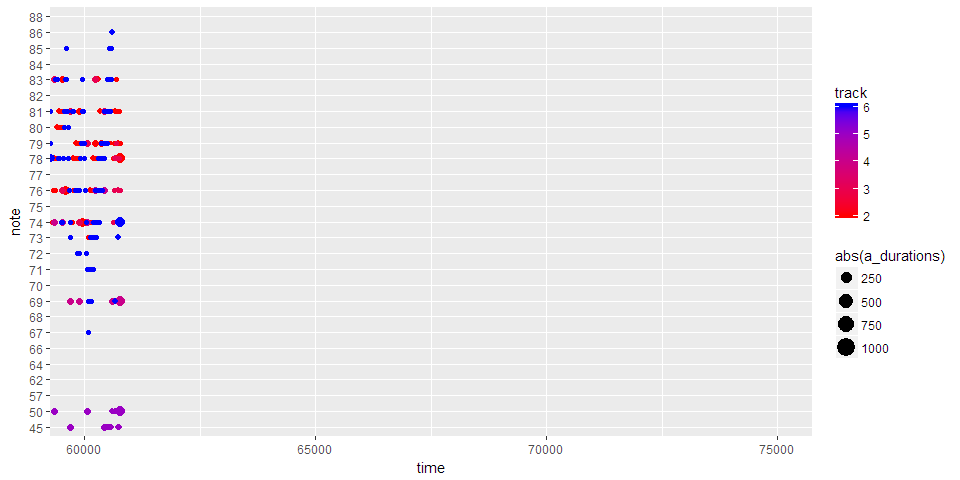
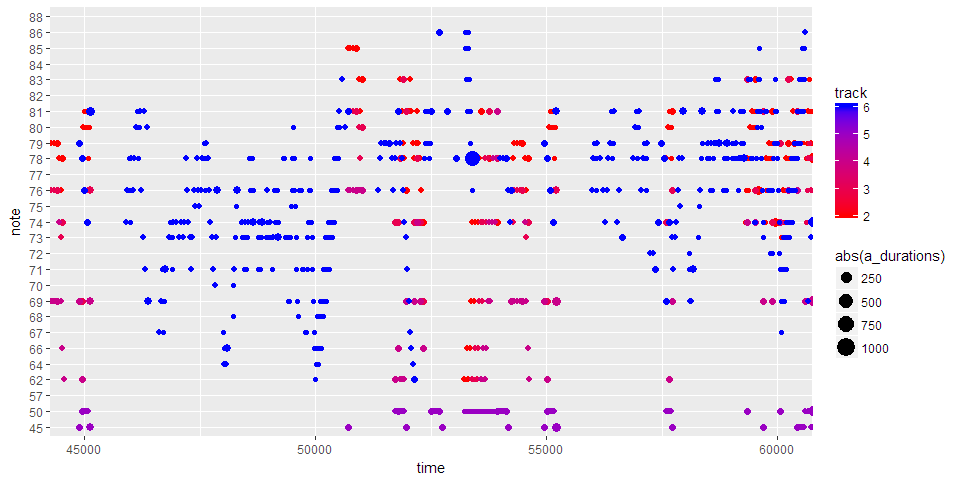
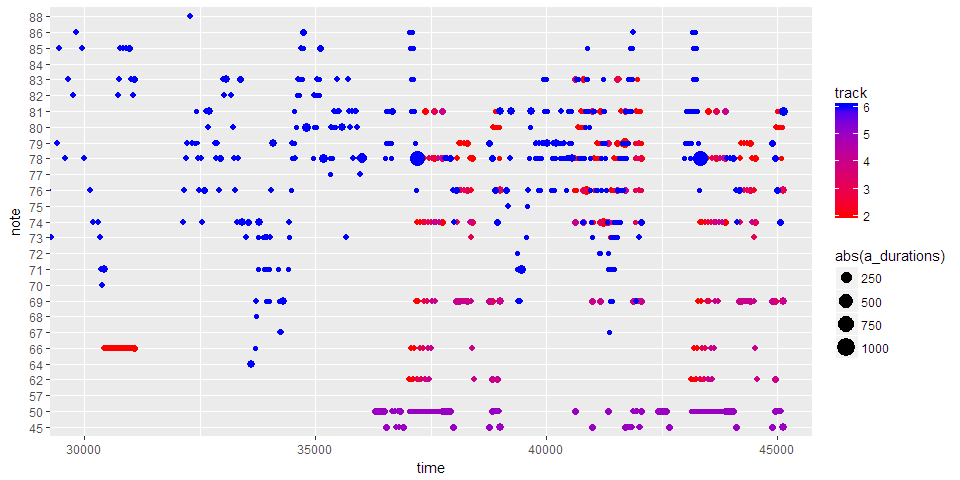
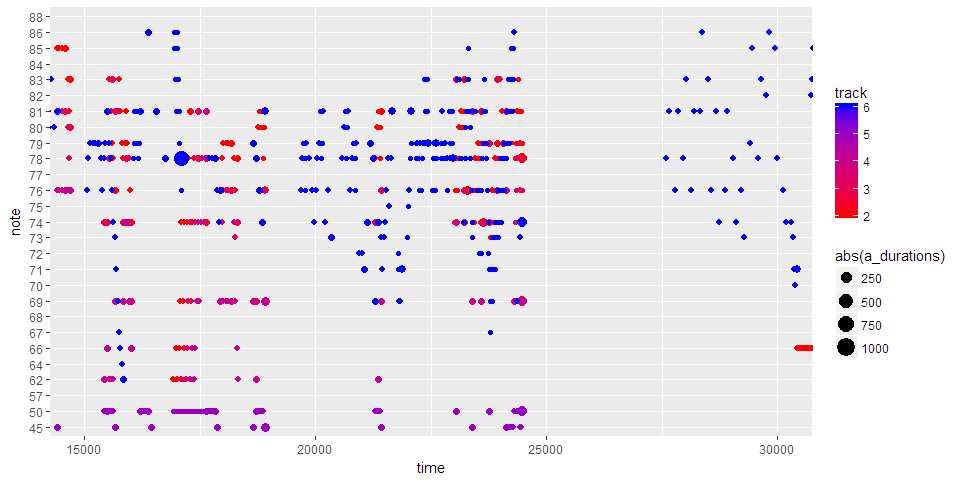
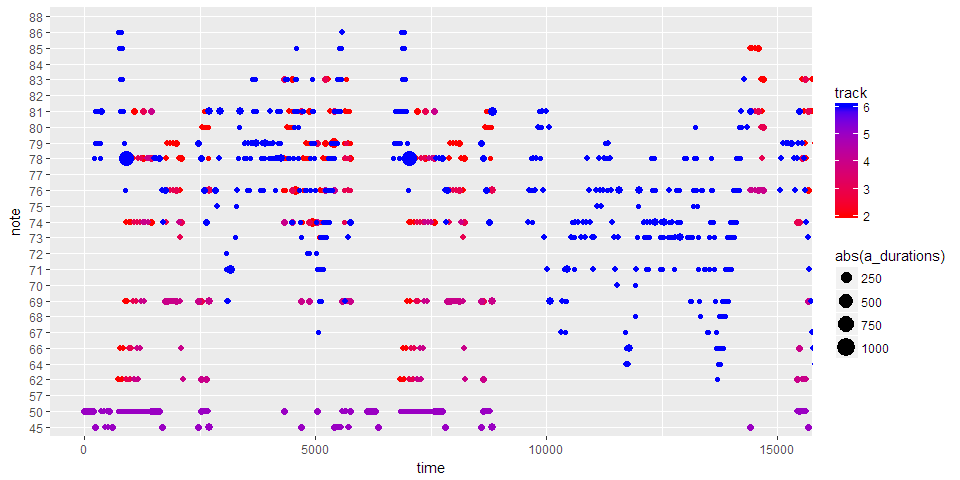
Average number of notes per track: 854.4

Average seconds per note per track: 0.194698

Average notes per minute per track: 308.1695221

##### Derived Tempo: 77.0423805

### Plots of note frequencies



Songinfo

Hovanes Keseyan

December 13, 2017

## Songinfo – RAG file

### Data Preparation

#### Load the songs into R

Asks for song locations and loads into R.

songloader <- function()  
{  
 n <- **readline**("How many songs would you like to load? ")  
 n <- **as.integer**(n)  
 if(**is.na**(n)){  
 n <- **songloader**()  
 }  
 wd <- **readline**("Working Directory: ")  
 *#if(is.na(wd)) getwd()*  
 *#else setwd(wd)*  
 f <- **vector**(length = n)  
 v <- **matrix**(nrow = n, ncol = 7)  
 songs <- **list**()  
 for (i in 1:n) {  
 f[i] <- **readline**("File name: ")  
 v[i] <- **readline**("Set variable name: ")  
 songs[[i]] <- **read.csv**(f[i], header=F, row.names = NULL,  
 col.names = **c**("track", "time", "type", 4:7), stringsAsFactors=FALSE)  
 }  
  
 **return**(songs)  
}  
songloader

## function()  
## {  
## n <- readline("How many songs would you like to load? ")  
## n <- as.integer(n)  
## if(is.na(n)){  
## n <- songloader()  
## }  
## wd <- readline("Working Directory: ")  
## #if(is.na(wd)) getwd()  
## #else setwd(wd)  
## f <- vector(length = n)  
## v <- matrix(nrow = n, ncol = 7)  
## songs <- list()  
## for (i in 1:n) {  
## f[i] <- readline("File name: ")  
## v[i] <- readline("Set variable name: ")  
## songs[[i]] <- read.csv(f[i], header=F, row.names = NULL,  
## col.names = c("track", "time", "type", 4:7), stringsAsFactors=FALSE)  
## }  
##   
## return(songs)  
## }

##ask for directory??  
**setwd**("C:/Users/Hovanes/Desktop/UCR/Project")  
##ask for filename??  
a = **read.csv**("createdsong.csv", header=F, row.names = NULL, col.names = **c**("track", "time", "type", 4:7), stringsAsFactors=FALSE)

#### Extract data

Start with only the frequency data for now

a\_notes = a[a$type==" Note\_on\_c",]  
**colnames**(a\_notes)=**c**("track", "time", "type", "channel", "note", "velocity")

#### Convert note on c to note off c

Separate the note on c types that have velocity zero (end note) so it's easier to plot

a\_notes\_on = a\_notes[a\_notes$velocity!="0",]  
a\_notes\_off = a\_notes[a\_notes$velocity=="0",]  
  
a\_notes\_first6=a\_notes\_on[a\_notes\_on$track<=6,]  
  
a\_durations <- a\_notes\_off[a\_notes\_off$track<=6, "time"] - a\_notes\_on[a\_notes\_on$track<=6,"time"]  
  
**head**(a\_notes)

## track time type channel note velocity NA  
## 3 2 24 Note\_on\_c 2 31 90 NA  
## 4 2 47 Note\_on\_c 2 31 0 NA  
## 5 2 48 Note\_on\_c 2 56 90 NA  
## 6 2 59 Note\_on\_c 2 56 0 NA  
## 7 2 60 Note\_on\_c 2 64 90 NA  
## 8 2 71 Note\_on\_c 2 64 0 NA

### Song statistics

##### Number of notes: 393

##### Duration (seconds): 4.623

Number of tracks analyzed: 5

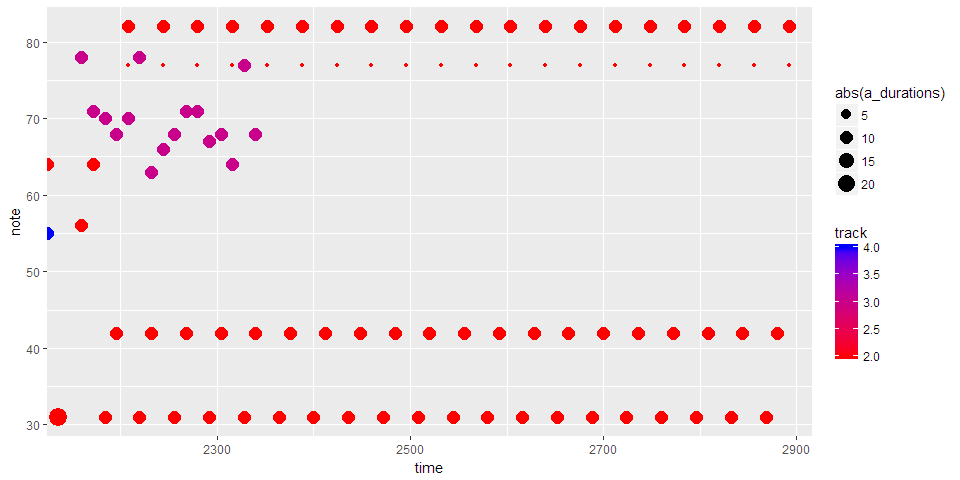
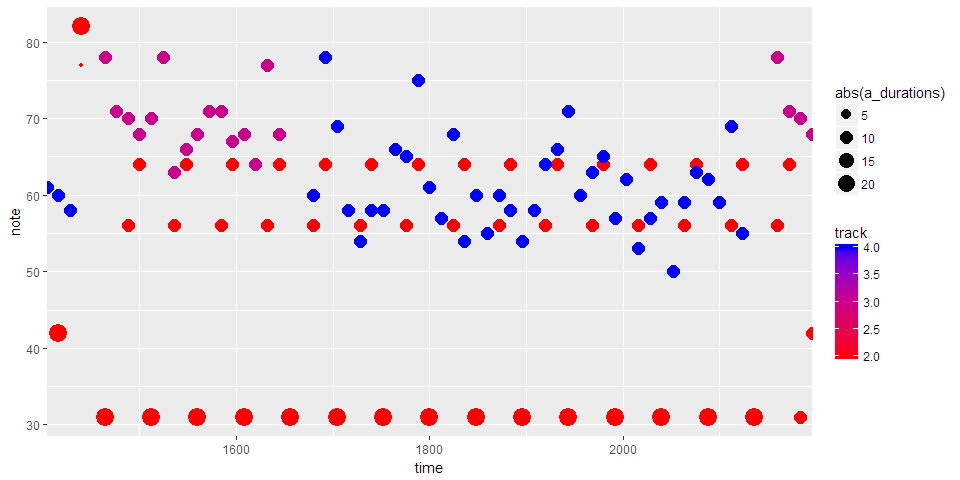
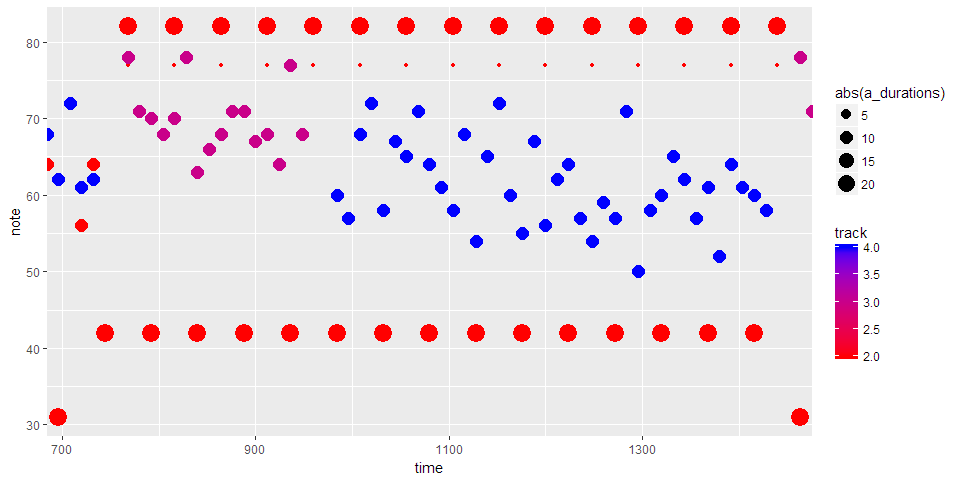
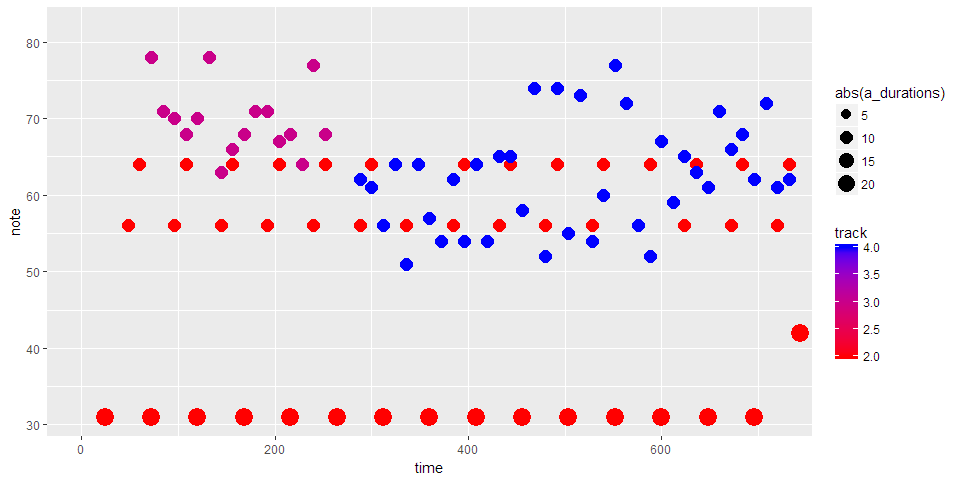
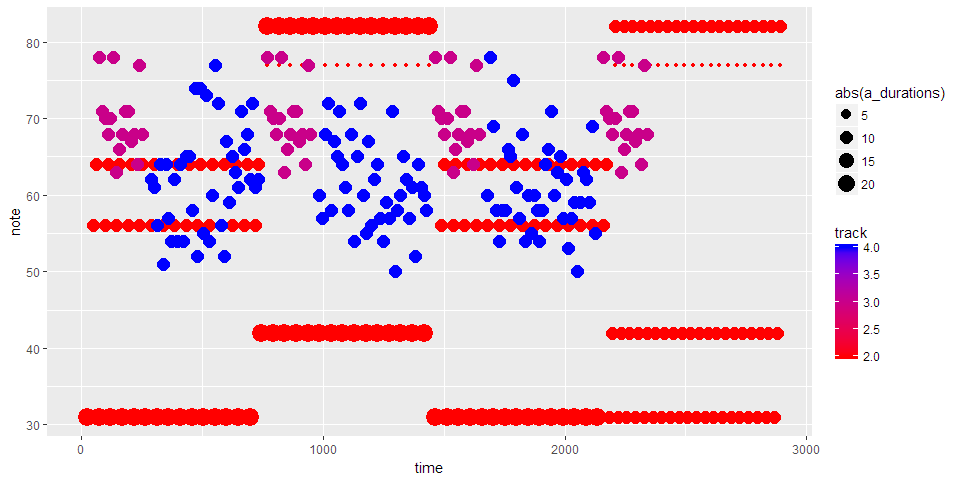
Average number of notes per track: 78.6

Average seconds per note per track: 0.0588168

Average notes per minute per track: 1020.1168073

##### Derived Tempo: 255.0292018

### Plots of note frequencies



# Appendix B – R Generation Algorithm

# Initialize column headers, newline vector, tempo, and note lengths

```{r initialize}

x <- data.frame(0, 0, "Header", 1, NA, NA, stringsAsFactors=FALSE)

y <- data.frame(0, 0, "0", 0, 0, 0, 0, stringsAsFactors=FALSE)

names(x) <- c("track", "time", "event", "channel", "note", "velocity")

names(y) <- names(x)

# 24 MIDI clock units per second

mcu <- 24

# current time

t <- x[nrow(x),2]

# song length (in seconds)

l <- 120

# song length (in MIDI clock units)

l <- l\*mcu

# tempo in beats per minute

tempo <- 120

# note lengths in MIDI clock units based on tempo

f <- tempo\*mcu/60-1

h <- tempo\*mcu/60/2-1

q <- tempo\*mcu/60/4-1

```

# Create rhythm from simple repeating beats

```{r rhythm}

# rhythm is on track 2

tr <- 2

y <- c(tr, 24, "Start\_track", NA, NA, NA); x <- rbind(x, y)

t <- 24

# identify notes for certain sounds

bl <- 31 # low bass

bm <- 42 # mid bass

sm <- 56 # mid snare

sh <- 77 # high snare

cm <- 64 # mid cymbal

ch <- 82 # high cymbal

# basic repeating beat

while (t < l/4) {

y <- c(tr, t, "Note\_on\_c", tr, bl, 90); x <- rbind(x, y); t<-t+h

y <- c(tr, t, "Note\_on\_c", tr, bl, 0); x <- rbind(x, y); t<-t+1

y <- c(tr, t, "Note\_on\_c", tr, sm, 90); x <- rbind(x, y); t<-t+q

y <- c(tr, t, "Note\_on\_c", tr, sm, 0); x <- rbind(x, y); t<-t+1

y <- c(tr, t, "Note\_on\_c", tr, cm, 90); x <- rbind(x, y); t<-t+q

y <- c(tr, t, "Note\_on\_c", tr, cm, 0); x <- rbind(x, y); t<-t+1

}

# switch the beat

while (t < 2\*l/4) {

y <- c(tr, t, "Note\_on\_c", tr, bm, 90); x <- rbind(x, y); t<-t+h

y <- c(tr, t, "Note\_on\_c", tr, bm, 0); x <- rbind(x, y); t<-t+1

y <- c(tr, t, "Note\_on\_c", tr, sh, 90); x <- rbind(x, y); t<-t-1

y <- c(tr, t, "Note\_on\_c", tr, sh, 0); x <- rbind(x, y); t<-t+1

y <- c(tr, t, "Note\_on\_c", tr, ch, 90); x <- rbind(x, y); t<-t+h

y <- c(tr, t, "Note\_on\_c", tr, ch, 0); x <- rbind(x, y); t<-t+1

}

# switch back

while (t < 3\*l/4) {

y <- c(tr, t, "Note\_on\_c", tr, bl, 90); x <- rbind(x, y); t<-t+h

y <- c(tr, t, "Note\_on\_c", tr, bl, 0); x <- rbind(x, y); t<-t+1

y <- c(tr, t, "Note\_on\_c", tr, sm, 90); x <- rbind(x, y); t<-t+q

y <- c(tr, t, "Note\_on\_c", tr, sm, 0); x <- rbind(x, y); t<-t+1

y <- c(tr, t, "Note\_on\_c", tr, cm, 90); x <- rbind(x, y); t<-t+q

y <- c(tr, t, "Note\_on\_c", tr, cm, 0); x <- rbind(x, y); t<-t+1

}

# aggressive beat

while (t < 4\*l/4) {

y <- c(tr, t, "Note\_on\_c", tr, bl, 90); x <- rbind(x, y); t<-t+q

y <- c(tr, t, "Note\_on\_c", tr, bl, 0); x <- rbind(x, y); t<-t+1

y <- c(tr, t, "Note\_on\_c", tr, bm, 90); x <- rbind(x, y); t<-t+q

y <- c(tr, t, "Note\_on\_c", tr, bm, 0); x <- rbind(x, y); t<-t+1

y <- c(tr, t, "Note\_on\_c", tr, sh, 90); x <- rbind(x, y); t<-t-1

y <- c(tr, t, "Note\_on\_c", tr, sh, 0); x <- rbind(x, y); t<-t+1

y <- c(tr, t, "Note\_on\_c", tr, ch, 90); x <- rbind(x, y); t<-t+q

y <- c(tr, t, "Note\_on\_c", tr, ch, 0); x <- rbind(x, y); t<-t+1

}

y <- c(tr, t, "End\_track", NA, NA, NA)

x <- rbind(x, y)

```

# Create chorus

```{r chorus}

#chorus is on track 3

tr <-3

y <- c(tr, 0, "Start\_track", NA, NA, NA); x <- rbind(x, y)

# length of chorus

lc <- 8\*mcu

# length of each verse

lv <- 21\*mcu

# time at which first chorus starts

tcf <- 72

# number of choruses

nc <- (l-tcf)/(lc+lv)

n <- seq(0, nc)

# times at which chorus starts

tc <- rep(0, nc)

tc <- tcf + n\*(lc+lv)

# generate set of random notes which will be the same chorus

rc <- round(rnorm(lc/(q+1), 70, 4),0)

# chorus generating script

for (i in 1:nc) {

t <- tc[i]

for (j in 1:(lc/(q+1))) {

y <- c(tr, t, "Note\_on\_c", tr, rc[j], 90); x <- rbind(x, y); t<-t+q

y <- c(tr, t, "Note\_on\_c", tr, rc[j], 0); x <- rbind(x, y); t<-t+1

}

}

y <- c(tr, t, "End\_track", NA, NA, NA); x <- rbind(x, y)

```

# Create verse using randomization

```{r verses}

# verses are on track 4

tr <- 4

y <- c(tr, 0, "Start\_track", NA, NA, NA); x <- rbind(x, y)

# number of verses

nv <- nc-1

# times at which verse starts

tv <- tc[1:nv] + lc + mcu # start verse a second after chorus ends

# verse generating script

for (i in 1:nv) {

t <- tv[i]

while (t < tv[i]+lv-2\*mcu) { # end verse 2 seconds before chorus starts

# generate random note every time

rv <- round(rnorm(1, 62, 6),0)

y <- c(tr, t, "Note\_on\_c", tr, rv, 90); x <- rbind(x, y); t<-t+q

y <- c(tr, t, "Note\_on\_c", tr, rv, 0); x <- rbind(x, y); t<-t+1

}

}

y <- c(tr, t, "End\_track", NA, NA, NA); x <- rbind(x, y)

```

# Sort by duration and save as csv

```{r export}

# sort

# add end of file row

y <- c(0, 0, "End\_of\_file", NA, NA, NA); x <- rbind(x, y)

head(x)

tail(x)

# save

x[c(1:2, 4:6)] <- sapply(x[c(1:2, 4:6)], as.numeric)

write.table(x, file = "createdsong.csv", row.names=F, col.names=F, quote=F, sep=", ", na=" ")

```

# Appendix C – Compression Algorithm

```{r songnames, include=F}

# Access Repository for Song Names and Load Songs to R

setwd("E:/DATA/CSV/wd") # set working directory to location of csv files

songnames = list.files(pattern="\*.csv") # retrieve song names from the working directory

length(songnames) # check number of songs loaded

songs <- list()

for (i in 1:length(songnames)) songs[[i]] <- read.csv(songnames[i]) # load every song and assign the song name to the name of the data set

object.size(songs) # size of uncompressed songs in memory

names(songs) <- songnames

```

```{r songscompressed, include=F}

# Compress Songs

songscompressed <- lapply(songs, function(x){x[x[,2]=="Note On",c(5,7)]})

object.size(songscompressed) # size of compressed songs in memory

setwd("E:/DATA/CSV/wdcompressed") # set working directory to location where compressed files will write

lapply(1:length(songscompressed), function(i) write.csv(songscompressed[[i]],

file = paste0(names(songscompressed[i]), ".csv"),

row.names = FALSE))

```

Number of Songs in Repository: `r length(songnames)`

\*\*Uncompressed\*\* memory allocation: \*\*`r round(object.size(songs)/1048576,1)` MB\*\*

\*\*Compressed\*\* memory allocation: \*\*`r round(object.size(songscompressed)/1048576,1)` MB\*\*

The compressed repository is \*\*`r round(object.size(songscompressed)/object.size(songs) \* 100, 2)`%\*\* of the size of the uncompressed repository.

```{r plots, echo=F, warning=F, message=F}

stages <- c("Uncompressed", "Compressed")

mem <- c(object.size(songs)/1048576, object.size(songscompressed)/1048576)

dat <- data.frame(stages, mem)

require(ggplot2)

require(methods)

plot <- ggplot(dat, aes(stages,mem)) + geom\_bar(aes(fill=stages), stat="identity") + ggtitle("Effectiveness") + ylab("Memory Allocated (MB)") + xlab("Stage")

plot

```

# Appendix D – Unity Code

## Controller Inputs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Controller\_input : MonoBehaviour {

public bool isButton = true;

public bool leftJoystick = false;

public bool rightJoystick = false;

public bool dPad = false;

public bool triggers = false;

public string buttonName;

private Vector3 startPos;

private Transform thisTransform;

private MeshRenderer mr;

public Color restColor = new Color(0.1F, 0.1F, 0.7F, 0.8F);

public Color activeColor = new Color(0.7F, 0.1F, 0.1F, 0.2F);

public float restR2;

public float restL2;

// Use this for initialization

void Start () {

thisTransform = transform;

startPos = thisTransform.position;

mr = thisTransform.GetComponent<MeshRenderer>();

mr.material.color = restColor;

//restR2 = Input.GetAxis("R2");

// restL2 = Input.GetAxis("L2");

}

// Update is called once per frame

void Update () {

if (isButton)

{

if (triggers)

{

bool triggerPulled = false;

/\*if (Input.GetAxis("R2") == 0)

{

triggerPulled = true;

mr.enabled = triggerPulled;

mr.material.color = activeColor;

}\*/

if (Input.GetAxis("L2") < 0 || Input.GetAxis("R2") < 0)

{

triggerPulled = true;

mr.enabled = triggerPulled;

mr.material.color = activeColor;

}

else

{

mr.enabled = true;

mr.material.color = restColor;

}

}

else

{

mr.enabled = Input.GetButton(buttonName);

if (mr.enabled)

{

mr.material.color = activeColor;

}

else if (!leftJoystick && !rightJoystick)

{

mr.enabled = true;

mr.material.color = restColor;

}

}

}

else

{

if (leftJoystick)

{

Vector3 inputDirection = Vector3.zero;

inputDirection.x = Input.GetAxis("Lx");

inputDirection.y = -Input.GetAxis("Ly");

thisTransform.position = startPos + inputDirection;

}

else if (rightJoystick)

{

Vector3 inputDirection = Vector3.zero;

inputDirection.x = Input.GetAxis("Rx");

inputDirection.y = -Input.GetAxis("Ry");

thisTransform.position = startPos + inputDirection;

}

else if (dPad)

{

Vector3 inputDirection = Vector3.zero;

inputDirection.x = Input.GetAxis("Dx");

inputDirection.y = Input.GetAxis("Dy");

thisTransform.position = startPos + 0.2F \* inputDirection;

}

else if (triggers)

{

Vector3 inputDirection = Vector3.zero;

inputDirection.x = Input.GetAxis("Triggers");

thisTransform.position = startPos + inputDirection;

}

}

}

}

## Call to Play Note

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Play\_Note : MonoBehaviour {

public string buttonName;

public double noteValue = 1;

public int buttonValue = 1;

public double noteMult = 1;

public static bool canPlay = true;

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

canPlay = true;

Vector3 LJS = Vector3.zero;

LJS.x = Input.GetAxis("Lx");

LJS.y = -Input.GetAxis("Ly");

Vector3 RJS = Vector3.zero;

RJS.x = Input.GetAxis("Rx");

RJS.y = -Input.GetAxis("Ry");

noteValue = Math.Sign(RJS.y\*RJS.x)\*Math.Round(Math.Sqrt(Math.Pow(2\*RJS.y, 2) + Math.Pow(RJS.x, 2)));

noteMult = Math.Sign(LJS.y\*LJS.x) \* Math.Round(Math.Sqrt(Math.Pow(LJS.y, 2) + Math.Pow(LJS.x, 2))+.5);

if (Input.GetButtonDown("Y"))

{

buttonValue = 4;

PlayedNote.playedNote = ActiveNote.activeNote + noteMult \* buttonValue;

}

if (Input.GetButtonDown("B"))

{

buttonValue = 3;

PlayedNote.playedNote = ActiveNote.activeNote + noteMult \* buttonValue;

}

if (Input.GetButtonDown("A"))

{

buttonValue = 1;

PlayedNote.playedNote = ActiveNote.activeNote + noteMult \* buttonValue;

}

if (Input.GetButtonDown("X"))

{

buttonValue = 2;

PlayedNote.playedNote = ActiveNote.activeNote + noteMult \* buttonValue;

}

if (RJS.y == 0 && RJS.x == 0 && LJS.y == 0 && LJS.x == 0)

{

PlayedNote.playedNote = null;

}

else if (canPlay)

{

PlayedNote.playedNote = ActiveNote.activeNote + noteValue \* noteMult + buttonValue \* noteMult;

}

noteMult = 0;

noteValue = 0;

buttonValue = 0;

canPlay = false;

}

}

## CSV Manager

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using System.IO;

using System.Text;

using System;

using System.Linq;

public class CSVManager : MonoBehaviour {

public List<int[]> input = new List<int[]>();

public int t = 0;

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

if (Input.GetButtonDown("Back"))

{

SaveToFile(PlayedNote.log);

}

if (Input.GetButtonDown("Start"))

{

ReadFromCreated();

}

PlayInput();

}

public void SaveToFile(List<string[]> log)

{

string[][] output = new string[log.Count][];

for (int i = 0; i < output.Length; i++)

{

output[i] = log[i];

}

int length = output.GetLength(0);

string delimiter = ",";

StringBuilder sb = new StringBuilder();

for (int index = 0; index < length; index++)

sb.AppendLine(string.Join(delimiter, output[index]));

string filePath = "C:/Users/Hovanes/Desktop/UCR/Project/UCG/ucg1.csv";

StreamWriter outStream = System.IO.File.AppendText(filePath);

outStream.WriteLine(sb);

outStream.Close();

}

public void ReadFromCreated()

{

string filePath = "C:/Users/Hovanes/Desktop/UCR/Project/createdsong.csv";

StreamReader reader = new StreamReader(@filePath);

string s = reader.ReadLine();

int result;

while (s != null)

{

string[] values = s.Split(',');

int[] line = new int[2];

if (Int32.TryParse(values[1], out result)) { line[0] = result; } else { line[0] = 0; };

if (Int32.TryParse(values[4], out result)) { line[1] = result; } else { line[1] = 0; };

if (Int32.TryParse(values[5], out result)) { if (result > 0) { input.Add(line); } };

}

}

public void PlayInput()

{

// sort by time ?

for (int i = 0; i < input.Count; i++) {

if (input[i][0] >= t && input[i][0] < t + 3)

{

PlayedNote.playedNote = input[i][1];

t += 3;

}

}

}

}

# Appendix E – Python Neural Net

from mido import MidiFile, MidiTrack, Message  
from keras.layers import LSTM, Dense, Activation, Dropout  
from keras.preprocessing import sequence  
from keras.models import Sequential  
from keras.optimizers import RMSprop  
from sklearn.preprocessing import MinMaxScaler  
import numpy as np  
import mido  
  
########### PROCESS MIDI FILE #############  
mid = MidiFile('allegroconspirito.mid') # a Mozart piece  
  
notes = []  
  
time = float(0)  
prev = float(0)  
  
for msg in mid:  
 ### this time is in seconds, not ticks  
 time += msg.time  
 if not msg.is\_meta:  
 ### only interested in piano channel  
 if msg.channel == 0:  
 if msg.type == 'note\_on':  
 # note in vector form to train on  
 note = msg.bytes()  
 # only interested in the note and velocity. note message is in the form of [type, note, velocity]  
 note = note[1:3]  
 note.append(time-prev)  
 prev = time  
 notes.append(note)  
###########################################  
  
######## SCALE DATA TO BETWEEN 0, 1 #######  
t = []  
for note in notes:  
 note[0] = (note[0]-24)/88  
 note[1] = note[1]/127  
 t.append(note[2])  
max\_t = max(t) # scale based on the biggest time of any note  
for note in notes:  
 note[2] = note[2]/max\_t  
###########################################  
  
############ CREATE DATA, LABELS ##########  
X = []  
Y = []  
n\_prev = 30  
# n\_prev notes to predict the (n\_prev+1)th note  
for i in range(len(notes)-n\_prev):  
 x = notes[i:i+n\_prev]  
 y = notes[i+n\_prev]  
 X.append(x)  
 Y.append(y)  
# save a seed to do prediction later  
seed = notes[0:n\_prev]  
###########################################  
  
############### BUILD MODEL ###############  
print('Build model...')  
model = Sequential()  
model.add(LSTM(128, input\_shape=(n\_prev, 3), return\_sequences=True))  
model.add(Dropout(0.2))  
model.add(LSTM(64, input\_shape=(n\_prev, 3), return\_sequences=False))  
model.add(Dropout(0.2))  
model.add(Dense(3))  
model.add(Activation('linear'))  
  
optimizer = RMSprop(lr=0.01)  
model.compile(loss='mse', optimizer='rmsprop')  
model.fit(X, Y, batch\_size=300, epochs=400, verbose=1)  
###########################################  
  
############ MAKE PREDICTIONS #############  
prediction = []  
x = seed  
x = np.expand\_dims(x, axis=0)  
  
for i in range(3000):  
 preds = model.predict(x)  
 print (preds)  
 x = np.squeeze(x)  
 x = np.concatenate((x, preds))  
 x = x[1:]  
 x = np.expand\_dims(x, axis=0)  
 preds = np.squeeze(preds)  
 prediction.append(preds)  
  
for pred in prediction:  
 pred[0] = int(88\*pred[0] + 24)  
 pred[1] = int(127\*pred[1])  
 pred[2] \*= max\_t  
 # to reject values that will be out of range  
 if pred[0] < 24:  
 pred[0] = 24  
 elif pred[0] > 102:  
 pred[0] = 102  
 if pred[1] < 0:  
 pred[1] = 0  
 elif pred[1] > 127:  
 pred[1] = 127  
 if pred[2] < 0:  
 pred[2] = 0  
###########################################  
  
###### SAVING TRACK FROM BYTES DATA #######  
mid = MidiFile()  
track = MidiTrack()  
mid.tracks.append(track)  
  
for note in prediction:  
 # 147 means note\_on  
 note = np.insert(note, 0, 147)  
 bytes = note.astype(int)  
 print (note)  
 msg = Message.from\_bytes(bytes[0:3])  
 time = int(note[3]/0.001025) # to rescale to midi's delta ticks. arbitrary value for now.  
 msg.time = time  
 track.append(msg)  
  
mid.save('new\_song.mid')  
###########################################