EXPLORING GENERATED MUSIC FROM A SONG SEED: TAYLOR SWIFT'S NEXT HIT?

A Thesis

Submitted to the School of Graduate Studies and Research

in Partial Fulfillment of the

Requirements for the Degree

Master of Science

Morgan Buterbaugh

Indiana University of Pennsylvania

May 2024

Indiana University of Pennsylvania
School of Graduate Studies and Research
Department of Mathematics

We hereby approve the thesis of

Morgan Buterbaugh

Candidate for the degree of Master of Science

3/29/24

John Chrispell, Ph. D.
Professor of Mathematics and Computer Science,
Advisor

3/29/2024

Rick Adkins, Ph. D.
Professor of Mathematics and Computer Science

3/29/24

Sam Grieggs, Ph. D.
Assistant Professor of Computer Science

ACCEPTED

Approval on File

Hilliary E. Creely, Ph.D.
Dean
School of Graduate Studies and Research

Title: Exploring Generated Music From a Song Seed: Taylor Swift's Next Hit?

Author: Morgan Buterbaugh

Thesis Chair: Dr. John Chrispell

Thesis Committee Members: Dr. Rick Adkins
Dr. Sam Grieggs

As machine learning continues to grow in today's world, some people may wonder what the future of music and song-making looks like. Can machines and data science be used to write melodies for a new song based on a given song? Employing a computational framework built in Python, exploring how music can be generated using a stochastic approach and basic machine learning to generate music from a random song seed using a stochastic process could be one possible solution. A seeding piece of music will be analyzed and used to create a Markov Chain stochastic process that composes a new originally unique musical piece. Notes and rests will be used as states, and the probability of the following state will be estimated from the seeding composition. The modeling framework will be allowed to compose its own music using stochastic state transitions by implementing a random initialization to set a starting state. Will this process produce a melody as catchy and successful as original pieces of work when using music by one of the top artists in the world, Taylor Swift?

ACKNOWLEDGMENTS

I would first like to give thanks to my thesis advisor, Dr. Chrispell, for all his help and motivation. Without him, my research would not have been as successful and far along. To all of my friends, Jaylee, Toby, Josh, Isabelle, Casey, Parker, and Ylenia, thank you for helping me in more ways than you know. From listening to me complain, trying their best to help me, giving me suggestions, and being great friends along this journey, I am forever grateful. Most importantly, I would like to thank my mum for always supporting me in everything I do. She is, and always has been, my biggest supporter, even though she had no clue as to what I was doing. A special thanks to my dog, Marlee May, for sitting on my lap and beside me while I worked on everything throughout the whole process. Lastly, of course, Taylor Swift. Her music has always been there for me and I am proud to call myself a Swiftie!

TABLE OF CONTENTS

CHAPTER Page

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

In today's world, technology is an ever-growing field. The utilization of computers, algorithms, and machine learning evolves daily, creating solutions for everyday problems. When it comes to composing music for a movie, video game, personal liking, or with the goal in mind of creating the next biggest hit, we can take advantage of these technological advancements. Although creating a piece of music in the past has always been done by a human, it is now possible to let a machine do the work. Generating a new song based on your favorite song, creating a piece of work that sounds similar to your favorite artist, or even creating a new song altogether are just a few possibilities. Regardless of what is being created, some sort of mathematics and mathematical processes are involved, whether it be on the front end where the user interacts with it, or working quietly in the background.

One approach to exploring computer generative music involves employing supervised stochastic machine learning. Using Python, the objective is to generate a song from a specific seeding piece of music by implementing a Markov Chain. By harnessing probabilities and randomness, this model facilitates the creation of a one-of-a-kind musical arrangement derived from a random song seed.

Machine Learning

Machine learning (ML) is a branch of artificial intelligence (AI) that uses algorithms and data sets to learn and create models. ML uses computer algorithms and data to improve through experiences, learn how to perform tasks that are difficult for humans to do, and learn without explicitly being programmed [8]. Although ML and AI are commonly interchanged, ML refers to the use of data and algorithms to create self-learning models that can predict certain outcomes, whereas AI refers to the attempt to create a machine with human-level cognitive abilities [5].

There are four different types of ML, supervised, unsupervised, semi-supervised, and reinforcement. Supervised learning models are trained from labeled datasets to classify data or accurately predict outcomes [4]. These types of models learn and become more accurate over time. Unsupervised learning models learn from patterns or trends, using ML algorithms to analyze and cluster unlabeled datasets [4]. The main goal of an unsupervised learning model is to be able to identify meaningful and useful patterns among the unlabeled dataset [3]. Semi-supervised learning uses both labeled and unlabeled data, making this a mix of both supervised and non-supervised learning. During the training process, a small amount of labeled data is used as a guide to classify and extract features from larger unlabeled data [5]. Reinforcement learning models are trained through the process of trial and error [8]. The model makes predictions by either getting a reward or a penalty, leading to the development of the best policy that gives the most rewards [3]. This project emphasizes the utilization of Python to implement supervised ML, since the user is in control of the models' input.

Stochastic Processes

Stochastic can be defined as a process that involves chance and probability [1]. There are two types of stochastic processes, discrete-time and continuous-time.

Observe a system's characteristic at discrete points in time t, and let X_t be the value of a specific system characteristic at time t. Generally, X_t is not known before time t. This is known as a stochastic process. A discrete-time stochastic process is a description of the relation between the random variables X_i for $i \in \{0, 1, 2, \dots\}$. A continuous-time stochastic process is one in which a state of the system can be viewed at any time, not just at a discrete instance in time [27]. In terms of music theory, stochastic music uses stochastic processes to generate elements that randomly stem from a mathematical process [1].

Markov Chain's

A special kind of discrete-time stochastic process is a Markov Chain. By definition, it is assumed that "at any time t, the discrete-time stochastic process can be in any one of S distinct discrete states labeled 1, 2, ... , s [27]." Thus, a discrete-time stochastic process is a Markov Chain if for $t = 0, 1, 2, \ldots$ and all states,

$$P(X_{t+1} = i_{t+1} | X_t = i_t, X_{t-1} = i_{t-1}, \ldots, X_1 = i_1, X_0 = i_0) = P(X_{t+1} = i_{t+1} | X_t = i_t).$$

Thus, the probability of the state at time $t = 1$ is dependent only on the previous state at time t, and not all the other previous states before time t that the chain passed through.

The assumption is made that for all states i and j and all times t, $P(X_{t+1} = j \mid X_t = i)$ is independent of t. Then,

$$P(X_{t+1} = j | X_t = i) = p_{ij},$$

where p_{ij} denotes the probability of a system that is in state j at time $t + 1$, given it was in state i at time t. Moving from state i to state j is called a transition, and the values of p_{ij} are known as the transition probabilities of the Markov Chain. The probabilities of transitioning from state i to state j do not vary with time [27].

The probability that the Markov Chain is in state i at time $t = 0$, given by q_i, is $P(X_0 = i) = q_i$. The initial probability distribution for the Markov Chain is the vector $\mathbf{q} = [q_1 \;\; q_2 \;\; \cdots \;\; q_s]$, and the transition probability matrix, P, is displayed as an $s \times s$ matrix,

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1s} \\ p_{21} & p_{22} & \cdots & p_{2s} \\ \vdots & \vdots & & \vdots \\ p_{s1} & p_{s2} & \cdots & p_{ss} \end{bmatrix},$$

where P must be non-negative since it is the transition probabilities. This transition matrix gives the probabilities for each transition between states at two consecutive times [26].

As stated in the textbook by Wayne Winston, "Given that the state at time t is i in the chain, then the process must be somewhere at time $t+1$ [27]." This means that for each i,

$$\sum_{j=1}^{j=s} P(\mathbf{X}_{t+1} = j | P(\mathbf{X}_t = i)) = 1,$$

which implies

$$\sum_{j=1}^{j=s} p_{ij} = 1 [27].$$

Thus, the sum of any row in the matrix P must be equal to one.

Previous Research

Iannis Xenakis, a French composer, architect, and mathematician is the individual responsible for originating musique stochastique, better known as stochastic music, in 1954. His first attempt at experimenting with stochastic music processes was with the composition of his piece "*Métastasis*." The performers of the piece were directed by special notation to produce sounds that a computer programmed by Xenakis specified. In 1958, Xenakis began working with an IBM 7090 computer and Markov Chains to create stochastic music [14]. Along with Markov Chains, he also worked with probability, Gaussian distributions, game theory, and Boolean algebra to create stochastic music processes [1].

In 1956, chemist Lejaren A. Hiller Jr., along with assistance from Leonard M. Isaacson, became the first person to compose an original piece using just a computer alone [18]. Using the ILLIAC, the first supercomputer housed by an academic institution, Hiller relied on the probability distribution of Markov Chains to demonstrate how computers could compose and generate music. Hiller had three previous failed attempts before his work, "Illiac Suite: String Quartet No. 4", became the first fully computer-generated musical composition, coming in around 4 minutes and 45 seconds [13]. An article published by the United Press the day following the unveiling of the song referred to it as "a suite composed by an electronic brain."

The opinions on the piece varied, and one listener even argued that music generation like this takes away the need for human composers [13].

In 1998, David M. Franz explored the use of Markov Chains for jazz improvisation, aiming to provide insights into improvisers' style and creativity [12]. Markov Chains of orders one to three were created and analyzed using Visual Basic programming. Statistical tools, such as subtraction matrices, were developed to extract information from the chains, leading to the generation of new jazz renditions based on the given jazz composition. The research postulates quantitative measures of creativity and style, concluding that Markov Chains offer valuable insights for analyzing jazz improvisation [12].

Evaluating the work of Hiller, three professors, Örjan Sandred, Mikael Laurson, and Mika Kuuskankare, revisited the musical piece "Illiac Suite: String Quartet No. 4" in 2009. Critiquing the work, the three proposed to implement the probability distribution as a stochastic rule in a constraint-based system to generate the harmony of the song as well. Using the Patch Work Musical Constraints system (PWMC), the team created two different approaches, stochastic and deterministic, to design the probability rules. The stochastic rule checked the notes that were being generated to approve if they were an appropriate depiction of the probability table. A constraint system was then created so that it would generate the sequence by suggesting notes randomly, going through the domain of possible candidates. The probability rule subsequently screened a sequence that adhered to the probability distribution. The final result showed that it was possible, without disrupting the original probability tables, to generate the harmony on top of the original musical piece [18].

A research study by Adhika Sigit Ramanto and Nur Ulfa Maulidevi, conducted in 2017, utilized Markov Chains to build a model that generated music based on user input about their mood. The purpose was to create a model that would produce and generate music based on the user's moods and feelings. The Tellegen-Watson-Clark Circumplex emotion wheel that categorizes different moods and dispositions was used to identify which route the

model should take for generating the new song. Using Java, each new musical piece was composed in C major before being transposed into a random key. After the user input their mood, the emotion wheel chose which mood to aim for and the model used a pre-specified tempo and pitch for that specified mood. Markov Chains were then used to generate the notes and their values, chords, and octave movement. The outcome of the research showed that 83.5% of respondents thought the music generated from the model fit their chosen mood and 86.5% thought the music was adequately composed [23].

Ilana Shapiro and Mark Huber's work in 2021, "Markov Chains for Computer Music Generation," is similar to the approach taken for this project [15]. The two used Markov Chains to generate music, giving an initial piece of music as the training data. Each node of the Markov Chain represented sound objects, all notes and rests, and was then translated into a transition matrix. An initial probability vector was then created to find the probability of each sound object occurring. The training data was parsed for every sound object using the 'ElementTree' and 'NumPy' libraries in Python. Once this was accomplished, they were able to generate new music. An initial probability vector was created and a standard uniform random variable number from 0 to 1 was generated. This number was then viewed as a point on the probability vector that was the result of inverse transform sampling being applied to the vector. The next highest sound object was then compared to the randomly generated point, allowing the option to choose the initial state of the Markov Model. The generated music was created using this method, but with the transition dictionary instead of the probability vector, with the length of the composition predetermined by the user. The new song was then written into a MIDI file that was then loaded into MuseScore, a musical software, to view and play. From MuseScore, they were able to compare both compositions of music to see which sound objects were different [15].

Although not published and credited, many others have tried their hand at utilizing machine learning to generate music using a Markov Chain. With the goal in mind of music

improvisation using Markov Chains, Erlijn J. Linskens investigated this possibility using Matlab in 2014 [21]. Linskens used the pitch and duration of notes from a specified given song as the states for her Markov Chain and created improvised music [21]. Research completed by Thayabaranf Kathiresan in 2015 discusses various approaches to composing music using state space models. With the use of Hidden Markov Models (HMM), Categorical Distributions (CD), and Neural Networks (NN), Kathiresan was able to successfully generate both melody and rhythm based on a given MIDI file [17].

In 2016, Alvin Lin, a member of an online machine-learning forum, created a Markov Chain in Python to generate music. For the project, he first uploaded a MIDI file of the song "River Flows in You" by Yiruma to Python and used the notes as the states for the Markov Chain. He then created a matrix that contained all the notes and generated a new song, comparing the two songs [20]. A similar project was completed by Alexander Osipenko in 2019. He, too, chose to work in Python using Markov Chains to generate music. For his initial song, he used a CSV file that contained a sequence of chords. Instead of using one chord as a state, he used bigrams. Thus, the first and second chords made the first state, the second and third chords made the second state, etc. The probability distribution of the chords was then found, the states were defined, and a random choice of which chord to start with was made. Once all of this was completed, a new song was generated [25].

A different approach of using Markov Chains to find the similarity between songs was conducted by Lance Fernando in 2018. This project used Flask+Angular.js and MySQL to create the application. While a new song was not generated based on the other given songs, the similarity between 1000 songs was found by using a Markov Chain. The transition matrix was then used to calculate the Euclidean distance between the songs to measure the similarity [11]. While this project does not deal with generation of a new song, it does provide insight into how to compare songs with a transition matrix. Another research project completed in

2018 comes from Alex Bainter, who created a model in Java using Markov Chains that generated more of his favorite composition [7]. For his model, he used a MIDI file of the song that was then converted into a JSON file. The notes were quantized and then separated, preparing them for the Markov Chain. Using the JAVA library 'markov-chains,' the newly defined notes were parsed by the model. The model then generated new phrases of his favorite track using the '.walk()' function of the 'markov-chains' library, each giving a unique sound [7].

Some other previous research includes models that use Hidden Markov Models (HMM) and time series models, such as that completed by Anna K. Yanchenko and Sayan Mukherjee. They used both of these mathematical models to compose harmonies based on given MIDI files [24]. Other similar work comes from Sam Agnew, which involves using Google's Python library, 'Magenta', for TensorFlow to work on generating music using NN's [6]. Agnew used a recurrent neural network, RNN, file within TensorFlow that was pre-trained on thousands of MIDI files' melodies. For the input data, the model primed a given input MIDI file. The model used basic RNN, Lookback RNN, and Attention RNN, each providing more structured compositions [6]. While not dealing with music, Bhoomika Madhukar was able to create a system that generated text based on given input text [22]. The ideas from Madhukars' work can help build a function that will generate new music based on a given song.

CHAPTER 2

MATHEMATICS FOR STOCHASTIC MARKOV CHAINS

Before delving into the research question at hand, it will be helpful to examine a few examples of Markov Chains to better understand the concept.

Simple Markov Chain Examples

Example 1: We base this simple example on work done in the Sekhon textbook [26].

Spotify Wrapped showed that the top two musical artists of last year were Taylor Swift and Olivia Rodrigo.

- The data shows that during the year 13% of Swifties switched to a Livie; the remaining 87% of Swifties stayed loyal to Taylor.
- On the other hand, 24% of Livies switched to a Swiftie.

Express this information as a transition matrix that displays the probabilities of going from one state to another state.

To create the transition matrix, we must know all the possible states. There are two states in this example, Swiftie and Livie. These two states will make up both the rows and columns for the matrix. The rows of the matrix will represent the state we are currently in, and the columns of the matrix will represent which state we want to move to, with the sum of each row equal to one.

To finish constructing the transition matrix, we use the information given. We know that 13% of Swifties switched to a Livie, thus, we start with the row that says Swiftie and move to the column that says Livie.

$$\begin{array}{c} \\ \text{Swiftie} \\ \text{Livie} \end{array} \begin{array}{c} \begin{array}{cc} \text{Swiftie} & \text{Livie} \end{array} \\ \begin{pmatrix} & .13 \\ & \end{pmatrix} \end{array}$$

We also know that the remaining 87% of Swifties stayed loyal to Taylor. Hence, we are staying in the same row, Swiftie, and are now moving to the Swiftie column.

$$\begin{array}{c} \\ \text{Swiftie} \\ \text{Livie} \end{array} \begin{array}{c} \begin{array}{cc} \text{Swiftie} & \text{Livie} \end{array} \\ \begin{pmatrix} .87 & .13 \\ & \end{pmatrix} \end{array}$$

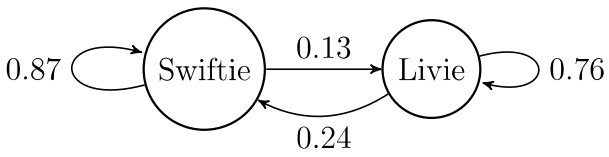
For Olivia's fans, we are given that 24% of Livies switched to a Swiftie. Now we are starting with the row that says Livie and moving to the column that says Swiftie.

$$\begin{array}{c} \\ \text{Swiftie} \\ \text{Livie} \end{array} \begin{array}{c} \begin{array}{cc} \text{Swiftie} & \text{Livie} \end{array} \\ \begin{pmatrix} .87 & .13 \\ .24 & \end{pmatrix} \end{array}$$

To find the missing probability for the Livie Livie spot in the matrix, we use the rule that the sum of every row must be equal to 1. Hence, $1 - 0.24 = 0.76$. Thus, 76% of Livies stayed loyal to Olivia.

$$\begin{array}{c} \\ \text{Swiftie} \\ \text{Livie} \end{array} \begin{array}{c} \begin{array}{cc} \text{Swiftie} & \text{Livie} \end{array} \\ \begin{pmatrix} .87 & .13 \\ .24 & .76 \end{pmatrix} \end{array}$$

Although not asked, we can easily represent the Markov Chain for this problem using a directed graph. To create this, we will use the probabilities from the transition matrix we just fabricated.



To read this Markov Chain, we follow the arrows that are associated with each state, or circle. Therefore, there is an 87% chance that Taylor's fans will stay loyal to her and a 13% chance that Swifties will switch to a Livie during the year. For Olivia, there is a 24% chance that Livies will switch to a Swiftie sometime during the year and a 76% chance that Olivia's fans will stay loyal to her.

Example 2: This example is a little more challenging, dealing with two-state transitions. This example comes from the Winston textbook [27].

"Assume that tomorrow's weather depends on the last two days of Smalltown's weather in the following manner:

- If the last two days have been sunny, then 95% of the time, tomorrow will be sunny.
- If yesterday was cloudy and today is sunny, then 70% of the time, tomorrow will be sunny.
- If yesterday was sunny and today is cloudy, then 60% of the time, tomorrow will be cloudy.
- If the last two days have been cloudy, then 80% of the time, tomorrow will be cloudy.

Using this information, model Smalltown's weather as a Markov Chain [27]."

The initial step is to create a transition matrix for all the different possible outcomes. There can be a sunny and then sunny day, a sunny then a cloudy day, a cloudy then a sunny day, and a cloudy and then cloudy day. Just like in *Example 1*, these states will be the rows and columns for the matrix, where the rows of the matrix will be the state we are currently in, the columns of the matrix will be which state we want to move to, with the sum of each row equal to one.

To start, we fill in all of the probabilities that were given:

$$P = \begin{array}{c} \\ SS \\ SC \\ CS \\ CC \end{array} \begin{array}{c} \begin{array}{cccc} SS & SC & CS & CC \end{array} \\ \begin{pmatrix} .95 & & & \\ & & & .60 \\ .70 & & & \\ & & & .80 \end{pmatrix} \end{array}.$$

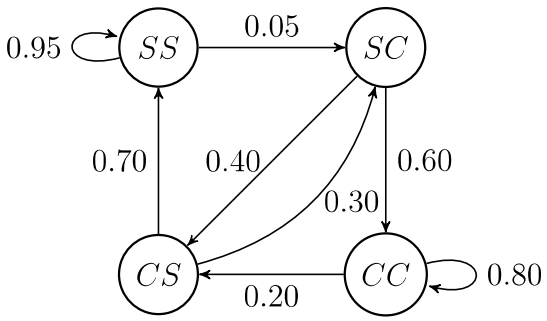
If the last two days were sunny, it would not be possible to go from sunny sunny to cloudy sunny, since the day prior was sunny and not cloudy. Using this logic, we can fill in these spots in the matrix with a zero probability.

$$P = \begin{array}{c} \\ SS \\ SC \\ CS \\ CC \end{array} \begin{array}{c} \begin{array}{cccc} SS & SC & CS & CC \end{array} \\ \begin{pmatrix} .95 & & .00 & .00 \\ .00 & .00 & & .60 \\ .70 & & .00 & .00 \\ .00 & .00 & & .80 \end{pmatrix} \end{array}$$

The final step to finishing the transition matrix is to use the rule that each row must sum to one. Thus, take the probabilities that are in each row and subtract them from one. This would then give the completed probability matrix:

$$P = \begin{array}{c} \\ SS \\ SC \\ CS \\ CC \end{array} \begin{array}{c} \begin{array}{cccc} SS & SC & CS & CC \end{array} \\ \begin{pmatrix} .95 & .05 & .00 & .00 \\ .00 & .00 & .40 & .60 \\ .70 & .30 & .00 & .00 \\ .00 & .00 & .20 & .80 \end{pmatrix} \end{array}.$$

Now that the transition matrix is complete, a Markov Chain can then be created for this scenario, represented as a directed graph.



Adapting Concept to Composition

How would using Markov Chains for generating music work? The idea is to create a new song based on an original song using a Markov Chain. The first step of using a Markov Chain to generate a new song would be finding an original song to work with. This original song would be our training data. Once the song is chosen, the states of the song need to be defined. The states would be all the rests, notes, and their respective lengths and octaves used to compose the original song. A Markov Chain is then created using all the states of the original song. This Markov Chain will then generate new rests, notes, and their respective lengths and octaves based on the training data. This would then create a newly generated song based on the original seeding piece.

This can be accomplished in Python, using random seeds as the work environment. A state transition dictionary has to be created, containing all the unique rests and notes from the original song. This state transition dictionary will serve as the stochastic transition matrix, P. A sequence of states is then created using a for loop that generates new rests and notes, all based on the training data. This output creates a newly generated song based on the original song, which can then be played. Since the seed for the generation is random, running the Python code again will give a different-sounding newly generated song.

Experimenting With "Twinkle Twinkle Little Star"

Before working with the more elaborate songs for this project, the ideas and methods were initially tested on the song "Twinkle Twinkle Little Star." This particular song was chosen because it is well-known, short, and easy to work with. The song was coded into Python, using the 'Musicpy' library within Python. There are several different ways to code music utilizing this library.

To begin, sheet music was found for the song "Twinkle Twinkle Little Star" to be the training data. After it was obtained, it was then translated into the 'Musicpy' format so it could be manipulated in Python. The following is the syntax that was used for encoding this song.

$$\textit{SONG NAME} = \text{chord}(\textit{'NOTE NAME'},\ \text{interval} = [\textit{NOTE LENGTH}]),$$

where *SONG NAME* is the name of the song, *NOTE NAME* is the name of the note, and *NOTE LENGTH* is the length of the note. This notation is a part of the 'Musicpy' library and is how this package reads a song. For example, *NOTE NAME* = C4 and *NOTE LENGTH* = 1/4 would represent a C4 quarter note. The entire song was formatted in this manner, giving:

TTLS = chord('C4, C4, G4, G4, A4, A4, G4, F4, F4, E4, E4, D4, D4, C4, G4, G4, F4, F4, E4, E4, D4, G4, G4, F4, F4, E4, E4, D4, C4, C4, G4, G4, A4, A4, G4, F4, F4, E4, E4, D4, D4, C4', interval=[1/4, 1/4, 1/4, 1/4, 1/4, 1/4, 1/2, 1/4, 1/4, 1/4, 1/4, 1/4, 1/4, 1/2, 1/4, 1/4, 1/4, 1/4, 1/4, 1/4, 1/2, 1/4, 1/4, 1/4, 1/4, 1/4, 1/4, 1/2, 1/4, 1/4, 1/4, 1/4, 1/4, 1/4, 1/2, 1/4, 1/4, 1/4, 1/4, 1/4, 1/4, 1/2, 1/4, 1/4, 1/4, 1/4, 1/4, 1/4, 1/2]).

To play the song, the following command is used:

$$\text{play}(\textit{SONG NAME},\ \text{bpm} = \textit{NUMBER}),$$

where *SONG NAME* is the same name you gave the previously mentioned defined song and

bpm = *NUMBER* is the beats per minute for the song. Thus, for this example, the command would be

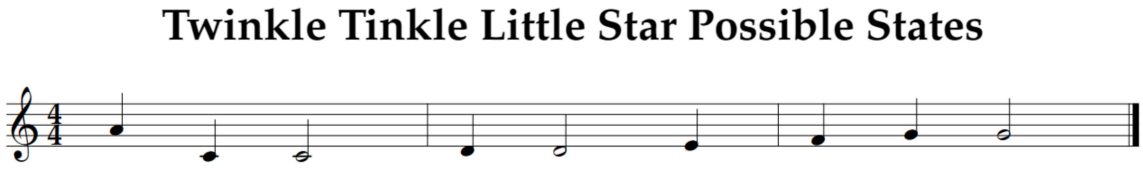
$$\text{play(TTLS, bpm} = 117).$$

The next step is to create a transition dictionary based on all the possible states. The states of this song are all the notes and their respective length and octaves used in "Twinkle Twinkle Little Star." Thus, the states would include:

$$\{\text{A4 1/4, C4 1/4, C4 1/2, D4 1/4, D4 1/2, E4 1/4, F4 1/4, G4 1/4, G4 1/2}\},$$

for a total of 9 states. Musically these are shown in Figure 2.1.

Figure 2.1. *All possible states in "Twinkle Twinkle Little Star" represented musically.*



The first step to creating a transition dictionary is to find what note each note goes to next. Hence if we are at step i, what is state j based on state i? The first pass here was done in Excel, with the setup being a 9×9 matrix where the possible notes are the rows and columns. To initiate the process, we start with the first note occurring in the song, i, and map it to the note that directly follows, j. Then, from this new note j, map it to the note $j + 1$ that follows it. This process is continued until the last state, or note, is reached. Employing this process for the whole song, the final product for the outcomes looks like the following:

$$
\mathrm{O} = \begin{array}{c} \\ \text{C4 0.25} \\ \text{C4 0.5} \\ \text{D4 0.25} \\ \text{D4 0.5} \\ \text{E4 0.25} \\ \text{F4 0.25} \\ \text{G4 0.25} \\ \text{G4 0.5} \\ \text{A4 0.25} \end{array}
\begin{array}{c}
\begin{array}{ccccccccc} \text{C4 0.25} & \text{C4 0.5} & \text{D4 0.25} & \text{45 0.5} & \text{E4 0.25} & \text{F4 0.25} & \text{G4 0.25} & \text{G4 0.5} & \text{A4 0.25} \end{array} \\
\begin{pmatrix}
2 & & & & & & 2 & & \\
 & & & & & & 1 & & \\
 & 2 & 2 & & & & & & \\
1 & & & & & & 1 & & \\
 & & 2 & 2 & 4 & & & & \\
 & & & & 4 & 4 & & & \\
 & & & & & 2 & 4 & & 2 \\
 & & & & & 2 & & & \\
 & & & & & & & 2 & 2
\end{pmatrix}
\end{array}.
$$

The total amount of transitions is found by taking the sum of each row in the matrix. Therefore, the total amount of transitions in "Twinkle Twinkle Little Star" is 41. Subsequently, to find the probability transition matrix, all the elements of each row in the matrix are divided by the sum of that given row, which will give the probability of each outcome occurring. Thus, for "Twinkle Twinkle Little Star," this results in the following probability of each outcome occurring:

$$
\mathrm{P} = \begin{array}{c} \\ \text{C4 0.35} \\ \text{C4 0.5} \\ \text{D4 0.25} \\ \text{D4 0.5} \\ \text{E4 0.25} \\ \text{F4 0.25} \\ \text{G4 0.25} \\ \text{G4 0.5} \\ \text{A4 0.25} \end{array}
\begin{array}{c}
\begin{array}{ccccccccc} \text{C4 0.25} & \text{C4 0.5} & \text{D4 0.25} & \text{D4 0.5} & \text{E4 0.25} & \text{F4 0.25} & \text{G4 0.25} & \text{G4 0.5} & \text{A4 0.25} \end{array} \\
\begin{pmatrix}
0.50 & & & & & & 0.50 & & \\
 & & & & & & 1 & & \\
 & 0.50 & 0.50 & & & & & & \\
0.50 & & & & & & 0.50 & & \\
 & & 0.25 & 0.25 & 0.50 & & & & \\
 & & & & 0.50 & 0.50 & & & \\
 & & & & & 0.25 & 0.50 & & 0.25 \\
 & & & & & 1 & & & \\
 & & & & & & & 0.5 & 0.5
\end{pmatrix}
\end{array}
$$

It is not necessary to find the probability transition matrix, however, since this was the first time trying to generate a new song based on a given song this action was completed. It is helpful to look at the probability of each note occurring based on a given note. The whole process of creating both matrices is the mathematical approach, however we need a new method for coding.

After the transitions in "Twinkle Twinkle Little Star" are found, we can then create a transition dictionary in Python. This dictionary will use a list as the data associated with the dictionary. To code this in Python, the following syntax is used:

$$\textit{DICTIONARY NAME} = \{\textit{'KEY'} : [\textit{'VALUE 1'}, \textit{'VALUE 2'}]\},$$

where we are giving a list with multiple values as the data associated with the key. *DICTIONARY NAME* is what you want to name the dictionary, *KEY* is a particular variable that is going to be mapped, and *VALUE 1* and *VALUE 2* are the specific variables the key is mapped to. Therefore, using the transitions found earlier, we can, by hand, create the dictionary for "Twinkle Twinkle Little Star," giving:

$$\begin{aligned}
\text{TTLSTransitionDictionary} = \{ \quad & \text{“}C41/4\text{”} : [\text{“}C41/4\text{”}, \text{”}G41/4\text{”}], \\
& \text{“}C41/2\text{”} : [\text{“}G41/4\text{”}], \\
& \text{“}D41/4\text{”} : [\text{“}C41/2\text{”}, \text{“}D41/4\text{”}], \\
& \text{“}D41/2\text{”} : [\text{“}C41/4\text{”}, \text{“}G41/4\text{”}], \\
& \text{“}E41/4\text{”} : [\text{“}D41/4\text{”}, \text{“}D41/2\text{”}, \text{“}E41/4\text{”}, \text{“}E41/4\text{”}], \\
& \text{“}F41/4\text{”} : [\text{“}E41/4\text{”}, \text{“}F41/4\text{”}], \\
& \text{“}G41/4\text{”} : [\text{“}F41/4\text{”}, \text{“}G41/4\text{”}, \text{“}G41/4\text{”}, \text{“}A41/4\text{”}], \\
& \text{“}G41/2\text{”} : [\text{“}F41/4\text{”}], \\
& \text{“}A41/4\text{”} : [\text{“}G41/2\text{”}, \text{“}A41/4\text{”}] \quad \}.
\end{aligned}$$

This dictionary shows all the possible states of the song, and which new state each specific state can go to. Thus, from state i, it can be determined which state j state i may transition to next.

The next step in the process of trying to generate a new song based on the training data is to create a for loop that will generate random notes and the length of each random note from the dictionary we created. To start, we need to construct an empty list that will later contain all the randomly selected notes and their respective length. Once the empty list has been created, we can choose a random state to start from, i.e. any of the notes that are defined in the dictionary. Once this initial state is chosen, we then append this note to the empty list so that it now contains only this state. Now we can create a for loop that will generate the random states. A for loop is used when a predetermined amount of repetitions is known. This for loop will iterate over the list that contains the newly added random states. During the loop, the value of the state that we are currently in will be updated by selecting a random state from the dictionary, which will now become the new state we are in. Then, each new value of the state we are in will be appended to the generated song.

For this example, an empty list was created, and the initial state was chosen to be E4 1/4, for no specific reasoning. After the initial state E4 1/4 was established, this first state was appended to the list. A for loop was then used to generate the randomly chosen states that would be appended to the list. The loop iterates over the list and the value of the state we are currently in is updated by selecting a random state from the dictionary 'TTLSTransitionDictionary.' This new randomly selected state will now become the new state we occupy. Each new value of the state we are in will be appended to the list.

Once we have all the randomized states in the list, we now need to convert this list into a way that can be parsed through to pick out all the notes and their corresponding length. First, we need to create an empty string that will house all the notes from the current list and an empty list that will house all the corresponding notes' lengths. Next, we need to

use a for loop that will parse through each of the states in the current list and split it into a note and a duration. The string we created now includes this note, and the list we just created now includes this note length. For example, since the first key in the previously constructed list is E4 1/4, this would now become E4, which will be added to the new note string variable, and 1/4, which will be added to the new duration list variable.

After the new song has been split up into a string that contains just the note names and a list that contains only the length of the notes, it is now attainable to listen to the new song. This is done the same way as before with the original song, where our note string is now our notes and our duration list is now the intervals.

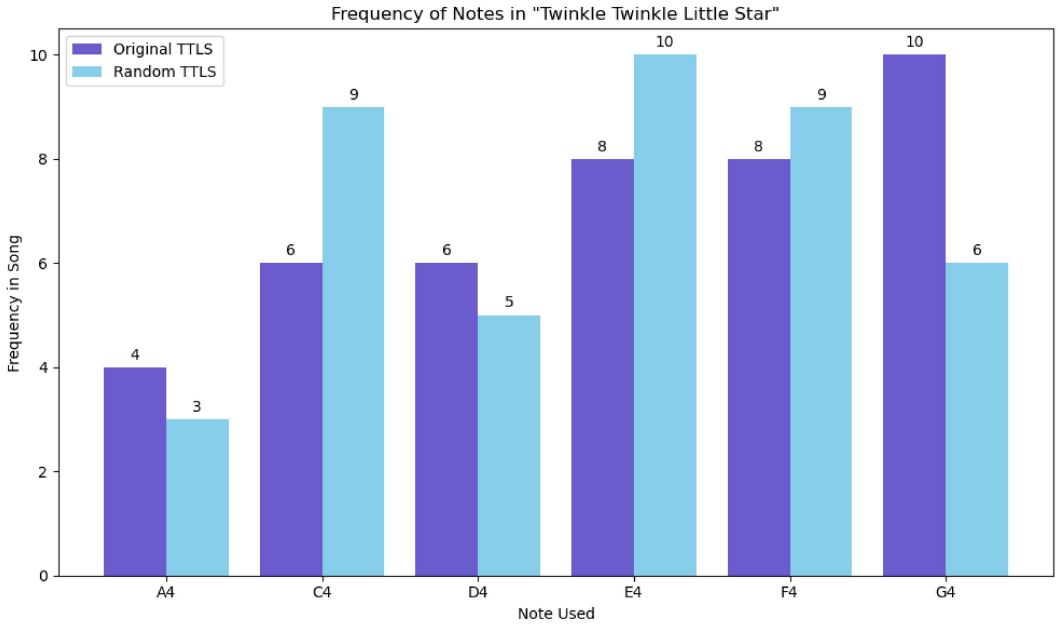
$$TTLSR = \text{chord}(noteString,\ \text{interval} = durationList)$$

To generate a new song from "Twinkle Twinkle Little Star," the number of random states was chosen to be 41 since there are a total of 42 states in the original song and we need to take into account the initial state we are in. Thus, the state list will now contain a total of 42 states, the initial state and the 41 random states generated from the for loop. Hence, this newly generated song will have the same amount of states as the original song, even though it may not be the same length of time.

The generated song and the original song are similar since the generated song uses the same notes and note lengths as the original song. What if we wanted to see how related they are? One way to see this is to compare the new song to the training data by creating different graphs. To examine how similar the songs are, we can create multiple bar plots. We can create one that plots the frequency of each note, the frequency of each specific note type, and another bar plot that plots the frequency of the length of the notes. Since there are no rests in the original song, we do not have to worry about excluding any lengths. All the analysis of this generated song comes from a specific seed that Python randomly chose to use for just one run-through of the code. If the code were to be run again, the analysis would look slightly different, since it would be operating on a different seed.

First, we can take a look at the frequency of each note used in both the original and the generated songs. Figure 2.2 shows the distribution of the notes within the original and newly generated songs. Comparing these two songs, we can see a difference in the amount of each note used in both songs. For example, the generated song uses more C4, E4, and F4 notes, and fewer A4, D4, and G4 notes. It is possible to have that both the original and new song have the same amount of notes, however, this did not occur in this specific seed.

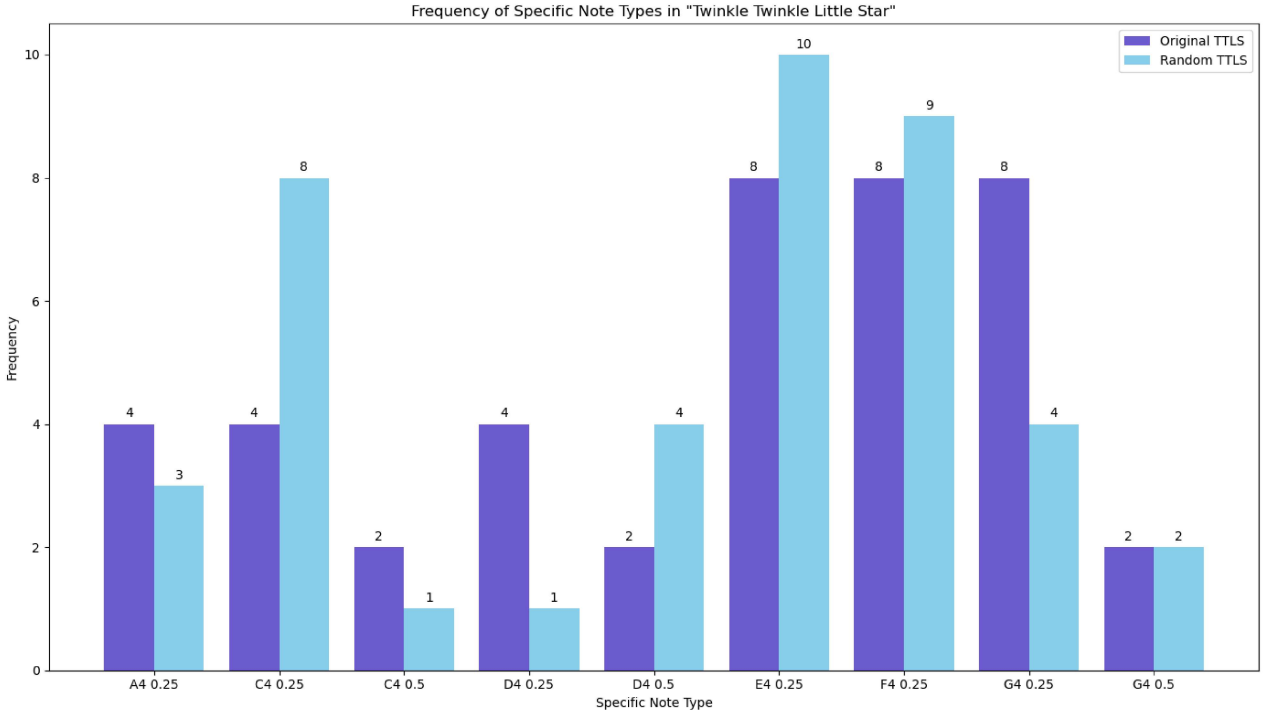
Figure 2.2. *Frequency of notes in both the original and newly generated "Twinkle Twinkle Little Star."*



Along with looking at the amount of each note used, we can also look at the frequency of each specific note type in both the original and new songs. Figure 2.3 displays the distribution of the specific note types in the original and newly generated song. We can again compare the two songs to find the difference in the amount of each specific note type used in each song. We can see that G4 0.5 occurs the same number of times in both songs. Here

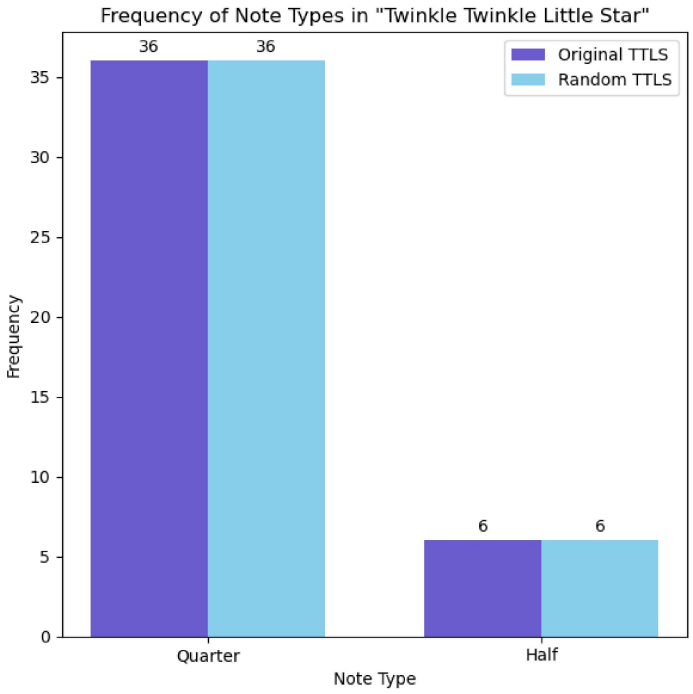
we can also notice the specific note types that differ the most between the two songs are C4 0.25, D4 0.25, and G4 0.25, and the other note values differ by one or two times for both songs.

Figure 2.3. *Frequency of specific note types in both the original and newly generated "Twinkle Twinkle Little Star."*



Finally, we can take a look at the total number of quarter and half notes used in both songs. Figure 2.4 exhibits the length of notes used in both versions of the song. Both the original and the random seed-generated song use 36 quarter notes and 6 half notes. Hence, each song will have the same length. This is not necessarily always true, it just happened to work out in this specific seed that was randomly chosen by Python.

Figure 2.4. *Frequency of note lengths in both the original and newly generated "Twinkle Twinkle Little Star."*



Overall, this newly generated rendition of "Twinkle Twinkle Little Star" is very similar in this frequency composition. It has the feel and essence of the original song, yet is clearly different when listened to. Since we are using the probabilities of each state from the original song to create the new song, this makes sense and is what we would expect. There are still distinguishable differences between the two, however, it is easy to tell that this new song is based on the original "Twinkle Twinkle Little Star."

CHAPTER 3

INTRICATE MELODIES AND COMPLEX ARRANGEMENTS

Now that we know it is achievable and Python is capable of using a Markov Chain and transition dictionary to generate a new song based on a given song, it is time to look broader. We now seek to delve into more complex compositions. Although it is possible to work with many parts of a song, such as harmony, chords, or bass line, the main focus will be on the melody. It should be noted that the song choices for this project were chosen out of personal preference.

"You Belong With Me" by Taylor Swift

The first intricate song selected to work with was "You Belong With Me" by Taylor Swift, from her 2008 album, "Fearless." The initial step was to obtain free sheet music, however, none of the free online sheet music was in the key of the original song rendition of F#. A transposition was then composed using MuseScore, and all ties were removed, written as if they were regular notes, to allow 'Musicpy' the ability to parse and play the song. This song was used as the training data, which will be used to estimate the probabilities for the Markov Chain and transition matrix.

Once this was accomplished, the sheet music could be translated into ASCII format to be manipulated using the library 'Musicpy'. The approach to creating the state transition dictionary for this song in Python differs from what was used for "Twinkle Twinkle Little Star." This new format requires that all possible states are first defined, then a list of possible state transitions can be found. The following format for notes and rests were used respectively;

$$\text{FSh31o8} = \text{chord}(\text{`F\#3'},\ \text{interval} = [1/8])$$
$$\text{rest4o4} = \text{rest}(4/4).$$

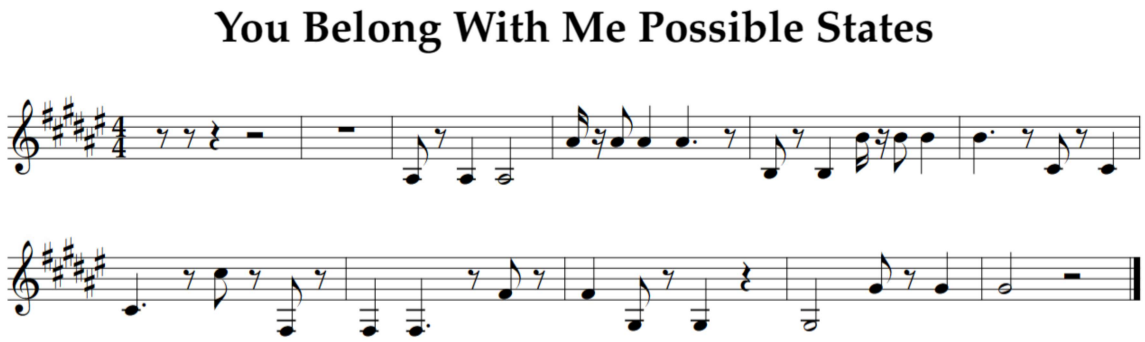
For example, the above formats represent an F#3 eight note and a whole rest. This notation was used for all the rest and note types in the song, which each represent a different

transition state. There are a total of 4 different rests and 30 different note types defined, each representing all of the possible states in the training data. The following are all the 34 possible states:

{rest 0.125, rest 0.25, rest 0.5, rest 1.0, A#3 0.125, A#3 0.25, A#3 0.5,
A#4 0.0625, A#4 0.125, A#4 0.25, A#4 0.375, B3 0.125, B3 0.25,
B4 0.0625, B4 0.125, B4 0.25, B4 0.375, C#4 0.125, C#4 0.25, C#4 0.375,
C#5 0.125, D#4 0.125, D#4 0.25, F#3 0.125, F#3 0.25, F#3 0.375,
F#4 0.125, F#4 0.25, G#3 0.125, G#3 0.25, G#3 0.5,
G#4 0.125, G#4 0.25, G#4 0.5, G#4 0.125, G#4 0.25, G#4 0.5}.

Musically, these states are represented in Figure 3.1.

Figure 3.1. *All possible states in "You Belong With Me" represented musically.*



After all the possible states were defined, it was now time to code the song. The format used for this song differs from what was used for "Twinkle Twinkle Little Star." For this new format, each state gets added to the following state, which gets concatenated to the next following state, etc. Hence, the song is written with a '+' between each state. For example, a snippet of the code for the song is:

$$\text{YBWM} = \text{CSh41o4} + \text{ASh31o8} + \text{ASh31o8} + \text{GSh31o4} + \text{FSh31o8} + \text{FSh31o8} + ...$$

To play the song, the same syntax format is used as before,

$$\text{play}(\text{YBWM},\ \text{bpm} = 130,\ \text{instrument} = 25),$$

where we are playing the song just made, titled 'YBWM', the beats per minute is 130, and the instrument we are playing it on is a nylon guitar.

The next step in generating a new song with a Markov Chain based on "You Belong With Me" was to create a transition dictionary. To create this, the state transition matrix first had to be established. This matrix was constructed manually in Excel, although, a less hands-on process would work too. A method could be developed so that the computer would do the work, making it easier and less manual labor intensive. The training data was analyzed to find all the possible notes and rests, along with their respective length and octaves, which were then added as a state, or as both a row and column, to the matrix. The song was parsed through to find the total number of occurrences for each state, completing the matrix. Once the state transition matrix was created, the transition dictionary was then fabricated by hand and coded into Python. This dictionary, 'YBWMD,' includes every possible state in the original song as the keys of the dictionary. The dictionary's values are all the other states a given state transitions to in the original song. This dictionary is used for the probabilities of the Markov Chain.

Once the dictionary was produced, a list that contains all the possible states was constructed, excluding all the rest states since the 'Musicpy' library cannot handle songs that start with a rest. The purpose of this list is for Python to select a state that occurs in the training data randomly. This random state is then manually set as the initial state for building the newly generated song. An empty variable is also created to house the new song as it is being actively generated.

Similar to how the newly generated "Twinkle Twinkle Little Star" song was coded, a for loop is also implemented for this generated song. This loop runs for a pre-specified amount of iterations, where inside the loop a randomly selected value associated with the

key that was randomly chosen from the list that contains all the possible states is picked from 'YBWMD.' For instance, if the randomly selected key from is B41o8, then B41o8 is located within 'YBWMD.' In 'YBWMD,' this key and its corresponding values are:

B41o8 : [ASh41o8, B41o8, B41o8, B41o4].

Once B41o8 has been matched with the correct key, then randomly either ASh41o8, B41o8, B41o8, or B41o4 is chosen as the new state to transition to. This state is then set as the new value for the initial state and is added to the variable that houses the new song. Outside the for loop, the new song variable is now given a meaningful title, such as 'verse1,' to represent a specific part of the song. After being renamed, the new song variable is set equal to 0 again for the next run-through of the for loop. If this variable is not set equal to 0, the new random states will keep adding onto the previously added states to the new song variable.

Thus, the for loop takes the random state given by the list that contains all the possible states and randomly chooses a note from its values in 'YBWMD' to play next. This would make up the first state of the new song that is being generated. The second state is found by following the same steps; the first state is run through the for loop where it starts at its respective key from the dictionary and then randomly chooses a note from its values to play. This second state will then be added onto the song chain, meaning we will have the first state plus the second state plus the third state, etc.

For generating the new "You Belong With Me" song, the same musical structure as the original song was used. Meaning, trying to mimic the structure of the seeding song by creating different distinct architectural sections of the newly generated song. This is done so that the newly generated song has repeated parts, just as the original composition does. These sections consisted of two different verses, a pre-chorus, a chorus, two different instrumentals, the bridge, a different chorus, and an outro. For this new song, the order was:

'verse1 + prechorus + chorus + instrumental1 + verse2 + prechorus + chorus + chorus + instrumental2 + bridge + diffchorus + chorus + outro.'

Hence, a for loop was fabricated for each of the distinct sections with its respective number of states.

The number of random states was chosen to be the same amount as the total number of states in each section of the original song. Table 3.1 shows the specific number of random states for each section of the song the for loop iterates through. Unlike when generating a new "Twinkle Twinkle Little Star," we do not have to take into consideration the initial state since we are only using this to generate the actual first state in the for loop. Overall, there are a total of 618 state transitions for both the original and newly generated "You Belong With Me." Thus, a newly generated composition based on the training data using a stochastic model was obtained.

Table 3.1. *Number of state transitions in each section of the original song to be generated for the new rendition of "You Belong With Me."*

Section of Song	**Number of Random State Transitions**
verse1	81
prechorus	48
chorus	44
instumental1	4
verse2	100
instumental2	12
bridge	65
diffchorus	42
outro	42

"Don't Blame Me" by Taylor Swift

The second complex melody arrangement chosen was "Don't Blame Me" by Taylor Swift from her 2017 album, "reputation." Again, the first step was to acquire sheet music for the song. However, as the sheet music for "You Belong With Me," there needed to be some changes made. Using MuseScore, the song was re-written in the correct time signature, transposed to the original songs' key of A minor, and the ties were removed, making all of the tied notes into regular notes. This rendition of the song was then coded into Python as the training data, once again using the library 'Musicpy', in the same manner as "You Belong

With Me." For "Don't Blame Me," there are a total of 5 different rests and 42 different note types that are defined, each representing a single possible state in the training data. The 47 possible states are:

{rest 0.03125, rest 0.125, rest 0.25, rest 0.5, rest 1.0,

A3 0.08$\overline{3}$, A3 0.125, A3 0.1875, A3 0.25, A3 0.375,

A4 0.08$\overline{3}$, A4 0.125, A4 0.1$\overline{6}$, A4 0.1875, A4 0.25,

B3 0.375, C4 0.08$\overline{3}$, C4 0.125, C4 0.1$\overline{6}$, C4 0.25, C4 0.375,

C5 0.08$\overline{3}$, C5 0.125, C5 0.25, C5 0.5, D3 0.125, D3 0.25,

D4 0.08$\overline{3}$, D4 0.125, D4 0.1$\overline{6}$, D4 0.2, D4 0.375,

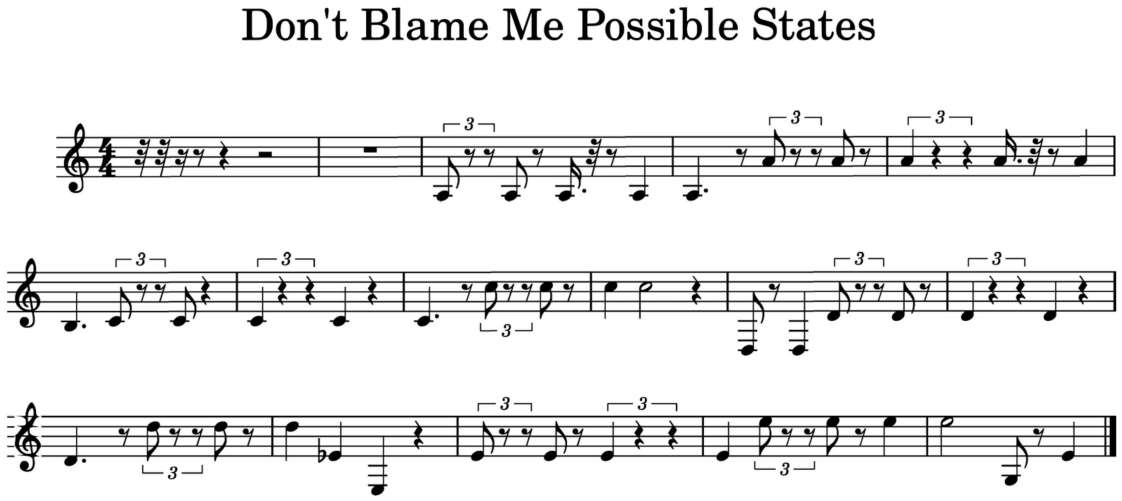
D5 0.08$\overline{3}$, D5 0.125, D5 0.25, Eb4 0.25, E3 0.25,

E4 0.08$\overline{3}$, E4 0.125, E4 0.1$\overline{6}$, E4 0.25,

E5 0.08$\overline{3}$, E5 0.125, E5 0.25, E5 0.5, G3 0.125, G4 0.25}.

Musically, these states are represented in Figure 3.2.

Figure 3.2. *All possible states in "Don't Blame Me" represented musically.*



Moving forward, the next step was to create the transition dictionary. For this dictionary, we wanted Python to create it all on its own, requiring no hand calculations. Many failed attempts were made at trying to create a working dictionary, nevertheless, a solution was developed. Before the dictionary could even be built, the first step that had to be taken was to define a function that would map between ASCII notation to the corresponding note format in 'Musicpy.' The purpose of this function is to map the names of the states that are given to the actual chord format. For example, the following is a snippet of part of the function:

```
def ASCIItoNote(note):
    if note == 'rest4o4':
        return rest(4/4)
    if note == 'D31o8':
        return chord('D3', interval = [1/8]).
```

To start the process of working on the dictionary, the song was converted into a string. The string was then split on the '+' symbol and converted into a list that contained only the states of the song. After this was completed, an empty list and an empty dictionary were initialized. The list will store the unique states in the song list, and the dictionary will store the unique states as the keys and the values will be lists of the subsequent states the specific key can transition to. Thus, the dictionary will be generated using ML in Python. Two for loops were then fabricated to generate the dictionary. The first for loop iterated through the song list and added each new state to the empty list. This state was then appended to the empty dictionary as the keys of the dictionary. The second for loop then appended the next successive state to the values in the dictionary created for each key. Hence constructing a transition matrix where each state is associated with the states that can follow it. The final result is a completed dictionary, 'DBMD,' which represents the Markov Chain transition probabilities.

After the transition dictionary was created, the next step was to generate a new song using the Markov Chain probabilities just found. In order to do this, we needed to pick a random state to start the song on. Again, we wanted Python to do all the work, thus the approach taken to generate a random state to start from differs from what was done for both "Twinkle Twinkle Little Star" and "You Belong With Me." An empty list was initialized and a for loop was built, which parses through all the keys in 'DBMD' and appends each unique key to the initialized list. Once all of the possible states were combined into one list, Python then chose a random state from the dictionary to be the initial state for the new song. This state is then run through the function 'ASCIItoNote' previously defined, mapping from ASCII notation to the note format in 'Musicpy.' Once this was completed, these states were then added to the variable that houses the newly generated piece of music.

The process of generating the new song was the same as the process used previously for "You Belong With Me." Again with the idea in mind of using the same musical structure as the original song, the same number of states per distinct section of the original song was used for the generating process. Thus, mimicking the structure of the seeding song. The sections of the original song consisted of an introduction chorus, two different verses, a pre-chorus, two different riffs, a solo, a bridge, and an outro. The order of the new song was:

'introchorus + verse1 + prechorus + chorus + verse2 + prechorus + chorus + riff + riff + solo + bridge + chorus + diffriff + outro.'

A for loop was fabricated for each of the distinct sections in the original song with its respective number of states. It can be noted here that a potential way to run this would be to have a stochastic process defined on each distinct architectural section of the original song to create a new version. However, doing it this way may sound too similar to the work done by the original artist, which is not the intention of this research.

Table 3.2 displays the total number of random states for each section of the song. Additionally, since we were only using the initial state to generate the actual first state of the for loop, we did not have to consider that this would add an additional random state

to the for loop outcome. This gives a total of 728 state transitions overall for both the original "Don't Blame Me" composition and the newly generated song. Hence, using the probabilities from the Markov Chain created based on the training data, "Don't Blame Me," a new piece of music was then generated.

Table 3.2. *Number of state transitions in each section of the original song to be generated for the new rendition of "Don't Blame Me."*

Section of Song	**Number of Random State Transitions**
introchorus	47
verse1	66
prechorus	40
chorus	96
verse2	80
riff	28
solo	19
bridge	38
diffriff	30
outro	24

CHAPTER 4

STATISTICAL ANALYSIS

After both songs were generated, an analysis of each new song compared to its respective original base song was assembled. All of the analyses of the songs were done using Python. The first step was to print each song, which gives the notes and length of the notes. The second step was printing the song notes and song note lengths separately. Bar graphs were created to show the note frequencies, specific note type frequencies, and the length of note frequencies. This was done for both the original songs and the newly generated random songs.

It should be noted here that even though rests are included as a possible state, they are not separated as a possible state in the 'Musicpy' library in Python. They are capable of being played but will not show up as their own specific state in the generated music piece. This means that for the newly generated songs, some notes are combined with a rest or multiple rests, leading to the frequency count and the total number of specific states for a certain note to not add up to the same sum in some cases. Although this may seem an issue, it was not changed since the main goal is to look at the melody notes and not the rest time.

Bar Graphs

"You Belong With Me"

Figure 4.1 gives the bar graph for the total amount of each note found in the original composition of "You Belong With Me" and its newly generated song. The original composition is in yellow, while the newly generated song is in orange. The bar graph for the total number of each specific note type used in the original song and the generated song can be seen in Figure 4.2. Figure 4.3 displays the bar graph for the total number of each note length in both the original and generated songs.

Figure 4.1. *Frequency of notes in both the original and newly generated "You Belong With Me."*

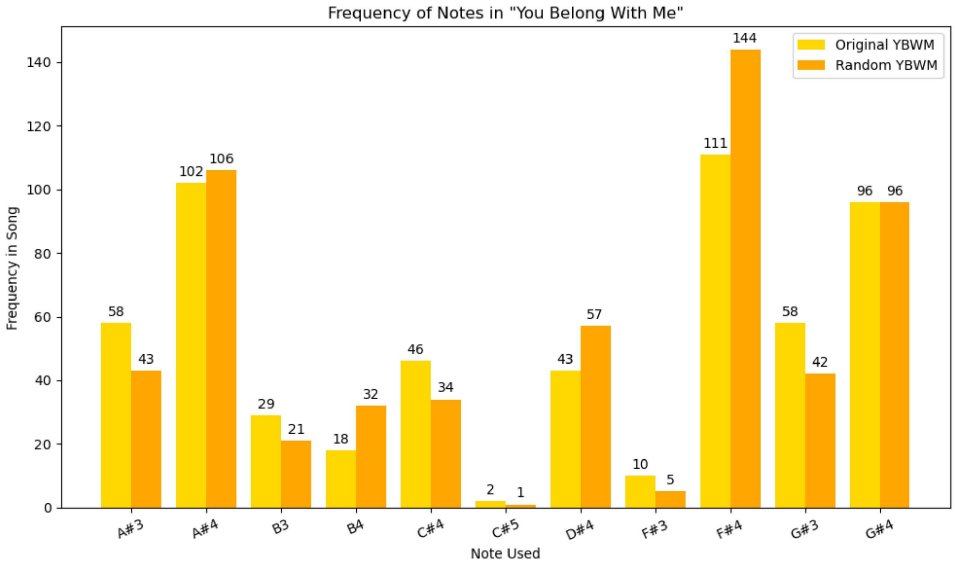


Figure 4.2. *Frequency of specific note types in both the original and newly generated "You Belong With Me."*

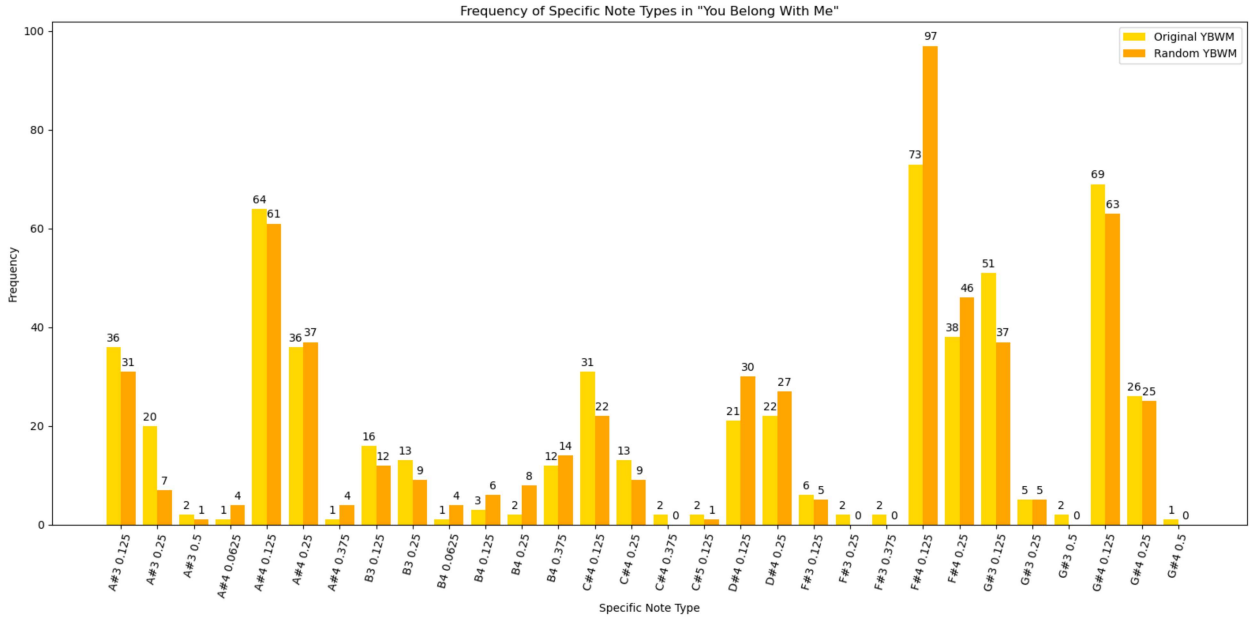
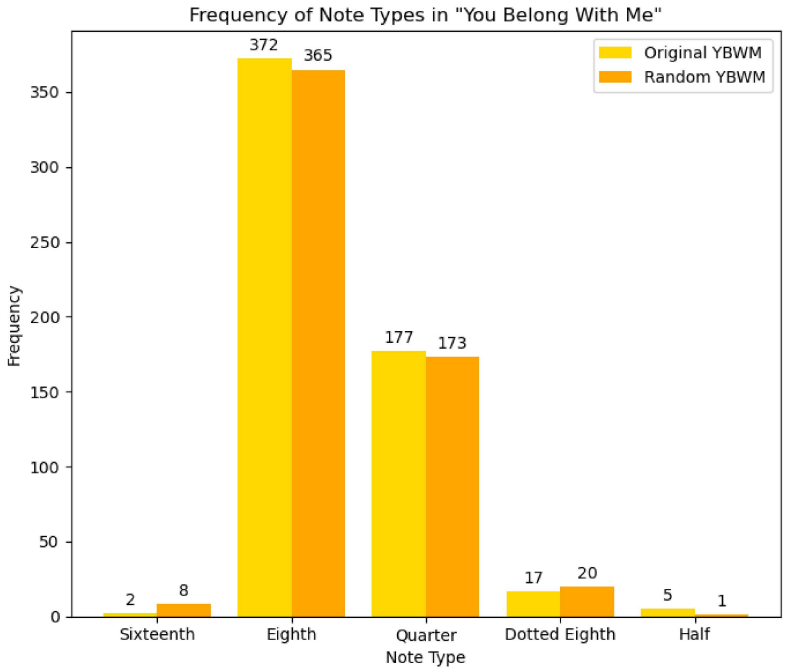


Figure 4.3. *Frequency of note lengths in both the original and newly generated "You Belong With Me."*



"Don't Blame Me"

The total amount of each note found in both the original song "Don't Blame Me" and the generated song based on "Don't Blame Me" can be found in Figure 4.4. The original composition is pink, while the newly generated song is red. Figure 4.5 exhibits the bar graph for both the original and generated songs, giving the total sum of each specific note type used. The bar graph that presents the total amount of each note length used in the original and newly generated songs can be seen in Figure 4.6.

Figure 4.4. *Frequency of notes in both the original and newly generated "Don't Blame Me."*

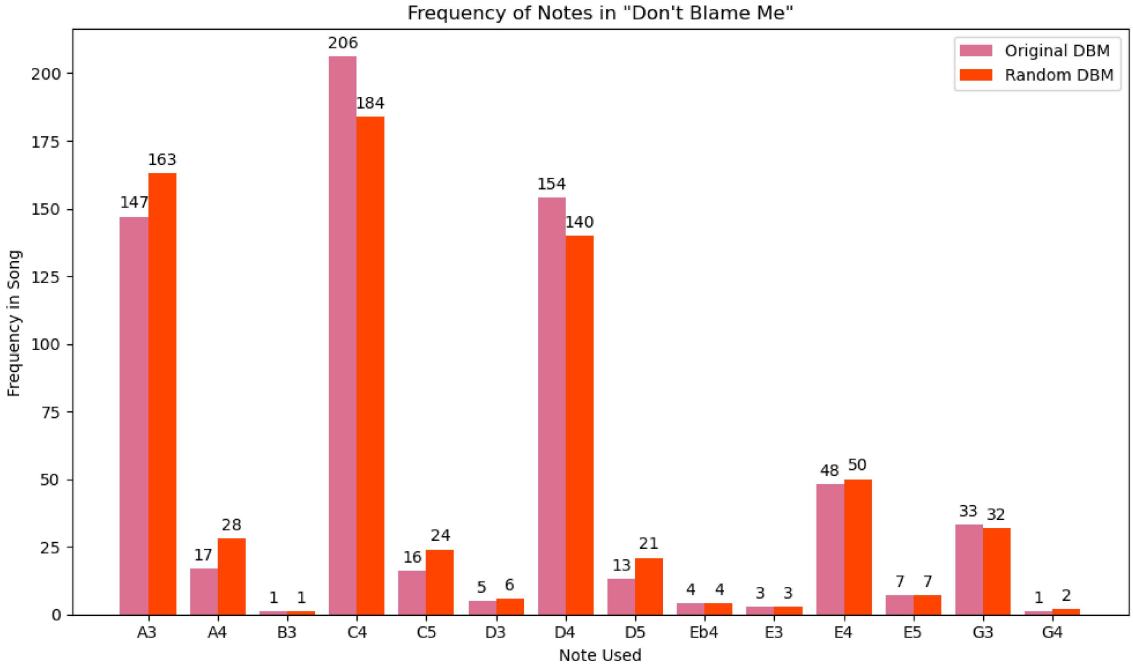


Figure 4.5. *Frequency of specific note types in both the original and newly generated "Don't Blame Me."*

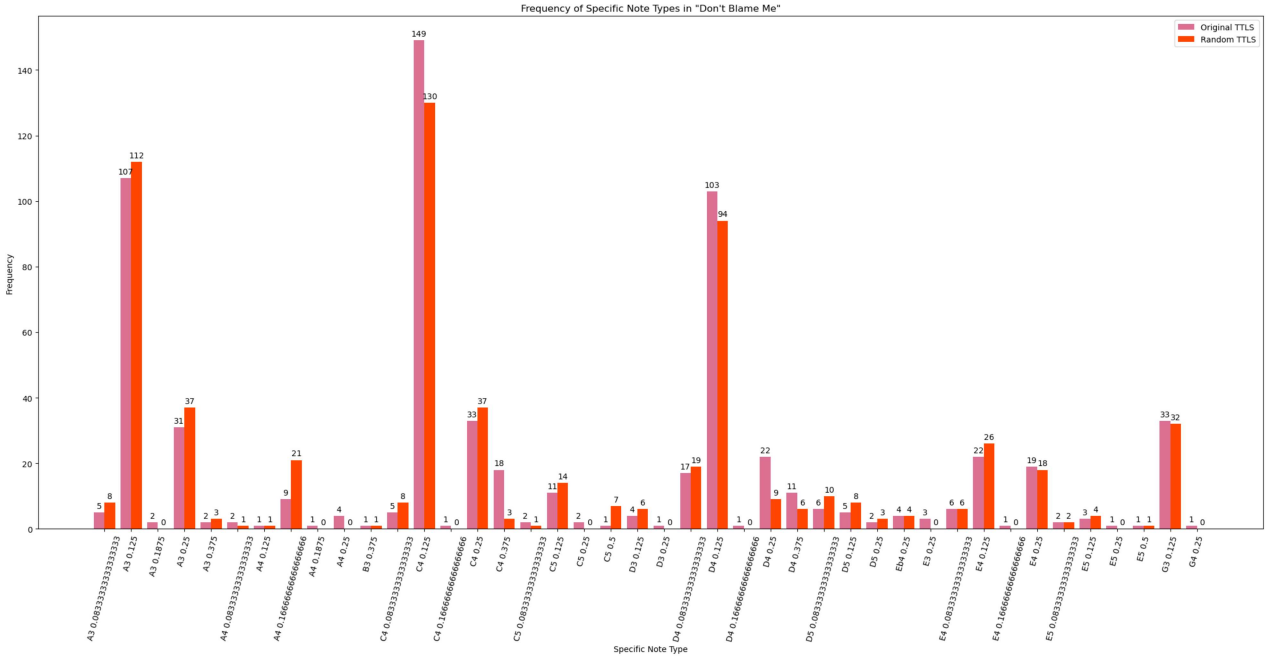
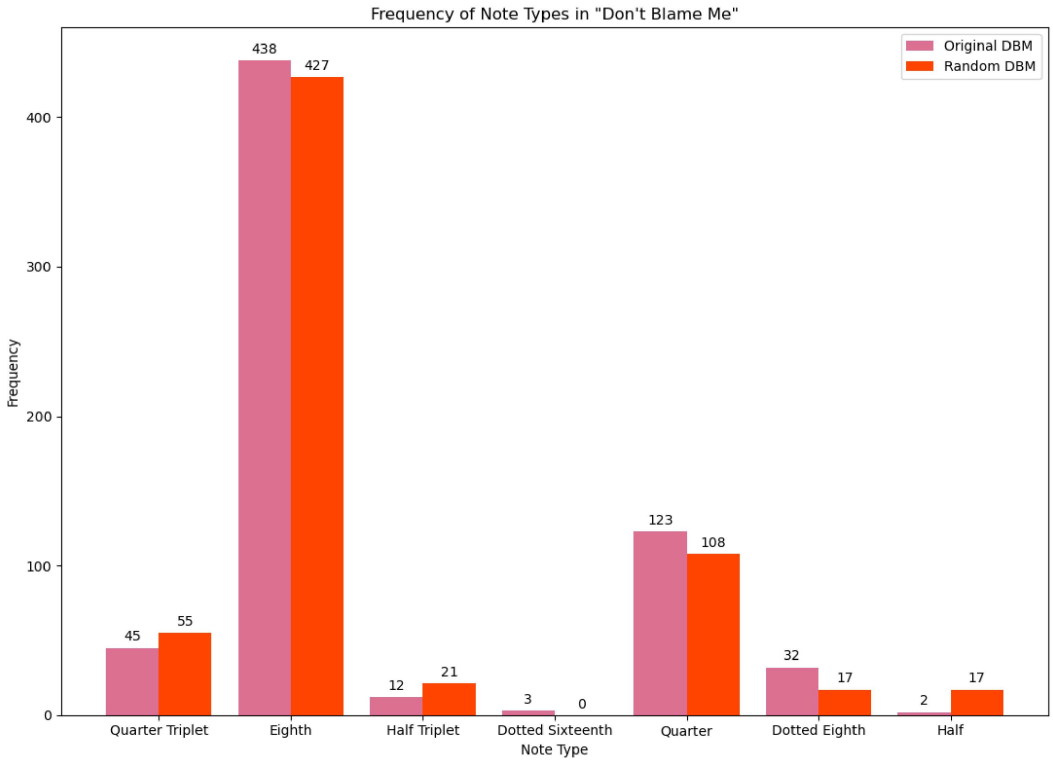


Figure 4.6. *Frequency of note lengths in both the original and newly generated "Don't Blame Me" song.*



Comparisons

Using the three pairs of bar graphs created for each song, we can compare the original song to its corresponding Markov Chain generated song.

"You Belong With Me"

Taking a look at the frequency of each note, each specific note type, and the length of the notes can provide information about how similar the songs are in the usage of notes throughout the whole song. Table 4.1 provides a side-by-side comparison of each note used in both song renditions. It can be seen that the two notes that stand out the most as being used more often in the newly generated "You Belong With Me" song are B4 and F#4. The three notes that are used less are A#3, B3, C#4, and G#3, while G#4 is used the same

number of times. The remaining notes fall within the range of −5 to 4 number of uses as the original song.

Table 4.1. *Total number of each note used in both the original and newly generated "You Belong With Me" songs.*

Note	**YBWM**	**New YBWM**
A#3	58	43
A#4	102	106
B3	29	21
B4	18	32
C#4	46	34
C#5	2	1
D#4	43	57
F#3	10	5
F#4	111	144
G#3	58	42
G#4	96	96

Table 4.2 shows the comparison of each specific note type used in both song renditions. Upon examination, we can see that the notes that are used significantly more in the generated song than the original composition, of at least 8 more occurrences, are D#4 0.125 and F#4 0.125, and the notes that are used significantly less, of at least 8 less occurrences, are A#3 0.25, C#4 0.125, and G#3 0.125. The only note that is used the same number of times in both song renditions is G#3 0.125. All of the other notes are within ±6 number of occurrences when comparing both songs.

As stated earlier, since rests are not considered as a state in the generated songs, we can notice that some of the totals do not sum up to the same number. For example, A#3 appears a total of 43 times in the new generated song, however there are only 31 A#3 0.125 notes, 7 A#3 0.25 notes, and 1 A#3 0.5 note, which add up to 39 total occurrences. Since there are no other possible states for A#3 than these three states, the other occurrences take

place when a rest length is added to the note length, making the length larger than any of the note possibilities. Again, since the goal of this project is to only look at the melody and notes, this complication it is not taken into consideration.

Table 4.2. *Total number of each specific note type used in both the original and newly generated "You Belong With Me" songs.*

Specific Note Type	**YBWM**	**New YBWM**
A#3 0.125	36	31
A#3 0.25	20	7
A#3 0.5	2	1
A#4 0.0625	1	4
A#4 0.125	64	61
A#4 0.25	36	37
A#4 0.375	1	4
B3 0.125	16	12
B3 0.25	13	9
B4 0.0625	1	4
B4 0.125	3	6
B4 0.25	2	8
B4 0.375	12	14
C#4 0.125	31	22
C#4 0.25	13	9
C#4 0.375	2	0
C#5 0.125	2	1
D#4 0.125	21	30
D#4 0.25	22	27
F#3 0.125	6	5
F#3 0.25	2	0
F#3 0.375	2	0
F#4 0.125	73	97
F#4 0.25	38	46
G#3 0.125	51	37
G#3 0.25	5	5
G#3 0.5	2	0
G#4 0.125	69	63
G#4 0.25	26	25
G#4 0.5	1	0

We can also compare the total number of each note lengths used in the songs. Table 4.3 gives this comparison. From this table, we can see that it is noticeable that most note lengths are used around almost the same number of times. The biggest differentiation being sixteenth notes, numerically 0.0625, used 6 more times in the generated song, and eight notes, numerically 0.125, used 7 times less in the generated song.

Table 4.3. *Total number of each notes' length used in both the original and newly generated "You Belong With Me" songs.*

Note Length	**YBWM**	**New YBWM**
0.0625	2	8
0.125	372	365
0.25	177	173
0.375	17	20
0.5	5	1

"Don't Blame Me"

For both the original "Don't Blame Me" and the generated song based on the original, Table 4.4 provides a side-by-side contrast of each note used. For the Markov Chain generated song, it is evident from the table that notes A3, A4, C5, and D5 are used at least 8 additional times, C4 and D4 occur at least 14 fewer times, and B3, Eb4, and E3 are used the same number of times. The remaining notes appear with comparable frequency for both songs, ranging from 1 less to 2 more occurrences.

Table 4.5 compares each specific note type used in both songs. A4 $0.1\overline{6}$ is the only note that occurs greater than 8 times more in the new song than the original song. C4 0.125, C4 0.375, D4 0.125, and D4 0.25 notes have at 8 fewer occurrences in the generated song than the original. Notes A4 0.125, B3 0.375, Eb4 0.25, E4 $0.08\overline{3}$, E5 $0.08\overline{3}$, and E5 0.5 appear the

same number of times in both songs. The remaining notes appear between 5 less to 6 more occurrences per note in the generated song than the original song.

Table 4.4. *Total number of each note used in both the original and newly generated "Don't Blame Me" songs.*

Note	**DBM**	**New DBM**
A3	147	163
A4	17	28
B3	1	1
C4	206	184
C5	16	24
D3	5	6
D4	154	140
D5	13	21
Eb4	4	4
E3	3	3
E4	48	50
E5	7	7
G3	33	32
G4	1	2

Once again, since rests are no longer considered as a state in the generated song, we notice that some of the totals do not sum up to the same number. However, since this project is only looking at melody, and not rests, this complication will not be dealt with at this time.

Finally, Table 4.6 gives this comparison of the total number for each note length used in both songs. The only note length that is close to having the same frequency for both songs is a dotted sixteenth note, numerically 0.1875, with this note appearing 3 fewer times in the generated song. The remaining notes are used within a range of 15 times less or more.

Table 4.5. *Total number of each specific note type used in both the original and newly generated "Don't Blame Me" songs.*

Specific Note Type	**DBM**	**New DBM**
A3 0.08$\bar{3}$	5	8
A3 0.125	107	112
A3 0.1875	2	0
A3 0.25	31	37
A3 0.375	2	3
A4 0.08$\bar{3}$	2	1
A4 0.125	1	1
A4 0.1$\bar{6}$	9	21
A4 0.1875	1	0
A4 0.25	4	0
B3 0.375	1	1
C4 0.08$\bar{3}$	5	8
C4 0.125	149	130
C4 0.1$\bar{6}$	1	0
C4 0.25	33	37
C4 0.375	18	3
C5 0.08$\bar{3}$	2	1
C5 0.125	11	14
C5 0.25	2	0
C5 0.5	1	7
D3 0.125	4	6
D3 0.25	1	0
D4 0.08$\bar{3}$	17	19
D4 0.125	103	94
D4 0.1$\bar{6}$	1	0
D4 0.25	22	9
D4 0.375	11	6
D5 0.08$\bar{3}$	6	10
D5 0.125	5	8
D5 0.25	2	3
Eb4 0.25	4	4
E3 0.25	3	0
E4 0.08$\bar{3}$	6	6
E4 0.125	22	26
E4 0.1$\bar{6}$	1	0
E4 0.25	19	18
E5 0.08$\bar{3}$	2	2
E5 0.125	3	4
E5 0.25	1	0
E5 0.5	1	1
G3 0.125	33	32
G4 0.25	1	0

Table 4.6. *Total number of each notes' length used in both the original and newly generated "Don't Blame Me" songs.*

Note Length	**DBM**	**New DBM**
$0.08\overline{3}$	45	55
0.125	438	427
$0.1\overline{6}$	12	21
0.1875	3	0
0.25	123	108
0.375	32	17
0.5	2	17

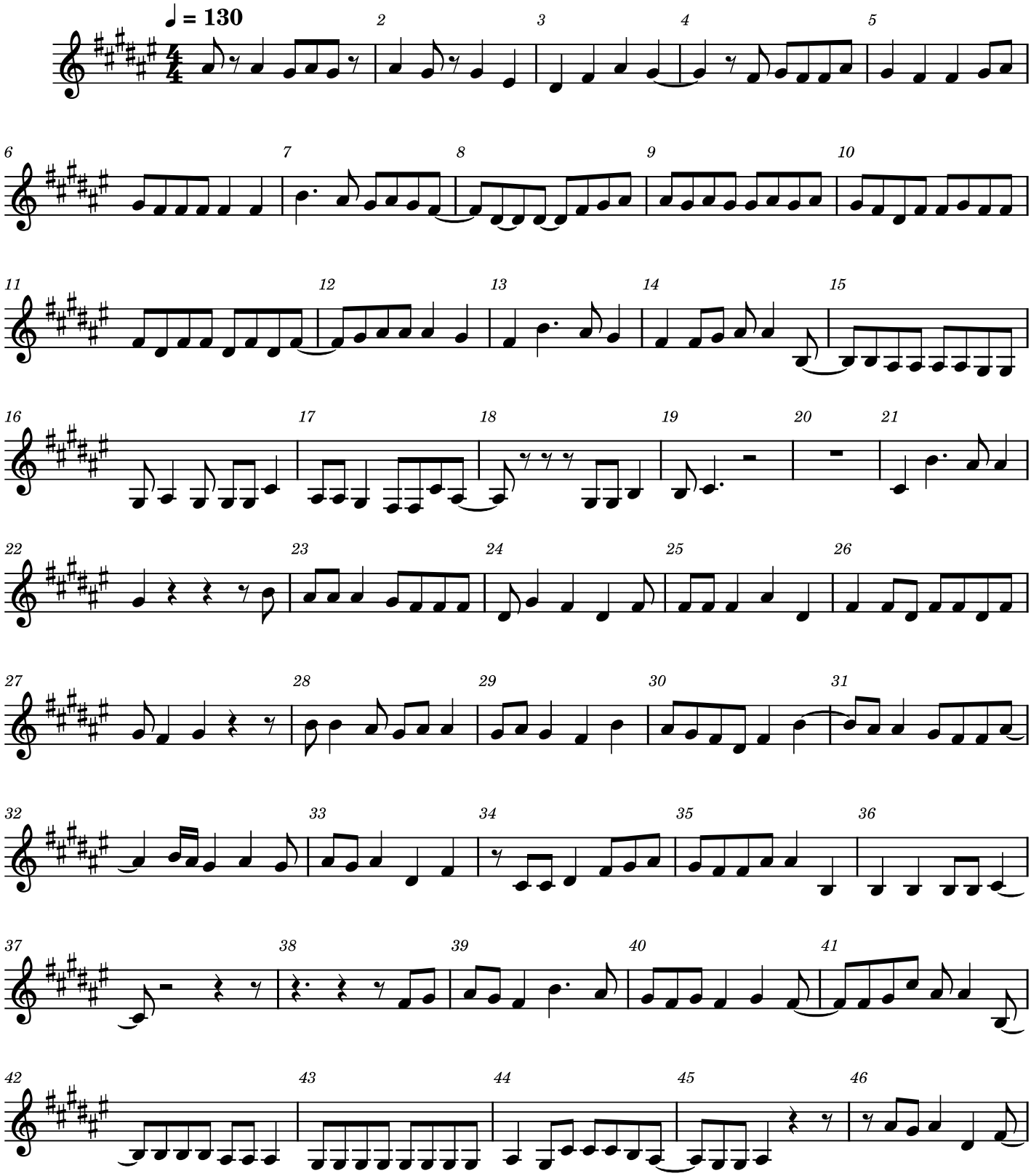
Overall, it can be easily seen that the generated song compositions were trained and based on their respective original song. Although the new pieces of music do not sound the same as the original, they have elements of repetition since the song was coded to use the same chorus and pre-chorus, just as the original compositions do. There are also pieces of the generated songs that sound identical to the original piece of music, showing that the Markov Chain created a fragment of the original song.

Typesetting

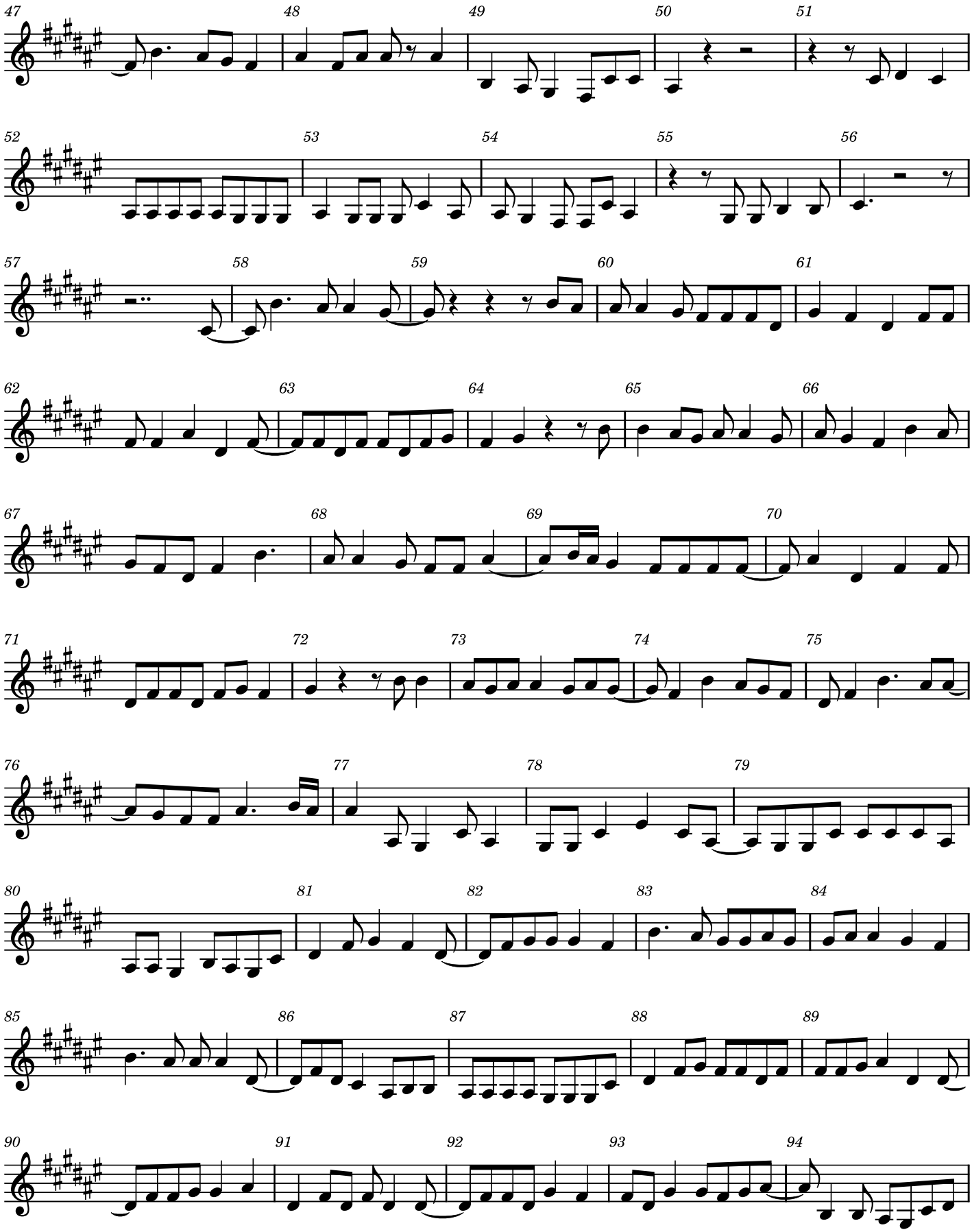
To visualize the newly generated song musically, we can typeset the ASCII notes back into musical notation on a staff. MuseScore was used for typesetting the newly generated "You Belong With Me" song. The piece was written and transposed by hand based on the transition states for the new song that was just generated. This method was only used for "You Belong With Me" and not with "Don't Blame Me," since it was a rigorous process. It can be noted here that a Python function could be created and utilized to automatically generate the sheet music for the new song. The sheet music for the newly generated song based on "You Belong With Me" can be seen on the page succeeding.

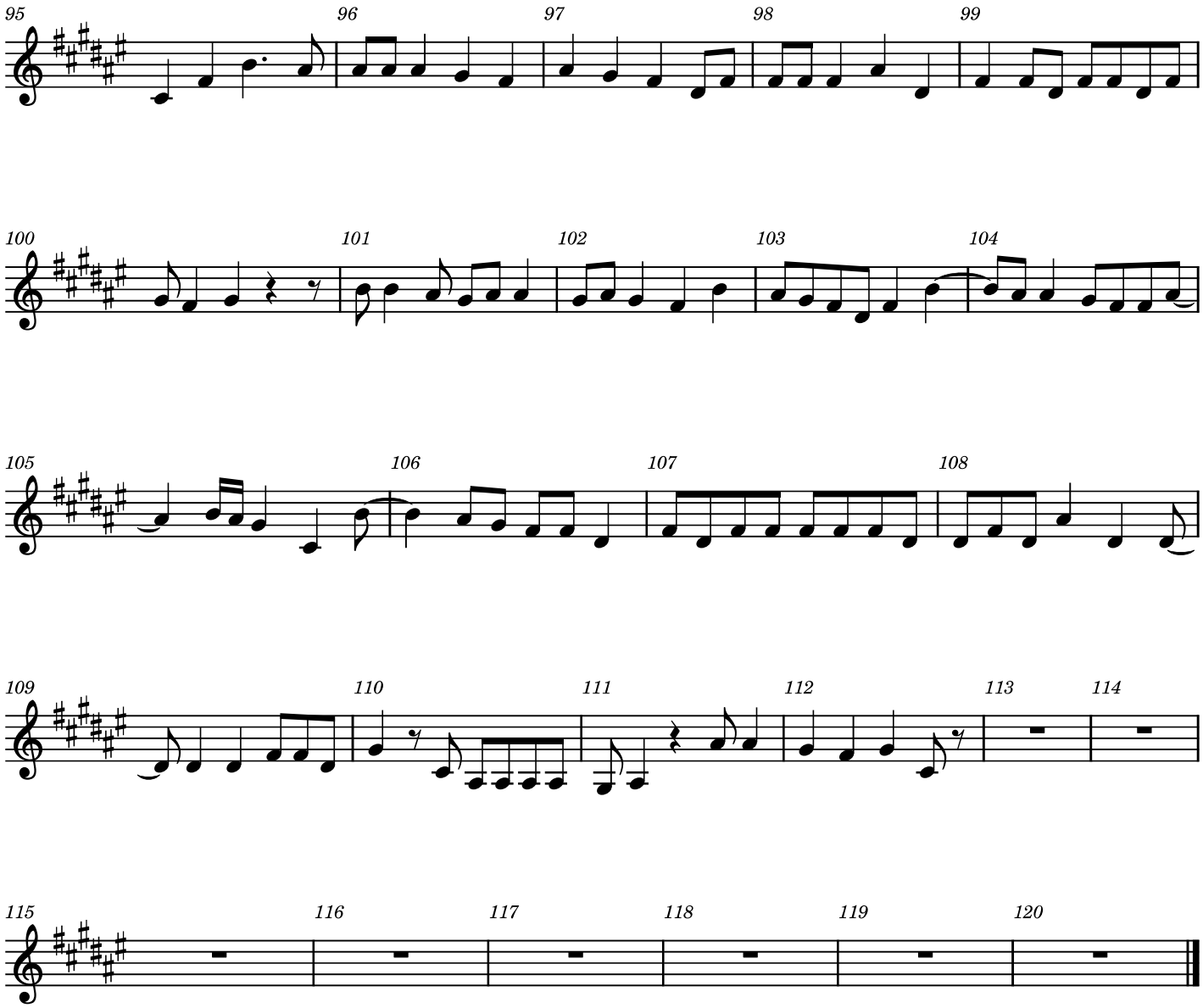
Newly Generated You Belong With Me

Transposed by Morgan Buterbaugh



2


47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94


95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120

CHAPTER 5

CURRENT SIMILAR WORK AND FUTURE DIRECTIONS

Music Generation Using ML and AI

Along with supervised ML, music can be generated using the three other ML models, unsupervised, semi-supervised, and reinforcement. Over the past year and a half, the Magenta team at Google Brain has been continuously working on developing code that would generate music using ML, leading to their project NSynth [16]. Using generative adversarial networks, the Magenta team is able to produce a ML model that generates polyphonic music, or music with chords. Built on top of Google Brain's previously established music maker, WaveNets, NSyth learns through reinforcement ML [16]. The user is able to combine different instruments to make new sounds, letting the model know if ouput audio is what was desired. Based on the user input, the generator then creates another audio file, which can then be tweaked by the user again.

Similar to working with stochastic models, music generation has been accomplished using online AI models. From the creators of ChatGPT, OpenAI introduced their new AI system, Jukebox, in 2019. Jukebox is an AI based music generator that can compose both musical aspects as well as lyrics. Since its release, it has been constantly undergoing updates and receiving upgrades [2]. The approach Jukebox takes is to use "quantization-based hierarchical VQ-VAEs" that compresses raw audio files into three levels [10]. Once the audio file is sorted, the model is then trained using Sparse Transformers and against over 1.2 million songs [2]. The model can become more accurate if the artist, genre, bpm, and time signature are provided, as well as lyrics can also be used to condition the model [10]. Jukebox provides re-renditions of songs, new samples, and novel styles that are all created behind the scenes using AI and unsupervised reinforced ML. Jukebox AI can be used in Google Collab. The user uploads a MIDI file of a song by any artist or genre, enters an artist and genre of choosing for the re-rendition and can optionally add lyrics [9]. The Python notebook will then

do all the work and present the user with new audio files based on their inputs. Some other AI music generators include Musenet, also owned by OpenAI but not as well formulated, Boomy, Soundraw, AIVA, Amper Music, and Ecrett Music [9].

Potential Courses of Action

Further directions and continuation of the research could involve numerous other methods and discoveries. Perusing the idea of continuing with the project, the next step would be to work out the issues for the rest states. Formatting the code so that 'Musicpy,' or even another music library in Python, can read and comprehend a rest would solve the problem of rests getting combined with the note states. Although fixing this issue would not change the way the new song sounds, it would allow for a more accurate statistical analysis of the songs' structure, such as the number of states, frequency of each specific note type, and the lengths of the notes. Another idea of continuation would be once each newly generated song is produced, a function could be written in Python so that it automatically creates the sheet music for that particular generated song. This automation would then allow for the comparison of sheet music, completing the process in a shorter amount of time than it would take to create the sheet music by hand. One major potential course of action would be to use MIDI files. Instead of having to manually enter the song into Python, using a MIDI file would immensely speed up the progression. It would also allow for more songs to be used, since the user would not have to know about music theory and ASCII formatting.

Many other alternatives and research questions could be asked and explored. Since the algorithm for generating new songs worked when using a one state transition, the next logical step would be to work with two state transitions. This would involve using a second order Markov Chain, where the first set of two states of the chain are the first two notes of the song, the second set of two states would be the second and third notes of the song, the third set of two states would be the third and fourth notes of the song, etc. Subsequently, a higher-order Markov Chain could be used as well, using n previous notes. Using more than

just one transition state can help to capture even more intricate dependencies in the music data, potentially leading to compositions that better resemble the original given song. Thus, the fewer transition states, the less the generated song will create a sound that resembles the original given song. Another question and future exploration could be to determine the total number of states that are necessary to ensure that the generated song will give back the original song. Thus giving a result that is the same as the original composition, providing optimal model complexity.

When it comes to the statistical analysis of the original and newly generated songs, there are many more ways to compare the two. Extending on what is already completed in this project, a third column could be added to the comparison. This column would contain the data from a generation that would run for a longer amount of time, i.e. 100 times the number of states in the original song. We could then normalize the results across states and compare them verses the original song. Without having actually completed this idea, it is hypothesised that you would see the original song statistics would be matched pretty well against the newly lengthened generated song. This leads to question of 'What is a good metric for the distance between the original song and the stochastically generated version?' Hence, what is the true distance between the songs. To compute this distance and find the similarity between the original and newly generated song, we would need to find a way to measure how far apart the new songs are from the song seed. This process would involve looking at the norm of the transition matrix for both song renditions. From this, the Euclidean distance between the two song renditions can then be used to measure the similarity [11].

Beyond working with melody, feature engineering could be implemented into the model, incorporating additional features beyond note sequences. This could include features such as chord progressions, bass line, rhythm patterns, dynamics, instrument timbres, and multiple instrumentation. Features can be used in conjunction with Markov Chains, or as inputs to

ML models, to enhance the modeling capabilities and generate more nuanced and expressive music. Hybrid modeling is another method that could be executed on this project model. Hybrid modeling combines Markov Chains with other ML techniques such as recurrent neural networks (RNN) or generative adversarial networks (GAN). Other possible networks to focus on using and implementing for this project could be long short-term memory (LSTM), multi-style chord music generation (MSCMG), or hidden Markov model (HMM) [19]. Employing other ML techniques on this established project model can leverage the strengths of different approaches, potentially improving the quality and diversity of the generated music.

CHAPTER 6

CONCLUSION

Using supervised machine learning and stochastic processes, new music was able to be produced when the model was trained on a given piece of music. Utilizing a seeding piece of music to create a Markov Chain, compositions of a new originally unique musical piece was able to be generated. When using the notes and rests as states, the probability of the following state was estimated from the seeding composition. Hence, the model was allowed to compose its own music using stochastic state transitions.

Even though this process did not produce a melody as catchy and definitely not as successful as Taylor Swift's music, it is still interesting to listen to and compare with her original songs. This same process could be used for any artist or song, leading to an infinite amount of new musical compositions. Although the results may not be the desired result, there is always something to learn about ML algorithms.

ML can be, and is proving itself to be, a great tool for current and future music composition. Whether it be some form of the four ML techniques, stochastic processes, or AI, there are many approaches to generating music with technology. It is an evolving field that leverages computational algorithms to create music autonomously or in collaboration with human composers. This technology employs various ML techniques, including deep learning, neural networks, and reinforcement learning, to analyze musical data and extract patterns, styles, and structures. By training models on existing musical compositions, these algorithms can generate new pieces that mimic specific genres, artists, or musical characteristics.

ML for music generation has applications in various domains, including music composition, sound design, film scoring, video game development, and even personalized music recommendation systems. Continued research and innovation in this field lead to a promising way for transforming the way music is created, experienced, and shared in the digital age.

References

1. *Stochastic Music.* Sweetwater, https://www.sweetwater.com/insync/stochastic-music/, 2004. Accessed September 22, 2023.

2. *Jukebox.* OpenAI, https://openai.com/research/jukebox, 2020. Accessed February 18, 2024.

3. *Introduction to Machine Learning.* Google, https://developers.google.com/machine-learning/intro-to-ml, 2023. Accessed September 22, 2023.

4. *What is Machine Learning (ML)?* IBM, https://www.ibm.com/topics/machine-learning, 2023. Accessed September 22, 2023.

5. *What Is Machine Learning? Definition, Types, and Examples.* Coursera, https://www.coursera.org/articles/what-is-machine-learning, 2024. Accessed January 5, 2024.

6. S. Agnew, *Generating music with Python and Neural Networks using Magenta for Tensorflow.* Twilio, https://www.twilio.com/en-us/blog/generate-music-python-neural-networks-magenta-tensorflow, 2018. Accessed November 8, 2023.

7. A. Bainter, *Generating More of My Favorite Aphex Twin Track.* Medium, https://medium.com/@alexbainter/generating-more-of-my-favorite-aphex-twin-track-cde9b7ecda3a, 2018. Accessed December 19, 2023.

8. S. Brown, *Machine learning, explained.* MIT Sloan School of Management, https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained, 2021. Accessed September 23, 2023.

9. A. Dawood, *What Is OpenAI Jukebox and How to Use it?* Mlyearning, https://www.mlyearning.org/openai-jukebox/, 2023. Accessed February 18, 2023.

10. P. Dhariwal, H. Jun, J. W. Kim, C. Payne, A. Radford, and I. Sutskever, *Jukebox: A Generative Model for Music.* Cornell University, https://arxiv.org/pdf/2005.00341.pdf, 2020. Accessed February 18, 2023.

11. L. Fernando, *Using Markov Chains to Extract Song Similarity.* Medium, https://towardsdatascience.com/progressions-eb79a573f7f1, 2018. Accessed December 19, 2024.

12. D. M. Franz, *Markov Chains as Tools for Jazz Improvisation Analysis*, Master's thesis, Virginia Polytechnic Institute and State University, 1998. Accessed January 8, 2024.

13. J. Gage, *He touched a nerve: How the first piece of AI music was born in 1956.* The Guardian, https://www.theguardian.com/music/2021/dec/07/he-touched-a-nerve-how-the-first-piece-of-ai-music-was-born-in-1956, 2021. Accessed October 10, 2023.

14. A. Gaur, *Iannis Xenakis.* Encyclopedia Britannica, https://www.britannica.com/biograhy/Iannis-Xenakis, 2024. Accessed January 11, 2024.

15. M. Huber and I. Shapiro, *Markov Chains for Computer Music Generation.* Journal of Humanistic Mathematics, 11 (2021), pp.167-195. Accessed September 22, 2023.

16. D. Kabra, *How AI evolved in the world of music: From Markov Chains to LSTMs and NSynths.* Medium, https://medium.com/version-1/how-ai-evolved-in-the-world-of-music-from-markov-chains-to-lstms-and-nsynths-cf0343ad1c9a, 2023. Accessed November 17, 2023.

17. T. Kathiresan, *Automatic Melody Generation*, Master's thesis, KTH Royal Institute of Technology, 2015. Accessed October 25, 2023.

18. M. Kuuskankare, M. Laurson, and Örjan Sandred, *Revisiting the Illiac Suite - A rule-based approach to stochastic processes.* Sonic Ideas/Ideas Sonicas, 2 (2009), pp. 42-46. Accessed September 27, 2023.

19. F. Li, *Chord-based music generation using long short-term memory neural networks in the context of artificial intelligence.* The Journal of Supercomputing, 80 (2023), pp. 6068-6092. Accessed February 14 2024.

20. A. Lin, *Generating Music Using Markov Chains.* Medium, https://medium.com/hackernoon/generating-music-using-markov-chains-40c3f3f46405, 2016. Accessed September 23, 2023.

21. E. J. Linskens, *Music Improvisation using Markov Chains*, Master's thesis, University College Maastricht Maastricht University, 2014. Accessed September 21, 2023.

22. B. Madhukar, *Hands-On Guide To Markov Chain For Text Generation.* AI Mysteries, https://analyticsindiamag.com/hands-on-guide-to-markov-chain-for-text-generation/, 2021. Accessed February 8, 2024.

23. N. U. Maulidevi and A. S. Ramanto, *Markov Chain Based Procedural Music Generator with User Chosen Mood Compatibility.* International Journal of Asia Digital Art & Design, 21 (2017), pp. 19-24. Accessed September 24, 2023.

24. S. Mukherjee and A. K. Yanchenko, *Classical Music Composition Using State Space Models.* Duke University, https://arxiv.org/pdf/1708.03822.pdf, 2018. Accessed October 11, 2023.

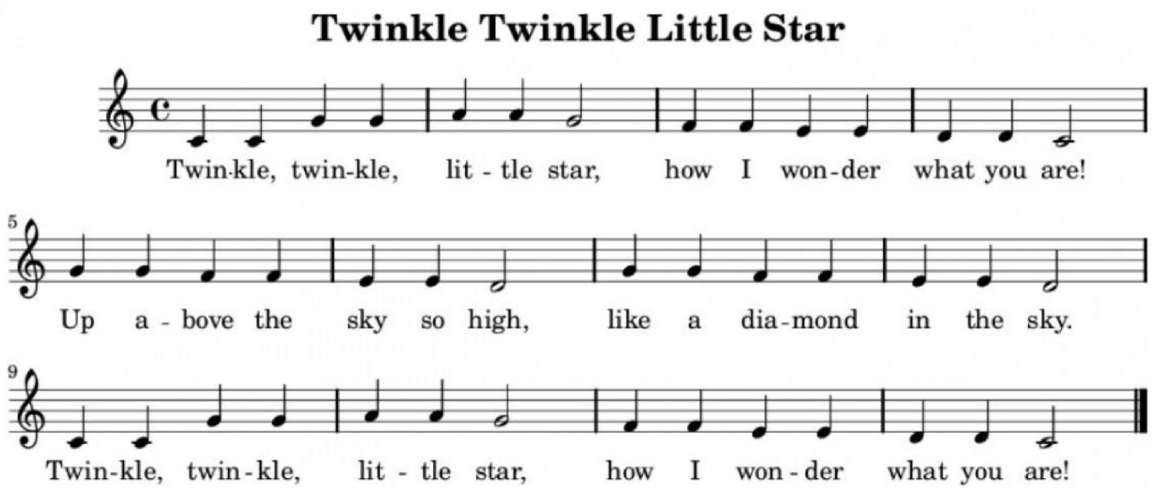
25. A. Osipenko, *Markov Chain for music generation.* Medium, https://towardsdatascience.com/markov-chain-for-music-generation-932ea8a88305, 2019. Accessed October 11, 2023.

26. R. Sekhon, *Applied Finite Mathematics.* LibreTexts, 2011, pp. 291-292, https://math.libretexts.org/Bookshelves/Applied-Mathematics/Applied-Finite-Mathematics-(Sekhon-and-Bloom). Accessed September 22, 2023.

27. W. L. Winston, *Operations Research Applications and Algorithms*. Thomson Brooks/ Cole, Pacific Grove, CA, Fourth ed., 2003, pp. 924-927. Accessed September 15, 2023.

Appendix A

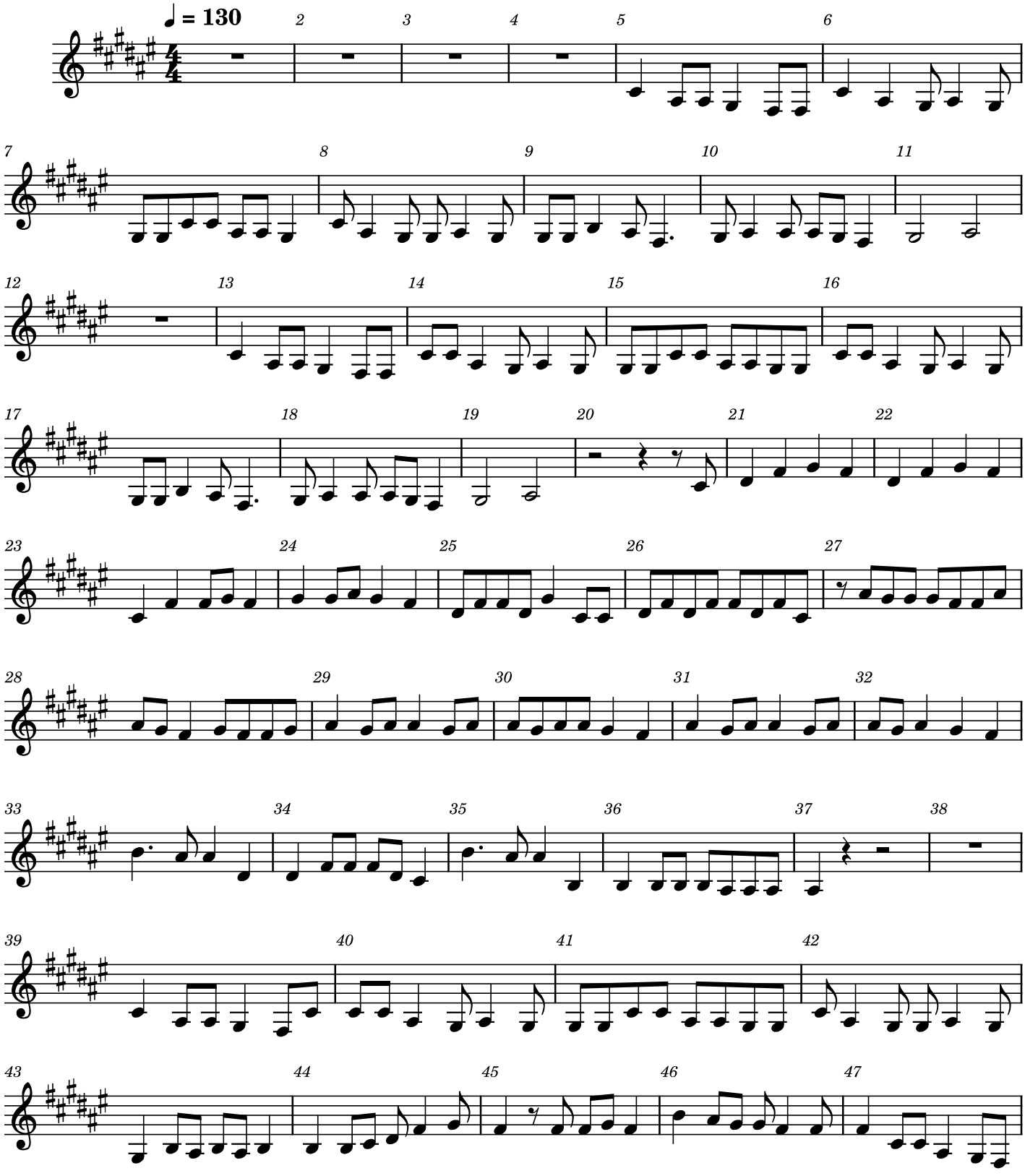
Sheet Music



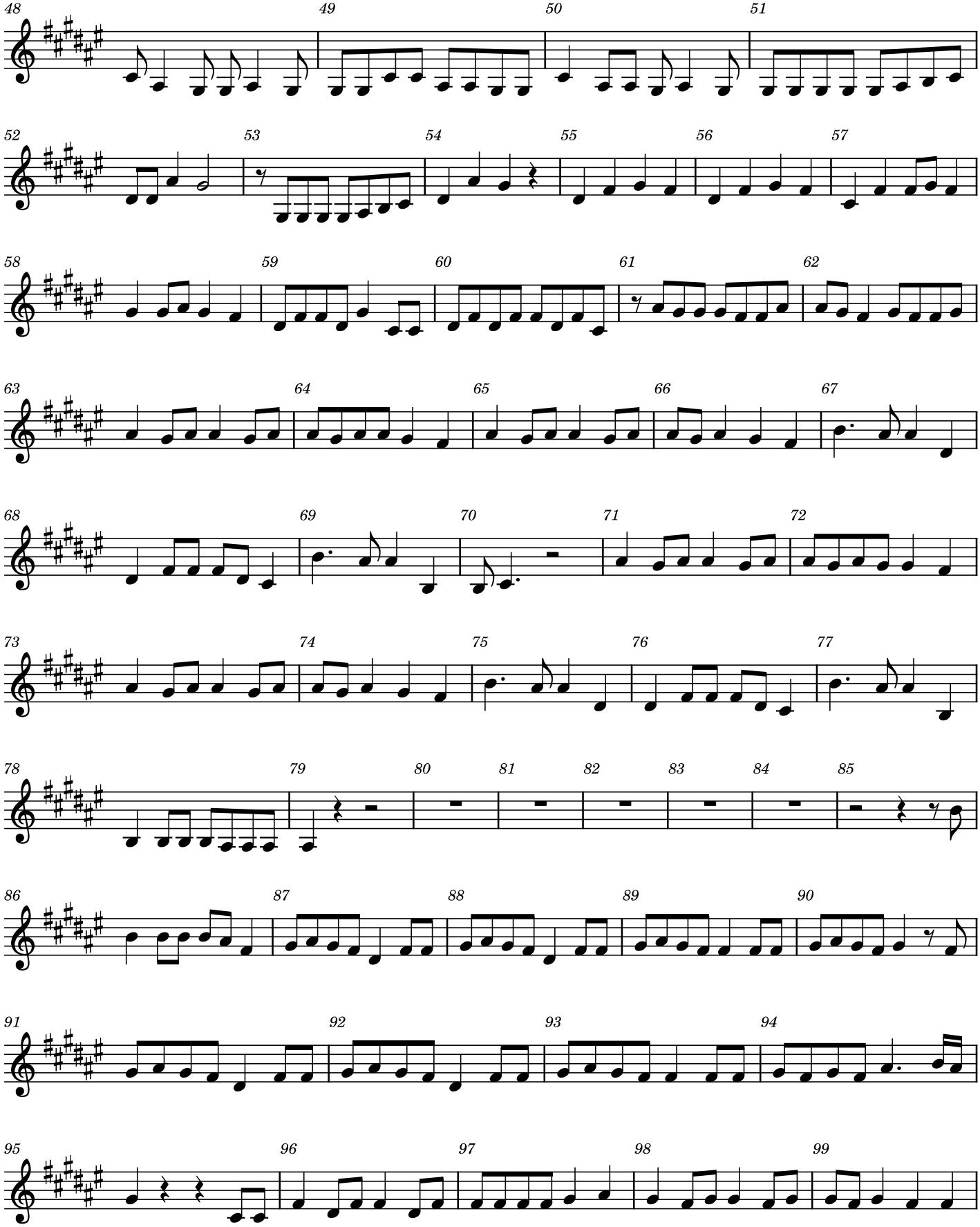
You Belong With Me

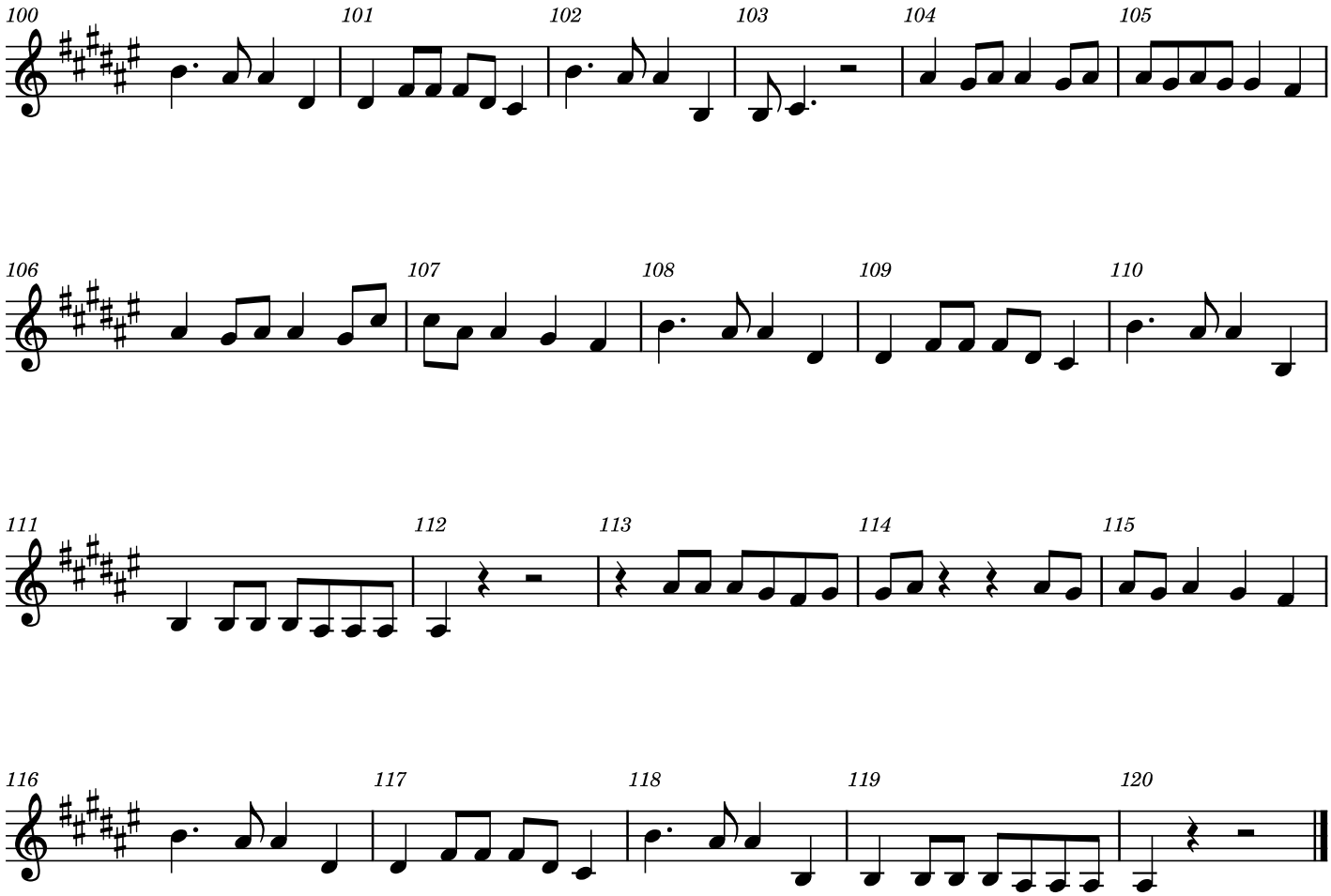
Transposed by Morgan Buterbaugh

Words and Music by Taylor Swift
and Liz Rose



2

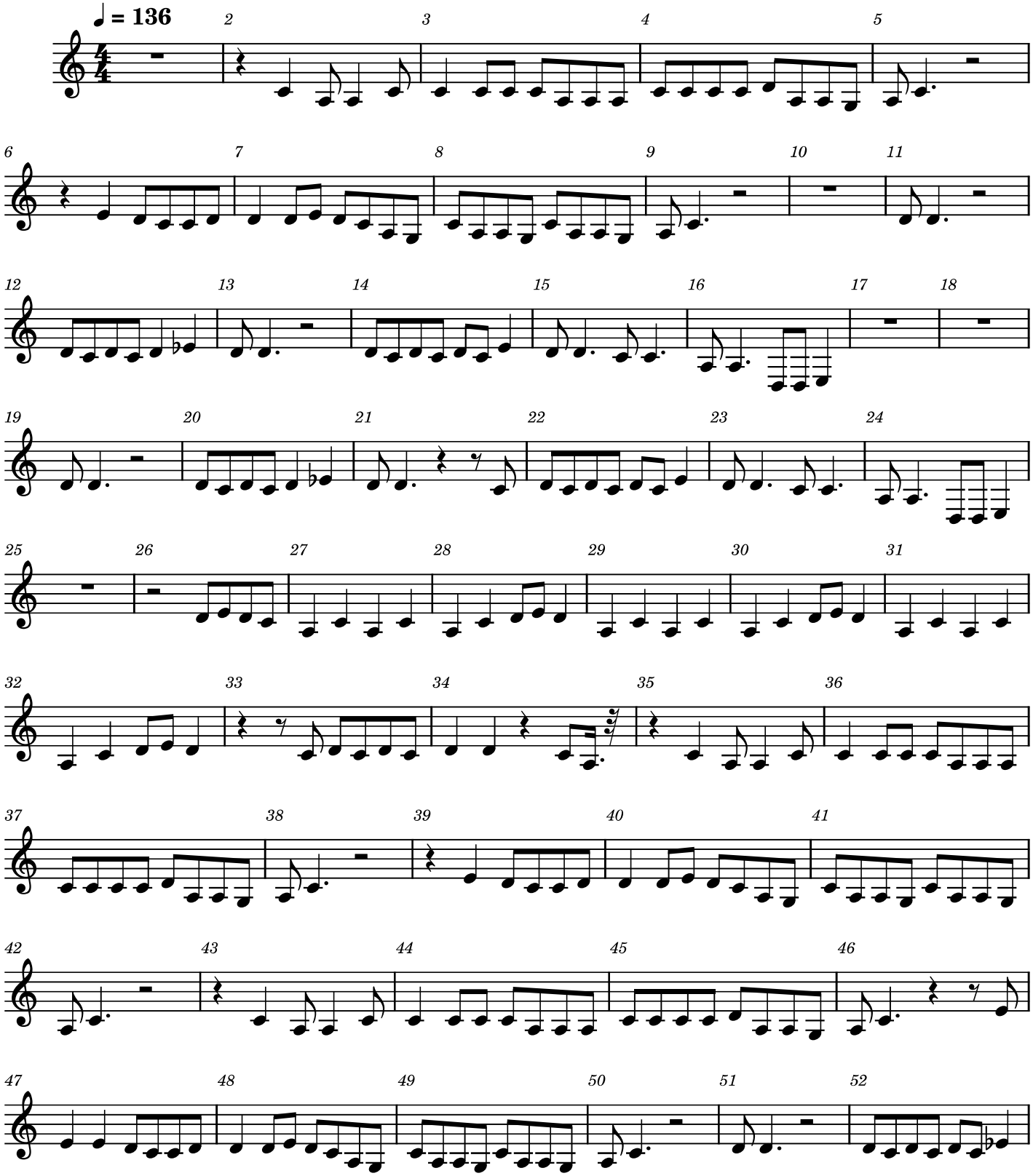

48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99


100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120

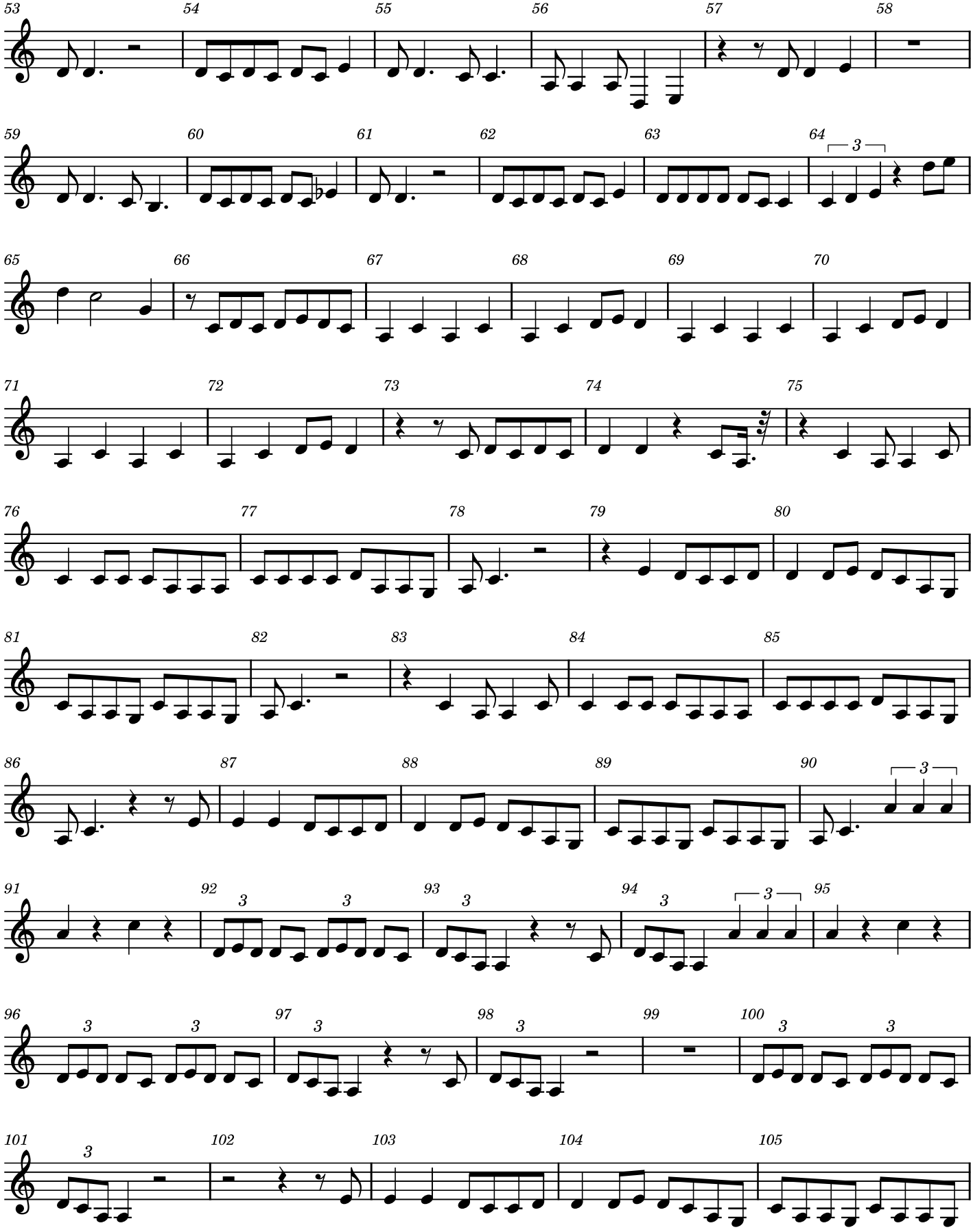
Don't Blame Me

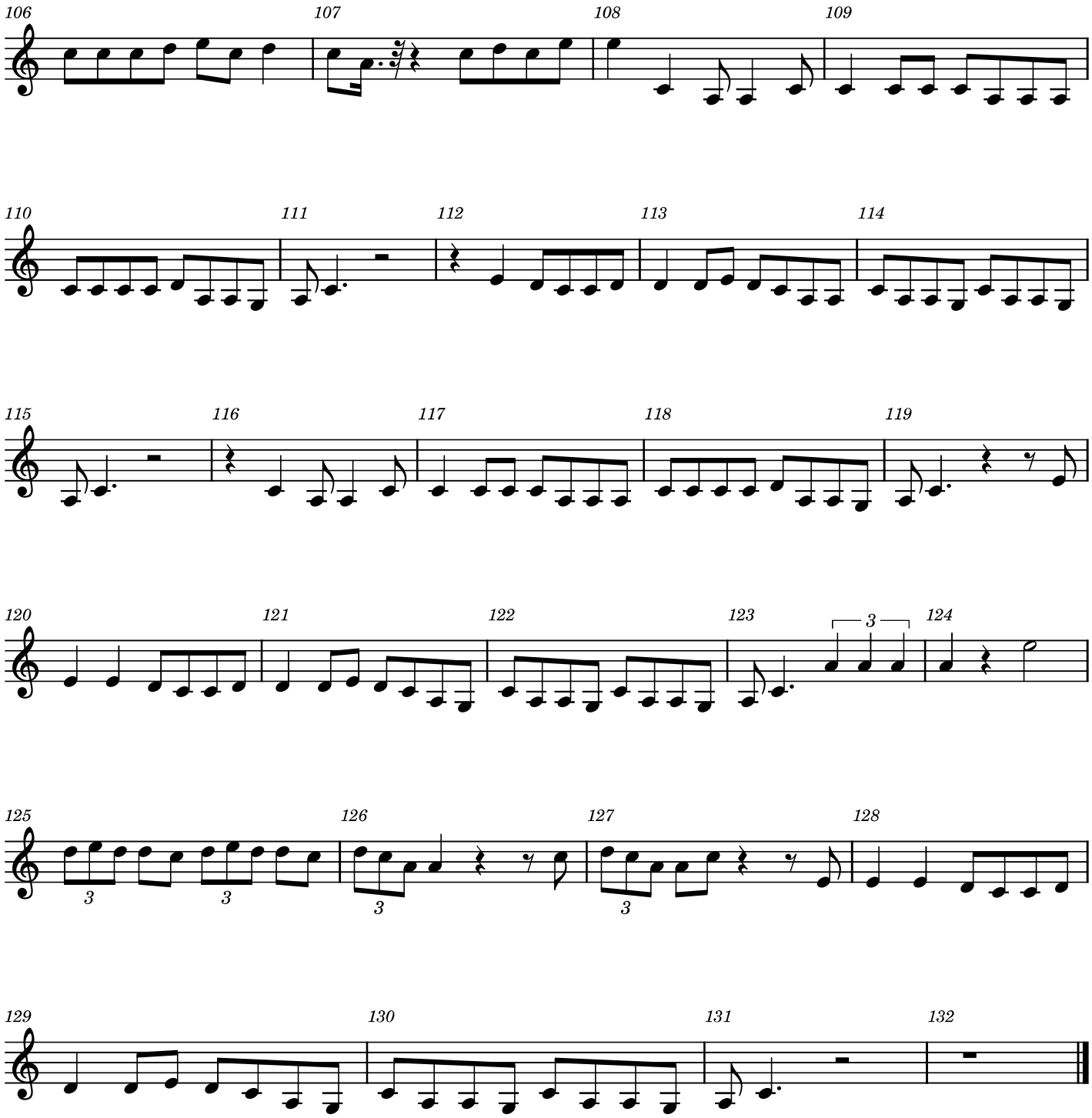
Morgan Buterbaugh

Words and Music by Taylor Swift,
Max Martin, and Shellback



2


53
54
55
56
57
58
59
60
61
62
63
64
3
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
3
91
92
3
3
93
3
94
3
3
95
96
3
3
97
3
98
3
99
100
3
3
101
3
102
103
104
105


106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
3
124
125
3
3
126
3
127
3
128
129
130
131
132

Appendix B

"You Belong With Me" State Transition Matrix

	Rest eighth	Rest quarter	Rest half	Rest whole	F#3 eighth	F#3 quarter	F#3 dotted quarter
Rest eighth							
Rest quarter	2	2	3				
Rest half		3		2			
Rest whole			1	7			
F#3 eighth					2		
F#3 quarter							
F#3 dotted quarter							
G#3 eighth					1	2	
G#3 quarter					3		
G#3 half							
A#3 eighth							2
A#3 quarter		3					
A#3 half			1	1			
B3 eighth							
B3 quarter							
C#4 eighth	2						
C#4 quarter							
C#4 dotted quarter			2				
D#4 eighth							
D#4 quarter							
F#4 eighth							
F#4 quarter	1						
G#4 eighth							
G#4 quarter	1	2					
G#4 half	1						
A#4 sixteenth							
A#4 eighth		1					
A#4 quarter							
A#4 dotted quarter							
B4 sixteenth							
B4 eighth							
B4 quarter							
B4 dotted quarter							
C#5 eighth							
	7	11	7	10	6	2	2

	G#3 eighth	G#3 quarter	G#3 half	A#3 eighth	A#3 quarter	A#3 half	B3 eighth	B3 quarter
Rest eighth	1							
Rest quarter								
Rest half								
Rest whole								
F#3 eighth								
F#3 quarter			2					
F#3 dotted quarter	2							
G#3 eighth	26	1		2	10			2
G#3 quarter							1	
G#3 half						2		
A#3 eighth	6	4		18	4		3	1
A#3 quarter	16			2				
A#3 half								
B3 eighth				6			8	
B3 quarter				2			7	5
C#4 eighth				4	7			
C#4 quarter				4	1			
C#4 dotted quarter								
D#4 eighth								
D#4 quarter								
F#4 eighth								
F#4 quarter								
G#4 eighth								
G#4 quarter								
G#4 half								
A#4 sixteenth								
A#4 eighth								
A#4 quarter								6
A#4 dotted quarter								
B4 sixteenth								
B4 eighth								
B4 quarter								
B4 dotted quarter								
C#5 eighth								
	51	5	2	38	22	2	19	14

	C#4 eighth	C#4 quarter	C#4 dotted quarter	D#4 eighth	D#4 quarter	F#4 eighth	F#4 quarter
Rest eighth	1					2	
Rest quarter	1				1		
Rest half							
Rest whole		3					
F#3 eighth	3	1					
F#3 quarter							
F#3 dotted quarter							
G#3 eighth	6	1					
G#3 quarter	1						
G#3 half							
A#3 eighth							
A#3 quarter							
A#3 half							
B3 eighth	3		2				
B3 quarter							
C#4 eighth	12			4	2		1
C#4 quarter							2
C#4 dotted quarter							
D#4 eighth		5		1		10	1
D#4 quarter		1			6	10	4
F#4 eighth	2			12	4	31	4
F#4 quarter	1	2		4	2	5	1
G#4 eighth						15	7
G#4 quarter	2					2	16
G#4 half							
A#4 sixteenth							
A#4 eighth							1
A#4 quarter					6		
A#4 dotted quarter							
B4 sixteenth							
B4 eighth							
B4 quarter							
B4 dotted quarter							
C#5 eighth							
	32	13	2	21	21	75	37

	G#4 eighth	G#4 quarter	G#4 half	A#4 sixteenth	A#4 eighth	A#4 quarter	A#4 dotted quarter
Rest eighth					2		
Rest quarter					2		
Rest half						2	
Rest whole							
F#3 eighth							
F#3 quarter							
F#3 dotted quarter							
G#3 eighth							
G#3 quarter							
G#3 half							
A#3 eighth							
A#3 quarter							
A#3 half							
B3 eighth							
B3 quarter							
C#4 eighth							
C#4 quarter							
C#4 dotted quarter							
D#4 eighth		2				1	
D#4 quarter						1	
F#4 eighth	16	3			2		1
F#4 quarter	4	6				4	
G#4 eighth	7	3			30	6	
G#4 quarter	2					1	
G#4 half							
A#4 sixteenth		1					
A#4 eighth	24	4			13	21	
A#4 quarter	16	7	1				
A#4 dotted quarter							
B4 sixteenth				1			
B4 eighth					1		
B4 quarter					1		
B4 dotted quarter					12		
C#5 eighth					1		
	69	26	1	1	64	36	1

	B4 sixteenth	B4 eighth	B4 quarter	B4 dotted quarter	C#5 eighth	
Rest eighth		1				7
Rest quarter						11
Rest half						7
Rest whole						11
F#3 eighth						6
F#3 quarter						2
F#3 dotted quarter						2
G#3 eighth						51
G#3 quarter						5
G#3 half						2
A#3 eighth						38
A#3 quarter						21
A#3 half						2
B3 eighth						19
B3 quarter						14
C#4 eighth						32
C#4 quarter				6		13
C#4 dotted quarter						2
D#4 eighth						20
D#4 quarter						22
F#4 eighth						75
F#4 quarter			1	6		37
G#4 eighth					1	69
G#4 quarter						26
G#4 half						1
A#4 sixteenth						1
A#4 eighth						64
A#4 quarter						36
A#4 dotted quarter	1					1
B4 sixteenth						1
B4 eighth		2	1			4
B4 quarter		1				2
B4 dotted quarter						12
C#5 eighth					1	2
	1	4	2	12	2	618
						618

Appendix C

Code

Listing 6.1. "Twinkle Twinkle Little Star"

```
from musicpy import *

TTLS = chord('C4 C4 G4 G4 A4 A4 G4 F4 F4 E4 E4 D4 D4 C4 G4 G4 F4 F4 E4 E4 D4 G4 G4 F4 F4 E4 E4 D4 C4 C4 G4 G4 A4 A4 G4 F4 F4 E4 E4 D4 D4 C4',
             interval=[1/4, 1/4, 1/4, 1/4, 1/4, 1/4, 1/2, 1/4, 1/4, 1/4, 1/4, 1/4, 1/4, 1/2, 1/4, 1/4,
                  1/4, 1/4, 1/4, 1/4, 1/2, 1/4, 1/4, 1/4, 1/4, 1/4, 1/4, 1/2, 1/4, 1/4, 1/4, 1/4, 1/4,
                  1/4, 1/2, 1/4, 1/4, 1/4, 1/4, 1/4, 1/4, 1/2])

play(TTLS,bpm=117)
```

Listing 6.2. Analyzing Original "Twinkle Twinkle Little Star"

```
from musical import *
import matplotlib.pyplot as plt

#### Analysis of Original Song
###### Only the notes:

def PlotNoteFrequencies(NoteList):
    # Order notes
    note_order = ['A4', 'C4', 'D4', 'E4', 'F4', 'G4']

    # Create an empty dictionary
    SongDict = {}

    # Set assign the notes as dictionary keys:
    for note in note_order:
        SongDict[note] = 0

    # Loop over the given note list again to give a value to the number of occurrences
    for note in NoteList:
        snote = str(note)
        if snote in SongDict:
            SongDict[snote] = SongDict[snote]+1

    print(SongDict)

    # Sort notes and frequencies
    Notes     = list(SongDict.keys())
    Frequency = [SongDict[note] for note in Notes]

    # create bar plot
    fig, ax = plt.subplots(figsize = (10, 5))
    bars = ax.bar(Notes, Frequency)

    for i in range(len(Notes)):
        ax.text(bars[i].get_x() + bars[i].get_width() / 2, bars[i].get_height(), str(Frequency[i]), ha='
            center', va='bottom')

    plt.xlabel("Note Used")
    plt.ylabel("Frequency in Song")
    plt.title("Frequency of Notes in Twinkle Twinkle Little Star")
    plt.xticks(rotation = 45)
    plt.show()
    fig.savefig("Original TTLS Note Frequencies.png", bbox_inches='tight')

PlotNoteFrequencies(TTLS.notes)


###### Notes and their respective length:

formatted_notes = ['-'.join([str(note), str(interval)]) for note, interval in zip(TTLS, TTLS.interval)]
print(formatted_notes)

def PlotSpecificNoteTypeFrequencies(NoteList):
```

```
    # Order specific notes
    specific_note_order = ['A4-0.25', 'C4-0.25', 'C4-0.5', 'D4-0.25', 'D4-0.5', 'E4-0.25', 'F4-0.25', 'G4-
        0.25', 'G4-0.5']

    # Creating an empty dictionary
    SongDict = {}

    # Set assign the notes as dictionary keys:
    for note in specific_note_order:
        SongDict[note] = 0

    # Loop over the given note list again to give a value to the number of occurrences.
    for note in NoteList:
        snote = str(note)
        if snote in SongDict:
            SongDict[snote] = SongDict[snote]+1

    print(SongDict)

    # Sort notes and frequencies
    Notes     = list(SongDict.keys())
    Frequency = [SongDict[note] for note in Notes]

    # creating the bar plot
    fig, ax = plt.subplots(figsize = (10, 5))
    bars = ax.bar(Notes, Frequency)

    for i in range(len(Notes)):
        ax.text(bars[i].get_x() + bars[i].get_width() / 2, bars[i].get_height(), str(Frequency[i]), ha='
            center', va='bottom')

    plt.xlabel("Specific-Note-Type-Used")
    plt.ylabel("Frequency-in-Song")
    plt.title("Frequency-of-Specific-Note-Types-in-Twinkle-Twinkle-Little-Star")
    plt.xticks(rotation = 45)
    plt.show()
    fig.savefig("Original-TTLS-Specific-Note-Type-Frequencies.png", bbox_inches='tight')

PlotSpecificNoteTypeFrequencies(formatted_notes)


###### Length of all notes:

def PlotNoteLengthFrequencies(LengthList):
    # Order lenths
    length_order = ['0.25', '0.5']

    # Creating an empty dictionary
    SongDict = {}

    # Set assign the rhythms as dictionary keys:
    for length in length_order:
        SongDict[length] = 0

    # Loop over the given rhythm list again to give a value to the number of occurrences.
    for length in LengthList:
        slength = str(length)
        if slength in SongDict:
            SongDict[slength] = SongDict[slength]+1

    print(SongDict)

    # Sort notes and frequencies
    Notes     = list(SongDict.keys())
    Frequency = [SongDict[note] for note in Notes]

    # creating the bar plot
    fig, ax = plt.subplots(figsize = (10, 5))
    bars = ax.bar(Notes, Frequency)

    for i in range(len(Notes)):
        ax.text(bars[i].get_x() + bars[i].get_width() / 2, bars[i].get_height(), str(Frequency[i]), ha='
            center', va='bottom')

    plt.xlabel("Length-of-Note")
    plt.ylabel("Frequency-in-Song")
```

```
    plt.title("Frequency of Note Lengths in Twinkle Twinkle Little Star")
    plt.xticks(rotation = 45)
    plt.show()
    fig.savefig("Original TTLS Note Length Frequencies.png", bbox_inches='tight')

PlotNoteLengthFrequencies(TTLS.interval)
```

Listing 6.3. Generating Random Twinkle Twinkle Little Star

```
from musicpy import *
import random

#### Generate the Random Song

TwinkleTransDictionary = {"C4 1/4": ["C4 1/4", "G4 1/4"],
                          "C4 1/2": ["G4 1/4"],
                          "D4 1/4": ["C4 1/2", "D4 1/4"],
                          "D4 1/2": ["C4 1/4", "G4 1/4"],
                          "E4 1/4": ["D4 1/4", "D4 1/2", "E4 1/4", "E4 1/4"],
                          "F4 1/4": ["E4 1/4", "F4 1/4"],
                          "G4 1/4": ["F4 1/4", "G4 1/4", "G4 1/4", "A4 1/4"],
                          "G4 1/2": ["F4 1/4"],
                          "A4 1/4": ["G4 1/2", "A4 1/4"]
                         }


states = []
intstate = "D4 1/2"
states.append(intstate)

for i in range(41):
    intstate = random.choice(TwinkleTransDictionary[intstate])
    states.append(intstate)

print(states)

notesString = ''
durationList = []
for item in states:
    note, duration = item.split()
    notesString += note + ","
    top,bottom = duration.split('/')
    durationList.append(int(top)/int(bottom))
notesString = notesString[:-1]

print(notesString)
print(durationList)

TTLSR = chord(notesString,
              interval=durationList)

play(TTLSR,bpm=117)
```

Listing 6.4. Analyzing Random "Twinkle Twinkle Little Star"

```
from musicpy import *
import random
import matplotlib.pyplot as plt

#### Analysis of Random Song
###### Only the notes:

def PlotNoteFrequencies(NoteList):
    # Order notes
    note_order = ['A4', 'C4', 'D4', 'E4', 'F4', 'G4']

    # Creating an empty dictionary
    SongDict = {}

    # Set assign the notes as dictionary keys:
    for note in note_order:
        SongDict[note] = 0
```

```
    # Loop over the given note list again to give a value to the number of occurrences.
    for note in NoteList:
        snote = str(note)
        if snote in SongDict:
            SongDict[snote] = SongDict[snote]+1

    print(SongDict)

    # Sort notes and frequencies
    Notes     = list(SongDict.keys())
    Frequency = [SongDict[note] for note in Notes]

    # creating the bar plot
    fig, ax = plt.subplots(figsize = (10, 5))
    bars = ax.bar(Notes, Frequency, color="purple")

    for i in range(len(Notes)):
        ax.text(bars[i].get_x() + bars[i].get_width() / 2, bars[i].get_height(), str(Frequency[i]), ha='
            center', va='bottom')

    plt.xlabel("Note Used")
    plt.ylabel("Frequency in Song")
    plt.title("Frequency of Notes in Newly Generated Twinkle Twinkle Little Star")
    plt.xticks(rotation = 45)
    plt.show()
    fig.savefig("Random TTLS Note Frequencies.png", bbox_inches='tight')

PlotNoteFrequencies(TTLSR.notes)


###### Notes and their respective length:

def PlotSpecificNoteTypeFrequencies(notesString, durationList):
    NoteList = [f"{note}-{duration}" for note, duration in zip(notesString.split(','), durationList)]

    SeenSpecificNoteList = []
    for note in NoteList:
        if str(note) not in SeenSpecificNoteList:
            SeenSpecificNoteList.append(str(note))

#     print("Seen Spcific Note Values = ", SeenSpecificNoteList)

    # Creating an empty dictionary
    SongDict = {}

    # Set assign the notes as dictionary keys:
    for note in SeenSpecificNoteList:
        SongDict[note] = 0

    # Loop over the given note list again to give a value to the number of occurrences.
    for note in NoteList:
        snote = str(note)
        SongDict[snote] = SongDict[snote] + 1

    print(SongDict)

    Notes = sorted(list(SongDict.keys()))
    Frequency = [SongDict[note] for note in Notes]

    # creating the bar plot
    fig, ax = plt.subplots(figsize = (10, 5))
    bars = ax.bar(Notes, Frequency, color='purple')

    for i in range(len(Notes)):
        ax.text(bars[i].get_x() + bars[i].get_width() / 2, bars[i].get_height(), str(Frequency[i]), ha='
            center', va='bottom')

    plt.xlabel("Specific Note Type Used")
    plt.ylabel("Frequency in Song")
    plt.title("Frequency of Specific Note Types in Newly Generated Twinkle Twinkle Little Star")
    plt.xticks(rotation = 45)
    plt.show()
    fig.savefig("Random TTLS Specific Note Type Frequencies.png", bbox_inches='tight')

PlotSpecificNoteTypeFrequencies(notesString, durationList)
```

```
###### Length of all notes:

def PlotNoteLengthFrequencies(LengthList):
    # Order lenths
    length_order = ['0.25', '0.5']

    # Creating an empty dictionary
    SongDict = {}

    # Set assign the rhythms as dictionary keys:
    for length in length_order:
        SongDict[length] = 0

    # Loop over the given rhythm list again to give a value to the number of occurrences.
    for length in LengthList:
        slength = str(length)
        if slength in SongDict:
            SongDict[slength] = SongDict[slength]+1

    print(SongDict)

    # Sort notes and frequencies
    Notes     = list(SongDict.keys())
    Frequency = [SongDict[note] for note in Notes]

    # creating the bar plot
    fig, ax = plt.subplots(figsize = (10, 5))
    bars = ax.bar(Notes, Frequency, color='purple')

    for i in range(len(Notes)):
        ax.text(bars[i].get_x() + bars[i].get_width() / 2, bars[i].get_height(), str(Frequency[i]), ha='
            center', va='bottom')

    plt.xlabel("Length of Note")
    plt.ylabel("Frequency in Song")
    plt.title("Frequency of Note Lengths in Newly Generated Twinkle Twinkle Little Star")
    plt.xticks(rotation = 45)
    plt.show()
    fig.savefig("Random TTLS Length of Notes Frequencies.png", bbox_inches='tight')

PlotNoteLengthFrequencies(TTLS.interval)
```

Listing 6.5. "You Belong With Me"

```
from musicpy import *

#Define the Rests and Notes.
rest1o8  = rest(1/8)
rest1o4  = rest(1/4)
rest1o2  = rest(1/2)
rest4o4  = rest(4/4)
FSh31o8  = chord('F#3', interval = [1/8])
FSh31o4  = chord('F#3', interval = [1/4])
FSh33o8  = chord('F#3', interval = [3/8])
GSh31o8  = chord('G#3', interval = [1/8])
GSh31o4  = chord('G#3', interval = [1/4])
GSh31o2  = chord('G#3', interval = [1/2])
ASh31o8  = chord('A#3', interval = [1/8])
ASh31o4  = chord('A#3', interval = [1/4])
ASh31o2  = chord('A#3', interval = [1/2])
B31o8    = chord('B3' , interval = [1/8])
B31o4    = chord('B3' , interval = [1/4])
CSh41o8  = chord('C#4', interval = [1/8])
CSh41o4  = chord('C#4', interval = [1/4])
CSh43o8  = chord('C#4', interval = [3/8])
DSh41o8  = chord('D#4', interval = [1/8])
DSh41o4  = chord('D#4', interval = [1/4])
FSh41o8  = chord('F#4', interval = [1/8])
FSh41o4  = chord('F#4', interval = [1/4])
GSh41o8  = chord('G#4', interval = [1/8])
GSh41o4  = chord('G#4', interval = [1/4])
GSh41o2  = chord('G#4', interval = [1/2])
```

```
ASh41o16 = chord('A#4', interval = [1/16])
ASh41o8  = chord('A#4', interval = [1/8])
ASh41o4  = chord('A#4', interval = [1/4])
ASh43o8  = chord('A#4', interval = [3/8])
B41o16   = chord('B4',  interval = [1/16])
B41o8    = chord('B4',  interval = [1/8])
B41o4    = chord('B4',  interval = [1/4])
B43o8    = chord('B4',  interval = [3/8])
CSh51o8  = chord('C#5', interval = [1/8])

YBWM = CSh41o4+ASh31o8+ASh31o8+GSh31o4+FSh31o8+FSh31o8+CSh41o4+ASh31o4+GSh31o8+ASh31o4
+GSh31o8+GSh31o8+GSh31o8+CSh41o8+CSh41o8+ASh31o8+ASh31o8+GSh31o4+CSh41o8+ASh31o4+GSh31o8
+GSh31o8+ASh31o4+GSh31o8+GSh31o8+GSh31o8+B31o4+ASh31o8+FSh33o8+GSh31o8+ASh31o4+ASh31o8
+ASh31o8+GSh31o8+FSh31o4+GSh31o2+ASh31o2+rest4o4+CSh41o4+ASh31o8+ASh31o8+GSh31o4+FSh31o8
+FSh31o8+CSh41o8+CSh41o8+ASh31o4+GSh31o8+ASh31o4+GSh31o8+GSh31o8+GSh31o8+CSh41o8+CSh41o8
+ASh31o8+ASh31o8+GSh31o8+GSh31o8+CSh41o8+CSh41o8+ASh31o4+GSh31o8+ASh31o4+GSh31o8+GSh31o8
+GSh31o8+B31o4+ASh31o8+FSh33o8+GSh31o8+ASh31o4+ASh31o8+ASh31o8+GSh31o8+FSh31o4+GSh31o2
+ASh31o2+rest1o2+rest1o4+rest1o8+CSh41o8+DSh41o4+FSh41o4+GSh41o4+FSh41o4+DSh41o4+FSh41o4
+GSh41o4+FSh41o4+CSh41o4+FSh41o4+FSh41o8+GSh41o8+FSh41o4+GSh41o4+GSh41o8+ASh41o8+GSh41o4
+FSh41o4+DSh41o8+FSh41o8+FSh41o8+DSh41o8+GSh41o4+CSh41o8+CSh41o8+DSh41o8+FSh41o4+DSh41o8
+FSh41o8+FSh41o8+DSh41o8+FSh41o8+CSh41o8+rest1o8+ASh41o8+GSh41o8+GSh41o8+FSh41o8
+FSh41o8+ASh41o8+ASh41o8+GSh41o8+FSh41o4+GSh41o8+FSh41o8+FSh41o8+GSh41o8+ASh41o4+GSh41o8
+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8+GSh41o8+ASh41o8+ASh41o8+GSh41o4+FSh41o4+ASh41o4
+GSh41o8+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8+GSh41o8+ASh41o4+GSh41o4+FSh41o4+B43o8
+ASh41o8+ASh41o4+DSh41o4+DSh41o4+FSh41o8+FSh41o8+FSh41o8+DSh41o8+CSh41o4+B43o8+ASh41o8
+ASh41o4+B31o4+B31o4+B31o8+B31o8+B31o8+ASh31o8+ASh31o8+ASh31o4+rest1o4+rest1o2
+rest4o4+CSh41o4+ASh31o8+ASh31o8+GSh31o4+FSh31o8+CSh41o8+CSh41o8+ASh31o4+GSh31o8+ASh31o4
+GSh31o8+GSh31o8+GSh31o8+CSh41o8+CSh41o8+ASh31o8+ASh31o8+GSh31o8+GSh31o8+CSh41o8+ASh31o4
+GSh31o8+GSh31o8+ASh31o4+GSh31o8+GSh31o4+B31o8+ASh31o8+B31o8+ASh31o8+B31o4+B31o4+B31o8
+CSh41o8+DSh41o8+FSh41o4+GSh41o8+FSh41o4+rest1o8+FSh41o8+FSh41o8+GSh41o8+FSh41o4+B41o4
+ASh41o8+GSh41o8+GSh41o8+FSh41o4+FSh41o8+FSh41o4+CSh41o8+CSh41o8+ASh31o8+GSh31o8+FSh31o8
+CSh41o8+ASh31o4+GSh31o8+GSh31o8+ASh31o4+GSh31o8+GSh31o8+GSh31o8+CSh41o8+CSh41o8+ASh31o8
+ASh31o8+GSh31o8+GSh31o8+CSh41o4+ASh31o8+ASh31o8+GSh31o8+ASh31o4+GSh31o8+GSh31o8+GSh31o8
+GSh31o8+GSh31o8+GSh31o8+ASh31o8+B31o8+CSh41o8+DSh41o8+DSh41o8+ASh41o4+GSh41o2+rest1o8
+GSh31o8+GSh31o8+GSh31o8+GSh31o8+ASh31o8+B31o8+CSh41o8+DSh41o4+ASh41o4+GSh41o4+rest1o4
+DSh41o4+FSh41o4+GSh41o4+FSh41o4+DSh41o4+FSh41o4+GSh41o4+FSh41o4+CSh41o4+FSh41o4+FSh41o8
+GSh41o8+FSh41o4+GSh41o4+GSh41o8+ASh41o8+GSh41o4+FSh41o4+DSh41o8+FSh41o8+FSh41o8+DSh41o8
+GSh41o4+CSh41o8+CSh41o8+DSh41o8+FSh41o8+DSh41o8+FSh41o8+FSh41o8+DSh41o8+FSh41o8+CSh41o8
+rest1o8+ASh41o8+GSh41o8+GSh41o8+GSh41o8+FSh41o8+FSh41o8+ASh41o8+ASh41o8+GSh41o8+FSh41o4
+GSh41o8+FSh41o8+FSh41o8+GSh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8
+GSh41o8+ASh41o8+ASh41o8+GSh41o4+FSh41o4+ASh41o4+GSh41o8+ASh41o8+ASh41o4+GSh41o8+ASh41o8
+ASh41o8+GSh41o8+ASh41o4+GSh41o4+FSh41o4+B43o8+ASh41o8+ASh41o4+DSh41o4+DSh41o4+FSh41o8
+FSh41o8+FSh41o8+DSh41o8+CSh41o4+B43o8+ASh41o8+ASh41o4+B31o4+B31o8+CSh43o8+rest1o2+ASh41o4
+GSh41o8+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8+GSh41o8+ASh41o8+GSh41o8+GSh41o4+FSh41o4
+ASh41o4+GSh41o8+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8+GSh41o8+ASh41o4+GSh41o4+FSh41o4
+B43o8+ASh41o8+ASh41o4+DSh41o4+DSh41o4+FSh41o8+FSh41o8+FSh41o8+DSh41o8+CSh41o4+B43o8
+ASh41o8+ASh41o4+B31o4+B41o4+B41o8+B41o8+B41o8+ASh41o8+FSh41o4+GSh41o8+ASh41o8+GSh41o8
+FSh41o8+DSh41o4+FSh41o8+FSh41o8+GSh41o8+ASh41o8+GSh41o8+FSh41o8+DSh41o4+FSh41o8+FSh41o8
+GSh41o8+ASh41o8+GSh41o8+FSh41o8+FSh41o4+FSh41o8+FSh41o8+GSh41o8+ASh41o8+GSh41o8+FSh41o8
+GSh41o4+rest1o8+FSh41o8+GSh41o8+ASh41o8+GSh41o8+FSh41o8+DSh41o4+DSh41o4+FSh41o8+GSh41o8
+ASh41o8+GSh41o8+FSh41o8+DSh41o4+FSh41o8+FSh41o8+GSh41o8+ASh41o8+GSh41o8+FSh41o8+FSh41o4
+FSh41o8+FSh41o8+GSh41o8+FSh41o8+GSh41o8+FSh41o8+ASh43o8+B41o16+ASh41o16+GSh41o4+rest1o4
+rest1o4+CSh41o8+CSh41o8+FSh41o4+DSh41o8+FSh41o8+FSh41o4+DSh41o8+FSh41o8+FSh41o8+FSh41o8
+FSh41o8+FSh41o8+GSh41o4+ASh41o4+GSh41o4+FSh41o8+GSh41o8+GSh41o4+FSh41o8+GSh41o8+GSh41o8
+FSh41o8+GSh41o4+FSh41o4+FSh41o4+B43o8+ASh41o8+ASh41o4+DSh41o4+DSh41o4+FSh41o8+FSh41o8
+FSh41o8+DSh41o8+CSh41o4+B43o8+ASh41o8+ASh41o4+B31o4+B31o8+CSh43o8+rest1o2+ASh41o4+GSh41o8
+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8+GSh41o8+ASh41o8+GSh41o8+GSh41o4+FSh41o4+ASh41o4
+GSh41o8+ASh41o8+ASh41o4+GSh41o8+CSh51o8+CSh51o8+ASh41o8+ASh41o4+GSh41o4+FSh41o4+B43o8
+ASh41o8+ASh41o4+DSh41o4+DSh41o4+FSh41o8+FSh41o8+FSh41o8+DSh41o8+CSh41o4+B43o8+ASh41o8
+ASh41o4+B31o4+B31o4+B31o8+B31o8+B31o8+ASh31o8+ASh31o8+ASh31o8+ASh31o4+rest1o4+rest1o2
+rest1o4+ASh41o8+ASh41o8+ASh41o8+GSh41o8+FSh41o8+GSh41o8+GSh41o8+ASh41o8+rest1o4+rest1o4
+ASh41o8+GSh41o8+ASh41o8+GSh41o8+ASh41o4+GSh41o4+FSh41o4+B43o8+ASh41o8+ASh41o4+DSh41o4
+DSh41o4+FSh41o8+FSh41o8+FSh41o8+DSh41o8+CSh41o4+B43o8+ASh41o8+ASh41o4+B31o4+B31o4+B31o8
+B31o8+B31o8+ASh31o8+ASh31o8+ASh31o8+ASh31o4

play(YBWM, bpm = 130, instrument = 25)
```

Listing 6.6. Analyzing Original "You Belong With Me"

```
from musicpy import *
import matplotlib.pyplot as plt

#Define the Rests and Notes.
```

```
rest1o8  = rest(1/8)
rest1o4  = rest(1/4)
rest1o2  = rest(1/2)
rest4o4  = rest(4/4)
FSh31o8  = chord('F#3', interval = [1/8])
FSh31o4  = chord('F#3', interval = [1/4])
FSh33o8  = chord('F#3', interval = [3/8])
GSh31o8  = chord('G#3', interval = [1/8])
GSh31o4  = chord('G#3', interval = [1/4])
GSh31o2  = chord('G#3', interval = [1/2])
ASh31o8  = chord('A#3', interval = [1/8])
ASh31o4  = chord('A#3', interval = [1/4])
ASh31o2  = chord('A#3', interval = [1/2])
B31o8    = chord('B3' , interval = [1/8])
B31o4    = chord('B3' , interval = [1/4])
CSh41o8  = chord('C#4', interval = [1/8])
CSh41o4  = chord('C#4', interval = [1/4])
CSh43o8  = chord('C#4', interval = [3/8])
DSh41o8  = chord('D#4', interval = [1/8])
DSh41o4  = chord('D#4', interval = [1/4])
FSh41o8  = chord('F#4', interval = [1/8])
FSh41o4  = chord('F#4', interval = [1/4])
GSh41o8  = chord('G#4', interval = [1/8])
GSh41o4  = chord('G#4', interval = [1/4])
GSh41o2  = chord('G#4', interval = [1/2])
ASh41o16 = chord('A#4', interval = [1/16])
ASh41o8  = chord('A#4', interval = [1/8])
ASh41o4  = chord('A#4', interval = [1/4])
ASh43o8  = chord('A#4', interval = [3/8])
B41o16   = chord('B4',  interval = [1/16])
B41o8    = chord('B4',  interval = [1/8])
B41o4    = chord('B4',  interval = [1/4])
B43o8    = chord('B4',  interval = [3/8])
CSh51o8  = chord('C#5', interval = [1/8])

YBWM = CSh41o4+ASh31o8+ASh31o8+GSh31o4+FSh31o8+FSh31o8+CSh41o4+ASh31o4+GSh31o8+ASh31o4
+GSh31o8+GSh31o8+GSh31o8+CSh41o8+CSh41o8+ASh31o8+ASh31o8+GSh31o4+CSh41o8+ASh31o4+GSh31o8
+GSh31o8+ASh31o4+GSh31o8+GSh31o8+GSh31o8+B31o4+ASh31o8+FSh33o8+GSh31o8+ASh31o4+ASh31o8
+ASh31o8+GSh31o8+FSh31o4+GSh31o2+ASh31o2+rest4o4+CSh41o4+ASh31o8+ASh31o8+GSh31o4+FSh31o8
+FSh31o8+CSh41o8+CSh41o8+ASh31o4+GSh31o8+ASh31o4+GSh31o8+GSh31o8+GSh31o8+CSh41o8+CSh41o8
+ASh31o8+ASh31o8+GSh31o8+GSh31o8+CSh41o8+CSh41o8+ASh31o4+GSh31o8+ASh31o4+GSh31o8+GSh31o8
+GSh31o8+B31o4+ASh31o8+FSh33o8+GSh31o8+ASh31o4+ASh31o8+ASh31o8+GSh31o8+FSh31o4+GSh31o2
+ASh31o2+rest1o2+rest1o4+rest1o8+CSh41o8+DSh41o4+FSh41o4+GSh41o4+FSh41o4+DSh41o4+FSh41o4
+GSh41o4+FSh41o4+CSh41o4+FSh41o4+FSh41o8+GSh41o8+FSh41o4+GSh41o4+GSh41o8+ASh41o8+GSh41o4
+FSh41o4+DSh41o8+FSh41o8+FSh41o8+DSh41o8+GSh41o4+CSh41o8+CSh41o8+DSh41o8+FSh41o4+DSh41o8
+FSh41o8+FSh41o8+DSh41o8+FSh41o8+CSh41o8+rest1o8+ASh41o8+GSh41o8+GSh41o8+GSh41o8+FSh41o8
+FSh41o8+ASh41o8+ASh41o8+GSh41o8+FSh41o4+GSh41o8+FSh41o8+GSh41o8+ASh41o4+GSh41o8
+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8+GSh41o8+ASh41o8+ASh41o8+GSh41o4+FSh41o4+ASh41o4
+GSh41o8+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8+GSh41o8+ASh41o4+GSh41o4+FSh41o4+B43o8
+ASh41o8+ASh41o4+DSh41o4+DSh41o4+FSh41o8+FSh41o8+FSh41o8+DSh41o8+CSh41o4+B43o8+ASh41o8
+ASh41o4+B31o4+B31o4+B31o8+B31o8+B31o8+ASh31o8+ASh31o8+ASh31o8+ASh31o4+rest1o4+rest1o2
+rest4o4+CSh41o4+ASh31o8+ASh31o8+GSh31o4+FSh31o8+CSh41o8+CSh41o8+ASh31o4+GSh31o8+ASh31o4
+GSh31o8+GSh31o8+GSh31o8+CSh41o8+CSh41o8+ASh31o8+ASh31o8+GSh31o8+GSh31o8+CSh41o8+ASh31o4
+GSh31o8+GSh31o8+ASh31o4+GSh31o8+GSh31o4+B31o8+ASh31o8+B31o8+ASh31o8+B31o4+B31o4+B31o8
+CSh41o8+DSh41o8+FSh41o4+GSh41o8+FSh41o4+rest1o8+FSh41o8+FSh41o8+GSh41o8+FSh41o4+B41o4
+ASh41o8+GSh41o8+GSh41o8+FSh41o4+FSh41o8+FSh41o4+CSh41o8+CSh41o8+ASh31o8+GSh31o8+FSh31o8
+CSh41o8+ASh31o4+GSh31o8+GSh31o8+ASh31o4+GSh31o8+GSh31o8+GSh31o8+CSh41o8+CSh41o8+ASh31o8
+ASh31o8+GSh31o8+GSh31o8+CSh41o4+ASh31o8+ASh31o8+GSh31o8+GSh31o8+GSh31o8+GSh31o8+GSh31o8
+GSh31o8+GSh31o8+GSh31o8+ASh31o8+B31o8+CSh41o8+DSh41o8+DSh41o8+ASh41o4+GSh41o2+rest1o8
+GSh31o8+GSh31o8+GSh31o8+GSh31o8+ASh31o8+B31o8+CSh41o8+DSh41o4+ASh41o4+GSh41o4+rest1o4
+DSh41o4+FSh41o4+GSh41o4+FSh41o4+DSh41o4+FSh41o4+GSh41o4+FSh41o4+CSh41o4+FSh41o4+FSh41o8
+GSh41o8+FSh41o4+GSh41o4+GSh41o8+ASh41o8+GSh41o4+FSh41o4+DSh41o8+FSh41o8+FSh41o8+DSh41o8
+GSh41o4+CSh41o8+CSh41o8+DSh41o8+FSh41o8+DSh41o8+FSh41o8+CSh41o8
+rest1o8+ASh41o8+GSh41o8+GSh41o8+GSh41o8+FSh41o8+FSh41o8+ASh41o8+GSh41o8+FSh41o4
+GSh41o8+FSh41o8+FSh41o8+GSh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8
+GSh41o8+ASh41o8+ASh41o8+GSh41o4+FSh41o4+ASh41o4+GSh41o8+ASh41o8+ASh41o4+GSh41o8+ASh41o8
+ASh41o8+GSh41o8+ASh41o4+GSh41o4+FSh41o4+B43o8+ASh41o8+ASh41o4+DSh41o4+DSh41o4+FSh41o8
+FSh41o8+FSh41o8+DSh41o8+CSh41o4+B43o8+ASh41o8+ASh41o4+B31o4+B31o8+CSh43o8+rest1o2+ASh41o4
+GSh41o8+ASh41o8+ASh41o4+GSh41o8+ASh41o8+GSh41o8+ASh41o8+GSh41o8+GSh41o4+FSh41o4
+ASh41o4+GSh41o8+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8+GSh41o8+ASh41o4+GSh41o4+FSh41o4
+B43o8+ASh41o8+ASh41o4+DSh41o4+DSh41o4+FSh41o8+FSh41o8+FSh41o8+DSh41o8+CSh41o4+B43o8
+ASh41o8+ASh41o4+B31o4+B41o4+B41o8+B41o8+B41o8+ASh41o8+FSh41o4+GSh41o8+ASh41o8+GSh41o8
+FSh41o8+DSh41o4+FSh41o8+FSh41o8+GSh41o8+ASh41o8+GSh41o8+FSh41o8+DSh41o4+FSh41o8+FSh41o8
+GSh41o8+ASh41o8+GSh41o8+FSh41o4+FSh41o8+GSh41o8+ASh41o8+GSh41o8+FSh41o8
+GSh41o4+rest1o8+FSh41o8+GSh41o8+ASh41o8+GSh41o8+FSh41o8+DSh41o4+DSh41o4+FSh41o8+GSh41o8
+ASh41o8+GSh41o8+FSh41o8+DSh41o4+FSh41o8+FSh41o8+GSh41o8+ASh41o8+GSh41o8+FSh41o4
```

```
+FSh41o8+FSh41o8+GSh41o8+FSh41o8+GSh41o8+FSh41o8+ASh43o8+B41o16+ASh41o16+GSh41o4+rest1o4
+rest1o4+CSh41o8+CSh41o8+FSh41o4+DSh41o8+FSh41o8+FSh41o4+DSh41o8+FSh41o8+FSh41o8+FSh41o8
+FSh41o8+FSh41o8+GSh41o4+ASh41o4+GSh41o4+FSh41o8+GSh41o8+GSh41o4+FSh41o8+GSh41o8+GSh41o8
+FSh41o8+GSh41o4+FSh41o4+FSh41o4+B43o8+ASh41o8+ASh41o4+DSh41o4+DSh41o4+FSh41o8+FSh41o8
+FSh41o8+DSh41o8+CSh41o4+B43o8+ASh41o8+ASh41o4+B31o4+B31o8+CSh43o8+rest1o2+ASh41o4+GSh41o8
+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8+GSh41o8+ASh41o8+GSh41o8+GSh41o4+FSh41o4+ASh41o4
+GSh41o8+ASh41o8+ASh41o4+GSh41o8+CSh51o8+CSh51o8+ASh41o8+ASh41o4+GSh41o4+FSh41o4+B43o8
+ASh41o8+ASh41o4+DSh41o4+DSh41o4+FSh41o8+FSh41o8+FSh41o8+DSh41o8+CSh41o4+B43o8+ASh41o8
+ASh41o4+B31o4+B31o4+B31o8+B31o8+B31o8+ASh31o8+ASh31o8+ASh31o8+ASh31o4+rest1o4+rest1o2
+rest1o4+ASh41o8+ASh41o8+ASh41o8+GSh41o8+FSh41o8+GSh41o8+GSh41o8+ASh41o8+rest1o4+rest1o4
+ASh41o8+GSh41o8+ASh41o8+GSh41o8+ASh41o4+GSh41o4+FSh41o4+B43o8+ASh41o8+ASh41o4+DSh41o4
+DSh41o4+FSh41o8+FSh41o8+FSh41o8+DSh41o8+CSh41o4+B43o8+ASh41o8+ASh41o4+B31o4+B31o4+B31o8
+B31o8+B31o8+ASh31o8+ASh31o8+ASh31o8+ASh31o4

play(YBWM, bpm = 130, instrument = 25)

#print(YBWM)
#print(YBWM.notes)
#print(YBWM.interval)
#YBWM.analyze_rhythm()

# Given string
songstr = 'CSh41o4+ASh31o8+ASh31o8+GSh31o4+FSh31o8+FSh31o8+CSh41o4+ASh31o4+GSh31o8+ASh31o4
+GSh31o8+GSh31o8+GSh31o8+CSh41o8+CSh41o8+ASh31o8+ASh31o8+GSh31o4+CSh41o8+ASh31o4+GSh31o8
+GSh31o8+ASh31o4+GSh31o8+GSh31o8+GSh31o8+B31o4+ASh31o8+FSh33o8+GSh31o8+ASh31o4+ASh31o8
+ASh31o8+GSh31o8+FSh31o4+GSh31o2+ASh31o2+rest4o4+CSh41o4+ASh31o8+ASh31o8+GSh31o4+FSh31o8
+FSh31o8+CSh41o8+CSh41o8+ASh31o4+GSh31o8+ASh31o4+GSh31o8+GSh31o8+GSh31o8+CSh41o8+CSh41o8
+ASh31o8+ASh31o8+GSh31o8+GSh31o8+CSh41o8+CSh41o8+ASh31o4+GSh31o8+ASh31o4+GSh31o8+GSh31o8
+GSh31o8+B31o4+ASh31o8+FSh33o8+GSh31o8+ASh31o4+ASh31o8+ASh31o8+GSh31o8+FSh31o4+GSh31o2
+ASh31o2+rest1o2+rest1o4+rest1o8+CSh41o8+DSh41o4+FSh41o4+GSh41o4+FSh41o4+DSh41o4+FSh41o4
+GSh41o4+FSh41o4+CSh41o4+FSh41o8+GSh41o8+FSh41o4+GSh41o4+GSh41o8+ASh41o8+GSh41o4
+FSh41o4+DSh41o8+FSh41o8+FSh41o8+DSh41o8+GSh41o4+CSh41o8+CSh41o8+DSh41o8+FSh41o4+DSh41o8
+FSh41o8+FSh41o8+DSh41o8+FSh41o8+CSh41o8+rest1o8+ASh41o8+GSh41o8+GSh41o8+GSh41o8+FSh41o8
+FSh41o8+ASh41o8+ASh41o8+GSh41o8+FSh41o4+GSh41o8+FSh41o8+FSh41o8+GSh41o8+ASh41o4+GSh41o8
+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8+GSh41o8+ASh41o8+ASh41o8+GSh41o4+FSh41o4+ASh41o4
+GSh41o8+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8+GSh41o8+ASh41o4+GSh41o4+FSh41o4+B43o8
+ASh41o8+ASh41o4+DSh41o4+FSh41o8+FSh41o8+DSh41o8+CSh41o4+B43o8+ASh41o8
+ASh41o4+B31o4+B31o4+B31o8+B31o8+B31o8+ASh31o8+ASh31o8+ASh31o8+ASh31o4+rest1o4+rest1o2
+rest4o4+CSh41o4+ASh31o8+ASh31o8+GSh31o4+FSh31o8+CSh41o8+CSh41o8+ASh31o4+GSh31o8+ASh31o4
+GSh31o8+GSh31o8+GSh31o8+CSh41o8+CSh41o8+ASh31o8+ASh31o8+GSh31o8+GSh31o8+CSh41o8+ASh31o4
+GSh31o8+GSh31o8+ASh31o4+GSh31o8+GSh31o4+B31o8+ASh31o8+B31o8+ASh31o8+B31o4+B31o4+B31o8
+CSh41o8+DSh41o8+FSh41o4+GSh41o8+FSh41o4+rest1o8+FSh41o8+FSh41o8+GSh41o8+FSh41o4+B41o4
+ASh41o8+GSh41o8+GSh41o8+FSh41o4+FSh41o8+FSh41o4+CSh41o8+CSh41o8+ASh31o8+GSh31o8+FSh31o8
+CSh41o8+ASh31o4+GSh31o8+GSh31o8+ASh31o4+GSh31o8+GSh31o8+GSh31o8+CSh41o8+CSh41o8+ASh31o8
+ASh31o8+GSh31o8+GSh31o8+CSh41o4+ASh31o8+ASh31o8+GSh31o8+ASh31o4+GSh31o8+GSh31o8+GSh31o8
+GSh31o8+GSh31o8+GSh31o8+ASh31o8+B31o8+CSh41o8+DSh41o8+DSh41o8+ASh41o4+GSh41o2+rest1o8
+GSh31o8+GSh31o8+GSh31o8+ASh31o8+B31o8+CSh41o8+DSh41o4+ASh41o4+GSh41o4+rest1o4
+DSh41o4+FSh41o4+GSh41o4+FSh41o4+DSh41o4+FSh41o4+GSh41o4+CSh41o4+FSh41o4+FSh41o8
+GSh41o8+FSh41o4+GSh41o4+GSh41o8+ASh41o8+GSh41o4+FSh41o4+DSh41o8+FSh41o8+FSh41o8+DSh41o8
+GSh41o4+CSh41o8+CSh41o8+DSh41o8+FSh41o8+DSh41o8+FSh41o8+DSh41o8+FSh41o8+CSh41o8
+rest1o8+ASh41o8+GSh41o8+GSh41o8+GSh41o8+FSh41o8+FSh41o8+ASh41o8+ASh41o8+GSh41o8+FSh41o4
+GSh41o8+FSh41o8+FSh41o8+GSh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8
+GSh41o8+ASh41o8+ASh41o8+GSh41o4+FSh41o4+ASh41o4+GSh41o8+ASh41o8+ASh41o4+GSh41o8+ASh41o8
+ASh41o8+GSh41o8+ASh41o4+GSh41o4+FSh41o4+B43o8+ASh41o8+ASh41o4+DSh41o4+DSh41o4+FSh41o8
+FSh41o8+FSh41o8+DSh41o8+CSh41o4+B43o8+ASh41o8+ASh41o4+B31o4+B31o8+CSh43o8+rest1o2+ASh41o4
+GSh41o8+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8+GSh41o8+ASh41o8+GSh41o8+GSh41o4+FSh41o4
+ASh41o4+GSh41o8+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8+GSh41o8+ASh41o4+GSh41o4+FSh41o4
+B43o8+ASh41o8+ASh41o4+DSh41o4+FSh41o8+FSh41o8+DSh41o8+CSh41o4+B43o8
+ASh41o8+ASh41o4+B31o4+B41o4+B41o8+B41o8+B41o8+ASh41o8+FSh41o4+GSh41o8+ASh41o8+GSh41o8
+FSh41o8+DSh41o4+FSh41o8+FSh41o8+GSh41o8+ASh41o8+GSh41o8+FSh41o8+DSh41o4+FSh41o8+FSh41o8
+GSh41o8+ASh41o8+GSh41o8+FSh41o8+FSh41o4+FSh41o8+FSh41o8+GSh41o8+FSh41o8+GSh41o8+FSh41o8
+GSh41o4+rest1o8+FSh41o8+GSh41o8+ASh41o8+GSh41o8+FSh41o8+DSh41o4+DSh41o4+FSh41o8+GSh41o8
+ASh41o8+GSh41o8+FSh41o8+DSh41o4+FSh41o8+GSh41o8+ASh41o8+GSh41o8+FSh41o8+FSh41o4
+FSh41o8+FSh41o8+GSh41o8+ASh43o8+B41o16+ASh41o16+GSh41o4+rest1o4
+rest1o4+CSh41o8+CSh41o8+FSh41o4+DSh41o8+FSh41o8+FSh41o4+DSh41o8+FSh41o8+FSh41o8+FSh41o8
+FSh41o8+FSh41o8+GSh41o4+ASh41o4+GSh41o4+FSh41o8+GSh41o8+GSh41o4+FSh41o8+GSh41o8+GSh41o8
+FSh41o8+GSh41o4+FSh41o4+FSh41o4+B43o8+ASh41o8+ASh41o4+DSh41o4+DSh41o4+FSh41o8+FSh41o8
+FSh41o8+DSh41o8+CSh41o4+B43o8+ASh41o8+ASh41o4+B31o4+B31o8+CSh43o8+rest1o2+ASh41o4+GSh41o8
+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8+GSh41o8+ASh41o8+GSh41o8+GSh41o4+FSh41o4+ASh41o4
+GSh41o8+ASh41o8+ASh41o4+GSh41o8+CSh51o8+CSh51o8+ASh41o8+ASh41o4+GSh41o4+FSh41o4+B43o8
+ASh41o8+ASh41o4+DSh41o4+DSh41o4+FSh41o8+FSh41o8+FSh41o8+DSh41o8+CSh41o4+B43o8+ASh41o8
+ASh41o4+B31o4+B31o4+B31o8+B31o8+B31o8+ASh31o8+ASh31o8+ASh31o8+ASh31o4+rest1o4+rest1o2
+rest1o4+ASh41o8+ASh41o8+ASh41o8+GSh41o8+FSh41o8+GSh41o8+GSh41o8+ASh41o8+rest1o4+rest1o4
+ASh41o8+GSh41o8+ASh41o8+GSh41o8+ASh41o4+GSh41o4+FSh41o4+B43o8+ASh41o8+ASh41o4+DSh41o4
+DSh41o4+FSh41o8+FSh41o8+FSh41o8+DSh41o8+CSh41o4+B43o8+ASh41o8+ASh41o4+B31o4+B31o4+B31o8
+B31o8+B31o8+ASh31o8+ASh31o8+ASh31o8+ASh31o4'
```

```
songList = songstr.split("+")

# Filter out the 'rests'
filtered_items = [item for item in songList if 'rest' not in item]

# Joining the filtered items back into a string
result = '+'.join(filtered_items)
print(result)
```

```
YBWMnr = CSh41o4+ASh31o8+ASh31o8+GSh31o4+FSh31o8+FSh31o8+CSh41o4+ASh31o4+GSh31o8+ASh31o4
+GSh31o8+GSh31o8+GSh31o8+CSh41o8+CSh41o8+ASh31o8+ASh31o8+GSh31o4+CSh41o8+ASh31o4+GSh31o8
+GSh31o8+ASh31o4+GSh31o8+GSh31o8+GSh31o8+B31o4+ASh31o8+FSh33o8+GSh31o8+ASh31o4+ASh31o8
+ASh31o8+GSh31o8+FSh31o4+GSh31o2+ASh31o2+CSh41o4+ASh31o8+ASh31o8+GSh31o4+FSh31o8+FSh31o8
+CSh41o8+CSh41o8+ASh31o4+GSh31o8+ASh31o4+GSh31o8+GSh31o8+GSh31o8+CSh41o8+CSh41o8+ASh31o8
+ASh31o8+GSh31o8+GSh31o8+CSh41o8+CSh41o8+ASh31o4+GSh31o8+ASh31o4+GSh31o8+GSh31o8+GSh31o8
+B31o4+ASh31o8+FSh33o8+GSh31o8+ASh31o4+ASh31o8+ASh31o8+GSh31o8+FSh31o4+GSh31o2+ASh31o2
+CSh41o8+DSh41o4+FSh41o4+GSh41o4+FSh41o4+DSh41o4+FSh41o4+GSh41o4+FSh41o4+CSh41o4+FSh41o4
+FSh41o8+GSh41o8+FSh41o4+GSh41o4+GSh41o8+ASh41o8+GSh41o4+FSh41o4+DSh41o8+FSh41o8+FSh41o8
+DSh41o8+GSh41o4+CSh41o8+CSh41o8+DSh41o8+FSh41o4+DSh41o8+FSh41o8+FSh41o8+DSh41o8+FSh41o8
+CSh41o8+ASh41o8+GSh41o8+GSh41o8+FSh41o8+FSh41o8+ASh41o8+ASh41o8+GSh41o8+FSh41o4
+GSh41o8+FSh41o8+FSh41o8+GSh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8
+GSh41o8+ASh41o8+ASh41o8+GSh41o4+FSh41o4+ASh41o4+GSh41o8+ASh41o8+ASh41o4+GSh41o8+ASh41o8
+ASh41o8+GSh41o8+ASh41o4+GSh41o4+FSh41o4+B43o8+ASh41o8+ASh41o4+DSh41o4+DSh41o4+FSh41o8
+FSh41o8+FSh41o8+DSh41o8+CSh41o4+B43o8+ASh41o8+ASh41o4+B31o4+B31o4+B31o8+B31o8+B31o8
+ASh31o8+ASh31o8+ASh31o8+ASh31o4+CSh41o4+ASh31o8+ASh31o8+GSh31o4+FSh31o8+CSh41o8+CSh41o8
+ASh31o4+GSh31o8+ASh31o4+GSh31o8+GSh31o8+GSh31o8+CSh41o8+CSh41o8+ASh31o8+ASh31o8+GSh31o8
+GSh31o8+CSh41o8+ASh31o4+GSh31o8+GSh31o8+ASh31o4+GSh31o8+GSh31o4+B31o8+ASh31o8+B31o8
+ASh31o8+B31o4+B31o4+B31o8+CSh41o8+DSh41o8+FSh41o4+GSh41o8+FSh41o4+FSh41o8+FSh41o8+GSh41o8
+FSh41o4+B41o4+ASh41o8+GSh41o8+GSh41o8+FSh41o4+FSh41o8+FSh41o4+CSh41o8+CSh41o8+ASh31o8
+GSh31o8+FSh31o8+CSh41o8+ASh31o4+GSh31o8+GSh31o8+ASh31o4+GSh31o8+GSh31o8+GSh31o8+CSh41o8
+CSh41o8+ASh31o8+ASh31o8+GSh31o8+GSh31o8+CSh41o4+ASh31o8+ASh31o8+GSh31o8+ASh31o4+GSh31o8
+GSh31o8+GSh31o8+GSh31o8+GSh31o8+GSh31o8+ASh31o8+B31o8+CSh41o8+DSh41o8+DSh41o8+ASh41o4
+GSh41o2+GSh31o8+GSh31o8+GSh31o8+GSh31o8+ASh31o8+B31o8+CSh41o8+DSh41o4+ASh41o4+GSh41o4
+DSh41o4+FSh41o4+GSh41o4+FSh41o4+DSh41o4+FSh41o4+GSh41o4+FSh41o4+CSh41o4+FSh41o4+FSh41o8
+GSh41o8+FSh41o4+GSh41o4+GSh41o8+ASh41o8+GSh41o4+FSh41o4+DSh41o8+FSh41o8+FSh41o8+DSh41o8
+GSh41o4+CSh41o8+CSh41o8+DSh41o8+FSh41o8+DSh41o8+FSh41o8+DSh41o8+FSh41o8+CSh41o8
+ASh41o8+GSh41o8+GSh41o8+GSh41o8+FSh41o8+FSh41o8+ASh41o8+ASh41o8+GSh41o8+FSh41o4+GSh41o8
+FSh41o8+FSh41o8+GSh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8+GSh41o8
+ASh41o8+ASh41o8+GSh41o4+FSh41o4+ASh41o4+GSh41o8+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8
+GSh41o8+ASh41o4+GSh41o4+FSh41o4+B43o8+ASh41o8+ASh41o4+DSh41o4+DSh41o4+FSh41o8+FSh41o8
+FSh41o8
+DSh41o8+CSh41o4+B43o8+ASh41o8+ASh41o4+B31o4+B31o8+CSh43o8+ASh41o4+GSh41o8+ASh41o8+ASh41o4
+GSh41o8+ASh41o8+ASh41o8+GSh41o8+ASh41o8+GSh41o8+GSh41o4+FSh41o4+ASh41o4+GSh41o8+ASh41o8
+ASh41o4+GSh41o8+ASh41o8+ASh41o8+GSh41o8+ASh41o4+GSh41o4+FSh41o4+B43o8+ASh41o8+ASh41o4+DSh41o4
+DSh41o4+FSh41o8+FSh41o8+FSh41o8+DSh41o8+CSh41o4+B43o8+ASh41o8+ASh41o4+B31o4+B41o4+B41o8
+B41o8+B41o8+ASh41o8+FSh41o4+GSh41o8+ASh41o8+GSh41o8+FSh41o8+DSh41o4+FSh41o8+FSh41o8
+GSh41o8+ASh41o8+GSh41o8+FSh41o8+DSh41o4+FSh41o8+FSh41o8+GSh41o8+ASh41o8+GSh41o8+FSh41o8
+FSh41o4+FSh41o8+FSh41o8+GSh41o8+ASh41o8+GSh41o8+FSh41o8+GSh41o4+FSh41o8+GSh41o8
+ASh41o8+GSh41o8+FSh41o8+DSh41o4+DSh41o4+FSh41o8+GSh41o8+ASh41o8+GSh41o8+FSh41o8+DSh41o4
+FSh41o8+FSh41o8+GSh41o8+ASh41o8+GSh41o8+FSh41o4+FSh41o8+FSh41o8+GSh41o8+FSh41o8
+GSh41o8+FSh41o8+ASh43o8+B41o16+ASh41o16+GSh41o4+CSh41o8+CSh41o8+FSh41o4+DSh41o8+FSh41o8
+FSh41o4+DSh41o8+FSh41o8+FSh41o8+FSh41o8+FSh41o8+GSh41o4+ASh41o4+GSh41o4+FSh41o8
+GSh41o8+GSh41o4+FSh41o8+GSh41o8+GSh41o8+FSh41o8+GSh41o4+FSh41o4+FSh41o4+B43o8+ASh41o8
+ASh41o4+DSh41o4+DSh41o4+FSh41o8+FSh41o8+FSh41o8+DSh41o8+CSh41o4+B43o8+ASh41o8+ASh41o4
+B31o4+B31o8+CSh43o8+ASh41o4+GSh41o8+ASh41o8+ASh41o4+GSh41o8+ASh41o8+ASh41o8+GSh41o8
+ASh41o8+GSh41o8+GSh41o4+FSh41o4+ASh41o4+GSh41o8+ASh41o8+ASh41o4+GSh41o8+CSh51o8+CSh51o8
+ASh41o8+ASh41o4+GSh41o4+FSh41o4+B43o8+ASh41o8+ASh41o4+DSh41o4+DSh41o4+FSh41o8+FSh41o8
+FSh41o8+DSh41o8+CSh41o4+B43o8+ASh41o8+ASh41o4+B31o4+B31o4+B31o8+B31o8+B31o8+ASh31o8
+ASh31o8+ASh31o8+ASh31o4+ASh41o8+ASh41o8+ASh41o8+GSh41o8+FSh41o8+GSh41o8+GSh41o8+ASh41o8
+ASh41o8+GSh41o8+ASh41o8+GSh41o8+ASh41o4+GSh41o4+FSh41o4+B43o8+ASh41o8+ASh41o4+DSh41o4
+DSh41o4+FSh41o8+FSh41o8+FSh41o8+DSh41o8+CSh41o4+B43o8+ASh41o8+ASh41o4+B31o4+B31o4+B31o8
+B31o8+B31o8+ASh31o8+ASh31o8+ASh31o8+ASh31o4
```

```
#### Analysis of Original Song
###### Only the notes:
```

```
def PlotNoteFrequencies(NoteList):

    # Order notes
    note_order = ['A#3', 'A#4', 'B3', 'B4', 'C#4', 'C#5', 'D#4', 'F#3', 'F#4', 'G#3', 'G#4']

    # Creating an empty dictionary
    SongDict = {}
```

```
    # Set assign the notes as dictionary keys:
    for note in note_order:
        SongDict[note] = 0

    # Loop over the given note list again to give a value to the number of occurrences.
    for note in NoteList:
        snote = str(note)
        if snote in SongDict:
            SongDict[snote] = SongDict[snote]+1

    print(SongDict)

    # Sort notes and frequencies
    Notes     = list(SongDict.keys())
    Frequency = [SongDict[note] for note in Notes]

    # creating the bar plot
    fig, ax = plt.subplots(figsize = (10, 5))
    bars = ax.bar(Notes, Frequency, color = "gold")

    for i in range(len(Notes)):
        ax.text(bars[i].get_x() + bars[i].get_width() / 2, bars[i].get_height(), str(Frequency[i]), ha='
            center', va='bottom')

    plt.xlabel("Note Used")
    plt.ylabel("Frequency in Song")
    plt.title("Frequency of Notes in You Belong With Me")
    plt.xticks(rotation = 45)
    plt.show()
    fig.savefig("Original YBWM Note Frequencies.png", bbox_inches='tight')

PlotNoteFrequencies(YBWM.notes)

###### Notes and their respective length:

formatted_notes = ['-'.join([str(note), str(interval)]) for note, interval in zip(YBWMnr, YBWMnr.interval)
    ]
print(formatted_notes)

def PlotSpecificNoteFrequencies(NoteList):

    # Order specific notes
    specific_note_order = ['A#3-0.125', 'A#3-0.25', 'A#3-0.5', 'A#4-0.0625', 'A#4-0.125', 'A#4-0.25', 'A#4
        -0.375', 'B3-0.125', 'B3-0.25', 'B4-0.0625', 'B4-0.125', 'B4-0.25', 'B4-0.375', 'C#4-0.125', 'C#4-
        0.25', 'C#4-0.375', 'C#5-0.125', 'D#4-0.125', 'D#4-0.25', 'F#3-0.125', 'F#3-0.25', 'F#3-0.375', 'F
        #4-0.125', 'F#4-0.25', 'G#3-0.125', 'G#3-0.25', 'G#3-0.5', 'G#4-0.125', 'G#4-0.25', 'G#4-0.5']

    # Creating an empty dictionary
    SongDict = {}

    # Set assign the notes as dictionary keys:
    for note in specific_note_order:
        SongDict[note] = 0

    # Loop over the given note list again to give a value to the number of occurrences.
    for note in NoteList:
        snote = str(note)
        if snote in SongDict:
            SongDict[snote] = SongDict[snote]+1

    print(SongDict)

    # Sort notes and frequencies
    Notes     = list(SongDict.keys())
    Frequency = [SongDict[note] for note in Notes]

    # creating the bar plot
    fig, ax = plt.subplots(figsize = (10, 5))
    bars = ax.bar(Notes, Frequency, color = "gold")

    for i in range(len(Notes)):
        ax.text(bars[i].get_x() + bars[i].get_width() / 2, bars[i].get_height(), str(Frequency[i]), ha='
            center', va='bottom')

    plt.xlabel("Specific Note Type Used")
    plt.ylabel("Frequency in Song")
```

```
    plt.title("Frequency of Specific Note Types in You Belong With Me")
    plt.xticks(rotation = 80)
    plt.show()
    fig.savefig("Original YBWM Specific Note Type Frequencies.png", bbox_inches='tight')

PlotSpecificNoteFrequencies(formatted_notes)

###### Length of all notes:

def PlotNoteLengthFrequencies(LengthList):

    # Order lenths
    length_order = ['0.0625', '0.125', '0.25', '0.375', '0.5']

    # Creating an empty dictionary
    SongDict = {}

    # Set assign the rhythms as dictionary keys:
    for length in length_order:
        SongDict[length] = 0

    # Loop over the given rhythm list again to give a value to the number of occurrences.
    for length in LengthList:
        slength = str(length)
        if slength in SongDict:
            SongDict[slength] = SongDict[slength]+1

    print(SongDict)

    # Sort notes and frequencies
    Notes     = list(SongDict.keys())
    Frequency = [SongDict[note] for note in Notes]

    # creating the bar plot
    fig, ax = plt.subplots(figsize = (10, 5))
    bars = ax.bar(Notes, Frequency, color = "gold")

    for i in range(len(Notes)):
        ax.text(bars[i].get_x() + bars[i].get_width() / 2, bars[i].get_height(), str(Frequency[i]), ha='
            center', va='bottom')

    plt.xlabel("Length of Note")
    plt.ylabel("Frequency in Song")
    plt.title("Frequency of Note Lengths in You Belong With Me")
    plt.xticks(rotation = 45)
    plt.show()
    fig.savefig("Original YBWM Length of Notes Frequencies.png", bbox_inches='tight')

PlotNoteLengthFrequencies(YBWMmr.interval)
```

Listing 6.7. Analyzing Random "You Belong With Me"

```
from musicpy import *
import random
import matplotlib.pyplot as plt

#Define the Rests and Notes.

rest1o8  = rest(1/8)
rest1o4  = rest(1/4)
rest1o2  = rest(1/2)
rest4o4  = rest(4/4)
FSh31o8  = chord('F#3', interval = [1/8])
FSh31o4  = chord('F#3', interval = [1/4])
FSh33o8  = chord('F#3', interval = [3/8])
GSh31o8  = chord('G#3', interval = [1/8])
GSh31o4  = chord('G#3', interval = [1/4])
GSh31o2  = chord('G#3', interval = [1/2])
ASh31o8  = chord('A#3', interval = [1/8])
ASh31o4  = chord('A#3', interval = [1/4])
ASh31o2  = chord('A#3', interval = [1/2])
B31o8    = chord('B3' , interval = [1/8])
B31o4    = chord('B3' , interval = [1/4])
CSh41o8  = chord('C#4', interval = [1/8])
```

```
CSh41o4  = chord('C#4', interval = [1/4])
CSh43o8  = chord('C#4', interval = [3/8])
DSh41o8  = chord('D#4', interval = [1/8])
DSh41o4  = chord('D#4', interval = [1/4])
FSh41o8  = chord('F#4', interval = [1/8])
FSh41o4  = chord('F#4', interval = [1/4])
GSh41o8  = chord('G#4', interval = [1/8])
GSh41o4  = chord('G#4', interval = [1/4])
GSh41o2  = chord('G#4', interval = [1/2])
ASh41o16 = chord('A#4', interval = [1/16])
ASh41o8  = chord('A#4', interval = [1/8])
ASh41o4  = chord('A#4', interval = [1/4])
ASh43o8  = chord('A#4', interval = [3/8])
B41o16   = chord('B4',  interval = [1/16])
B41o8    = chord('B4',  interval = [1/8])
B41o4    = chord('B4',  interval = [1/4])
B43o8    = chord('B4',  interval = [3/8])
CSh51o8  = chord('C#5', interval = [1/8])

#Make a transition Dictionary.
YBWMD = {rest1o8 : [GSh31o8, CSh41o8, FSh41o8, FSh41o8, ASh41o8, ASh41o8, B41o8],
         rest1o4 : [rest1o8, rest1o8, rest1o4, rest1o4, rest1o2, rest1o2, rest1o2, CSh41o8, DSh41o4,
             ASh41o8, ASh41o8],
         rest1o2 : [rest1o4, rest1o4, rest1o4, rest4o4, rest4o4, ASh41o4, ASh41o4],
         rest4o4 : [rest1o2, rest4o4, rest4o4, rest4o4, rest4o4, rest4o4, rest4o4, rest4o4, CSh41o4,
             CSh41o4, CSh41o4],
         FSh31o8 : [FSh31o8, FSh31o8, CSh41o8, CSh41o8, CSh41o8, CSh41o4],
         FSh31o4 : [GSh31o2, GSh31o2],
         FSh33o8 : [GSh31o8, GSh31o8],
         GSh31o8 : [FSh31o8, FSh31o4, FSh31o4, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8,
             GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8,
             GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8,
             GSh31o4, ASh31o8, ASh31o8, ASh31o4, ASh31o4, ASh31o4, ASh31o4, ASh31o4, ASh31o4, ASh31o4,
             ASh31o4, ASh31o4, ASh31o4, B31o4, B31o4, CSh41o8, CSh41o8, CSh41o8, CSh41o8, CSh41o8, CSh41o8
             , CSh41o4],
         GSh31o4 : [FSh31o8, FSh31o8, FSh31o8, B31o8, CSh41o8],
         GSh31o2 : [ASh31o2, ASh31o2],
         ASh31o8 : [FSh33o8, FSh33o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o4,
             GSh31o4, GSh31o4, GSh31o4, ASh31o8, ASh31o8, ASh31o8, ASh31o8, ASh31o8, ASh31o8, ASh31o8,
             ASh31o8, ASh31o8, ASh31o8, ASh31o8, ASh31o8, ASh31o8, ASh31o8, ASh31o8, ASh31o8, ASh31o8,
             ASh31o8, ASh31o4, ASh31o4, ASh31o4, ASh31o4, B31o8, B31o8, B31o8, B31o4],
         ASh31o4 : [rest1o4, rest1o4, rest1o4, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8,
             GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8, GSh31o8,
             ASh31o8, ASh31o8],
         ASh31o2 : [rest1o2, rest4o4],
         B31o8   : [ASh31o8, ASh31o8, ASh31o8, ASh31o8, ASh31o8, ASh31o8, B31o8, B31o8, B31o8, B31o8,
             B31o8, B31o8, B31o8, B31o8, CSh41o8, CSh41o8, CSh41o8, CSh43o8, CSh43o8],
         B31o4   : [ASh31o8, ASh31o8, B31o8, B31o8, B31o8, B31o8, B31o8, B31o8, B31o8, B31o4, B31o4, B31o4
             , B31o4, B31o4],
         CSh41o8 : [rest1o8, rest1o8, ASh31o8, ASh31o8, ASh31o8, ASh31o8, ASh31o4, ASh31o4, ASh31o4,
             ASh31o4, ASh31o4, ASh31o4, ASh31o4, CSh41o8, CSh41o8, CSh41o8, CSh41o8, CSh41o8, CSh41o8,
             CSh41o8, CSh41o8, CSh41o8, CSh41o8, CSh41o8, CSh41o8, CSh41o8, DSh41o8, DSh41o8, DSh41o8, DSh41o8,
             DSh41o4, DSh41o4, FSh41o4],
         CSh41o4 : [ASh31o8, ASh31o8, ASh31o8, ASh31o8, ASh31o4, FSh41o4, FSh41o4, B43o8, B43o8, B43o8,
             B43o8, B43o8, B43o8],
         CSh43o8 : [rest1o2, rest1o2],
         DSh41o8 : [CSh41o4, CSh41o4, CSh41o4, CSh41o4, CSh41o4, DSh41o8, FSh41o8, FSh41o8, FSh41o8,
             FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o4, GSh41o4, GSh41o4,
             ASh41o4],
         DSh41o4 : [CSh41o4, DSh41o4, DSh41o4, DSh41o4, DSh41o4, DSh41o4, DSh41o4, FSh41o8, FSh41o8,
             FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o4, FSh41o4,
             FSh41o4, FSh41o4, ASh41o4],
         FSh41o8 : [CSh41o8, CSh41o8, DSh41o8, DSh41o8, DSh41o8, DSh41o8, DSh41o8, DSh41o8, DSh41o8,
             DSh41o8, DSh41o8, DSh41o8, DSh41o8, DSh41o4, DSh41o4, DSh41o4, DSh41o4, FSh41o8,
             FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8,
             FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8,
             FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8,
             FSh41o4, FSh41o4, FSh41o4, FSh41o4, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8,
             GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8,
             GSh41o4, GSh41o4, GSh41o4, ASh41o8, ASh41o8, ASh43o8],
         FSh41o4 : [rest1o8, CSh41o8, CSh41o4, CSh41o4, DSh41o8, DSh41o8, DSh41o8, DSh41o8, DSh41o4,
             DSh41o4, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o4, GSh41o8, GSh41o8, GSh41o8,
             GSh41o8, GSh41o8, GSh41o4, GSh41o4, GSh41o4, GSh41o4, GSh41o4, GSh41o4, ASh41o4, ASh41o4, ASh41o4,
             ASh41o4, B41o4, B43o8, B43o8, B43o8, B43o8, B43o8, B43o8],
```

```
        GSh41o8 : [FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8,
             FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o8, FSh41o4, FSh41o4, FSh41o4, FSh41o4,
             FSh41o4, FSh41o4, FSh41o4, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8,
             GSh41o4, GSh41o4, GSh41o4, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8,
             ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8,
             ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8,
             ASh41o8, ASh41o8, ASh41o8, ASh41o4, ASh41o4, ASh41o4, ASh41o4, ASh41o4, ASh41o4, CSh51o8],
        GSh41o4 : [rest1o8, rest1o4, rest1o4, CSh41o8, CSh41o8, FSh41o8, FSh41o8, FSh41o4, FSh41o4,
             FSh41o4, FSh41o4, FSh41o4, FSh41o4, FSh41o4, FSh41o4, FSh41o4, FSh41o4, FSh41o4, FSh41o4,
             FSh41o4, FSh41o4, FSh41o4, FSh41o4, GSh41o8, GSh41o8, ASh41o4],
        GSh41o2 : [rest1o8],
        ASh41o16: [GSh41o4],
        ASh41o8 : [rest1o4, FSh41o4, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8,
             GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8,
             GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o4, GSh41o4, GSh41o4,
             GSh41o4, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8,
             ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o4, ASh41o4, ASh41o4, ASh41o4, ASh41o4, ASh41o4,
             ASh41o4, ASh41o4, ASh41o4, ASh41o4, ASh41o4, ASh41o4, ASh41o4, ASh41o4, ASh41o4, ASh41o4,
             ASh41o4, ASh41o4, ASh41o4, ASh41o4, ASh41o4],
        ASh41o4 : [B31o4, B31o4, B31o4, B31o4, B31o4, B31o4, DSh41o4, DSh41o4, DSh41o4, DSh41o4, DSh41o4,
              DSh41o4, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8,
             GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o8, GSh41o4, GSh41o4, GSh41o4,
             GSh41o4, GSh41o4, GSh41o4, GSh41o4, GSh41o2],
        ASh43o8 : [B41o16],
        B41o16  : [ASh41o16],
        B41o8   : [ASh41o8, B41o8, B41o8, B41o4],
        B41o4   : [ASh41o8, B41o8],
        B43o8   : [ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8, ASh41o8,
             ASh41o8, ASh41o8, ASh41o8],
        CSh51o8 : [ASh41o8, CSh51o8]
       }

#Pick the random state to start from.
possible_options = ['FSh31o8', 'FSh31o4', 'FSh33o8', 'GSh31o8', 'GSh31o4', 'GSh31o2', 'ASh31o8', 'ASh31o4'
     , 'ASh31o2', 'B31o8', 'B31o4', 'CSh41o8', 'CSh41o4', 'CSh43o8', 'DSh41o8', 'DSh41o4', 'FSh41o8', '
     FSh41o4', 'GSh41o8', 'GSh41o4', 'GSh41o2', 'ASh41o16', 'ASh41o8', 'ASh41o4', 'ASh43o8', 'B41o16', '
     B41o8', 'B41o4', 'B43o8', 'CSh51o8']
random.choice(possible_options)

#Initialize a random state.
state = CSh51o8
newsong = 0

#Make a random song.

for i in range(81):
    state = random.choice(YBWMD[state])
    print("state:",i,":",state)
    newsong += state
verse1 = newsong
newsong = 0  #Reset newsong for the next iteration

for i in range(48):
    state = random.choice(YBWMD[state])
    print("state:",i,":",state)
    newsong += state
prechorus = newsong
newsong = 0  #Reset newsong for the next iteration

for i in range(44):
    state = random.choice(YBWMD[state])
    print("state:",i,":",state)
    newsong += state
chorus = newsong
newsong = 0  #Reset newsong for the next iteration

for i in range(4):
    state = random.choice(YBWMD[state])
    print("state:",i,":",state)
    newsong += state
instrumental1 = newsong
newsong = 0  #Reset newsong for the next iteration

for i in range(100):
    state = random.choice(YBWMD[state])
    print("state:",i,":",state)
```

```
    newsong += state
verse2 = newsong
newsong = 0  #Reset newsong for the next iteration

for i in range(12):
    state = random.choice(YBWMD[state])
    print("state:",i,":",state)
    newsong += state
instrumental2 = newsong
newsong = 0  #Reset newsong for the next iteration

for i in range(65):
    state = random.choice(YBWMD[state])
    print("state:",i,":",state)
    newsong += state
bridge = newsong
newsong = 0  #Reset newsong for the next iteration

for i in range(42):
    state = random.choice(YBWMD[state])
    print("state:",i,":",state)
    newsong += state
diffchorus = newsong
newsong = 0  #Reset newsong for the next iteration

for i in range(42):
    state = random.choice(YBWMD[state])
    print("state:",i,":",state)
    newsong += state
outro = newsong
newsong = 0  #Reset newsong for the next iteration

YBWMR = verse1 + prechorus + chorus + instrumental1 + verse2 + prechorus + chorus + chorus + instrumental2
     + bridge + diffchorus + chorus + outro

# Print the length of the final random song
print(len(YBWMR))

for i in range(len(YBWMR)):
    print("state", [i] ,"note=", YBWMR.notes[i], "interval=", YBWMR.interval[i])

play(YBWMR, bpm = 130, instrument = 25)


#### Analysis of Random Song
###### Only the notes:

def PlotNoteFrequencies(NoteList):

    # Order notes
    note_order = ['A#3', 'A#4', 'B3', 'B4', 'C#4', 'C#5', 'D#4', 'F#3', 'F#4', 'G#3', 'G#4']

    # Creating an empty dictionary
    SongDict = {}

    # Set assign the notes as dictionary keys:
    for note in note_order:
        SongDict[note] = 0

    # Loop over the given note list again to give a value to the number of occurances.
    for note in NoteList:
        snote = str(note)
        #print("note",note)
        if snote in SongDict:
            SongDict[snote] = SongDict[snote]+1

    print(SongDict)

    # Sort notes and frequencies
    Notes     = list(SongDict.keys())
    Frequency = [SongDict[note] for note in Notes]

    # creating the bar plot
    fig, ax = plt.subplots(figsize = (10, 5))
    bars = ax.bar(Notes, Frequency, color = "orange")
```

```
    for i in range(len(Notes)):
        ax.text(bars[i].get_x() + bars[i].get_width() / 2, bars[i].get_height(), str(Frequency[i]), ha='
            center', va='bottom')

    plt.xlabel("Note Used")
    plt.ylabel("Frequency in Song")
    plt.title("Frequency of Notes in Newly Generated You Belong With Me")
    plt.xticks(rotation = 45)
    plt.show()
    fig.savefig("Random YBWM Note Frequencies.png", bbox_inches='tight')

PlotNoteFrequencies(YBWMR.notes)

###### Notes and their respective length:

formatted_notes = ['-'.join([str(note), str(interval)]) for note, interval in zip(YBWMR, YBWMR.interval)]
# print(formatted_notes)

def PlotSpecificNoteFrequencies(NoteList):

    # Order specific notes
    specific_note_order = ['A#3-0.125', 'A#3-0.25', 'A#3-0.5', 'A#4-0.0625', 'A#4-0.125', 'A#4-0.25', 'A#4
        -0.375', 'B3-0.125', 'B3-0.25', 'B4-0.0625', 'B4-0.125', 'B4-0.25', 'B4-0.375', 'C#4-0.125', 'C#4-
        0.25', 'C#4-0.375', 'C#5-0.125', 'D#4-0.125', 'D#4-0.25', 'F#3-0.125', 'F#3-0.25', 'F#3-0.375', 'F
        #4-0.125', 'F#4-0.25', 'G#3-0.125', 'G#3-0.25', 'G#3-0.5', 'G#4-0.125', 'G#4-0.25', 'G#4-0.5']

    # Creating an empty dictionary
    SongDict = {}

    # Set assign the notes as dictionary keys:
    for note in specific_note_order:
        SongDict[note] = 0

    # Loop over the given note list again to give a value to the number of occurances.
    for note in NoteList:
        snote = str(note)
#         print("note",note)
        if snote in SongDict:
            SongDict[snote] = SongDict[snote]+1

    print(SongDict)

    # Sort notes and frequencies
    Notes     = list(SongDict.keys())
    Frequency = [SongDict[note] for note in Notes]

    # creating the bar plot
    fig, ax = plt.subplots(figsize = (10, 5))
    bars = ax.bar(Notes, Frequency, color = "orange")

    for i in range(len(Notes)):
        ax.text(bars[i].get_x() + bars[i].get_width() / 2, bars[i].get_height(), str(Frequency[i]), ha='
            center', va='bottom')

    plt.xlabel("Specific Note Type Used")
    plt.ylabel("Frequency in Song")
    plt.title("Frequency of Specific Note Types in Newly Generated You Belong With Me")
    plt.xticks(rotation = 80)
    plt.show()
    fig.savefig("Random YBWM Specific Note Type Frequencies.png", bbox_inches='tight')

PlotSpecificNoteFrequencies(formatted_notes)

###### Length of all notes:

def PlotNoteLengthFrequencies(LengthList):

    # Order lenths
    length_order = ['0.0625', '0.125', '0.25', '0.375', '0.5']

    # Creating an empty dictionary
    SongDict = {}

    # Set assign the rhythms as dictionary keys:
    for length in length_order:
        SongDict[length] = 0
```

```
    # Loop over the given rhythm list again to give a value to the number of occurances.
    for length in LengthList:
        slength = str(length)
        #print("length",length)
        if slength in SongDict:
            SongDict[slength] = SongDict[slength]+1

    print(SongDict)

    # Sort notes and frequencies
    Notes     = list(SongDict.keys())
    Frequency = [SongDict[note] for note in Notes]

    # creating the bar plot
    fig, ax = plt.subplots(figsize = (10, 5))
    bars = ax.bar(Notes, Frequency, color = "orange")

    for i in range(len(Notes)):
        ax.text(bars[i].get_x() + bars[i].get_width() / 2, bars[i].get_height(), str(Frequency[i]), ha='
            center', va='bottom')

    plt.xlabel("Length of Note")
    plt.ylabel("Frequency in Song")
    plt.title("Frequency of Note Lengths in Newly Generated You Belong With Me")
    plt.xticks(rotation = 45)
    plt.show()
    fig.savefig("Random YBWM Length of Notes Frequencies.png", bbox_inches='tight')

PlotNoteLengthFrequencies(YBWMR.interval)
```

Listing 6.8. “Don’t Blame Me”

```
from musicpy import *

#Define the Rests and Notes.

rest1o32 = rest(1/32)
rest1o8  = rest(1/8)
rest1o4  = rest(1/4)
rest1o2  = rest(1/2)
rest4o4  = rest(4/4)
D31o8    = chord('D3', interval = [1/8])
D31o4    = chord('D3', interval = [1/4])
E31o4    = chord('E3', interval = [1/4])
G31o8    = chord('G3', interval = [1/8])
A33o16   = chord('A3', interval = [3/16])
A31o8    = chord('A3', interval = [1/8])
A31o4    = chord('A3', interval = [1/4])
A33o8    = chord('A3', interval = [3/8])
A31o12   = chord('A3', interval = [1/12])
B33o8    = chord('B3', interval = [3/8])
C41o8    = chord('C4', interval = [1/8])
C41o4    = chord('C4', interval = [1/4])
C43o8    = chord('C4', interval = [3/8])
C41o12   = chord('C4', interval = [1/12])
C41o6    = chord('C4', interval = [1/6])
D41o8    = chord('D4', interval = [1/8])
D41o4    = chord('D4', interval = [1/4])
D43o8    = chord('D4', interval = [3/8])
D41o12   = chord('D4', interval = [1/12])
D41o6    = chord('D4', interval = [1/6])
Eb41o4   = chord('Eb4', interval = [1/4])
E41o8    = chord('E4', interval = [1/8])
E41o4    = chord('E4', interval = [1/4])
E41o12   = chord('E4', interval = [1/12])
E41o6    = chord('E4', interval = [1/6])
G41o4    = chord('G4', interval = [1/4])
A43o16   = chord('A4', interval = [3/16])
A41o8    = chord('A4', interval = [1/8])
A41o4    = chord('A4', interval = [1/4])
A41o6    = chord('A4', interval = [1/6])
A41o12   = chord('A4', interval = [1/12])
C51o8    = chord('C5', interval = [1/8])
```

```
C5io4    = chord('C5', interval = [1/4])
C5io2    = chord('C5', interval = [1/2])
C5io12   = chord('C5', interval = [1/12])
D5io8    = chord('D5', interval = [1/8])
D5io4    = chord('D5', interval = [1/4])
D5io12   = chord('D5', interval = [1/12])
E5io8    = chord('E5', interval = [1/8])
E5io4    = chord('E5', interval = [1/4])
E5io2    = chord('E5', interval = [1/2])
E5io12   = chord('E5', interval = [1/12])

DBM = C4io4+A3io8+A3io4+C4io8+C4io4+C4io8+C4io8+C4io8+A3io8+A3io8+A3io8+C4io8+C4io8+C4io8+C4io8+D4io8+
      A3io8
+A3io8+G3io8+A3io8+C43o8+rest1o2+rest1o4+E4io4+D4io8+C4io8+C4io8+D4io8+D4io4+D4io8+E4io8+D4io8+C4io8
+A3io8+G3io8+C4io8+A3io8+A3io8+G3io8+C4io8+A3io8+A3io8+G3io8+A3io8+C43o8+rest1o2+rest4o4+D4io8+D43o8
+rest1o2+D4io8+C4io8+D4io8+C4io8+D4io4+Eb4io4+D4io8+D43o8+rest1o2+D4io8+C4io8+D4io8+C4io8+D4io8+C4io8
+E4io4+D4io8+D43o8+C4io8+C43o8+A3io8+A33o8+D3io8+D3io8+E3io4+rest4o4+rest4o4+D4io8+D43o8+rest1o2
+D4io8+C4io8+D4io8+C4io8+D4io4+Eb4io4+D4io8+D43o8+rest1o4+rest1o8+C4io8+D4io8+C4io8+D4io8+C4io8
+D4io8+C4io8+E4io4+D4io8+D43o8+C4io8+C43o8+A3io8+A33o8+D3io8+D3io8+E3io4+rest4o4+rest1o2+D4io8
+E4io8+D4io8+C4io8+A3io4+C4io4+A3io4+C4io4+A3io4+C4io4+D4io8+E4io8+D4io4+A3io4+C4io4+A3io4+C4io4
+A3io4+C4io4+D4io8+E4io8+D4io4+A3io4+C4io4+A3io4+C4io4+D4io8+E4io8+D4io8+rest1o4
+rest1o8+C4io8+D4io8+C4io8+D4io8+C4io8+D4io4+D4io4+rest1o4+C4io8+A33o16+rest1o32+rest1o4+C4io4
+A3io8+A3io4+C4io8+C4io4+C4io8+C4io8+C4io8+A3io8+A3io8+A3io8+C4io8+C4io8+C4io8+C4io8+D4io8+A3io8
+A3io8+G3io8+A3io8+C43o8+rest1o2+rest1o4+E4io4+D4io8+C4io8+C4io8+D4io8+D4io4+D4io8+E4io8+D4io8
+C4io8+A3io8+G3io8+C4io8+A3io8+A3io8+G3io8+C4io8+A3io8+A3io8+G3io8+A3io8+C43o8+rest1o2+rest1o4
+C4io4+A3io8+A3io4+C4io8+C4io4+C4io8+C4io8+C4io8+A3io8+A3io8+A3io8+C4io8+C4io8+C4io8+C4io8+D4io8
+A3io8+A3io8+G3io8+A3io8+C43o8+rest1o4+rest1o8+E4io8+E4io4+E4io4+D4io8+C4io8+C4io8+D4io8+D4io4
+D4io8+E4io8+D4io8+C4io8+A3io8+G3io8+C4io8+A3io8+A3io8+G3io8+C4io8+A3io8+A3io8+G3io8+A3io8+C43o8
+rest1o2+D4io8+D43o8+rest1o2+D4io8+C4io8+D4io8+C4io8+D4io8+C4io8+Eb4io4+D4io8+D43o8+rest1o2+D4io8
+C4io8+D4io8+C4io8+D4io8+C4io8+E4io4+D4io8+D43o8+C4io8+C43o8+A3io8+A3io4+A3io8+D3io4+E3io4+rest1o4
+rest1o8+D4io8+D4io4+E4io4+rest4o4+D4io8+D43o8+C4io8+B33o8+D4io8+C4io8+D4io8+C4io8+D4io8+C4io8+Eb4io4
+D4io8+D43o8+rest1o2+D4io8+C4io8+D4io8+C4io8+D4io8+C4io8+E4io4+D4io8+D4io8+D4io8+D4io8+D4io8+C4io8
+C4io4+C4io6+D4io6+E4io6+rest1o4+D5io8+E5io8+D5io4+C5io2+G4io4+rest1o8+C4io8+D4io8+C4io8+D4io8+E4io8
+D4io8+C4io8+A3io4+C4io4+A3io4+C4io4+A3io4+C4io4+D4io8+E4io8+D4io4+A3io4+C4io4+A3io4+C4io4+A3io4+C4io4
+D4io8+E4io8+D4io4+A3io4+C4io4+A3io4+C4io4+A3io4+C4io4+D4io8+E4io8+D4io4+rest1o4+rest1o8+C4io8+D4io8
+C4io8+D4io8+C4io8+D4io4+D4io4+rest1o4+C4io8+A33o16+rest1o32+rest1o4+C4io4+A3io8+A3io4+C4io8+C4io4
+C4io8+C4io8+C4io8+A3io8+A3io8+C4io8+C4io8+C4io8+C4io8+D4io8+A3io8+A3io8+G3io8+A3io8+C43o8
+rest1o2+rest1o4+E4io4+D4io8+C4io8+C4io8+D4io8+D4io4+D4io8+E4io8+D4io8+C4io8+A3io8+G3io8+C4io8
+A3io8+A3io8+G3io8+C4io8+A3io8+A3io8+G3io8+A3io8+C43o8+rest1o2+rest1o4+C4io4+A3io8+A3io4+C4io8
+C4io4+C4io8+C4io8+C4io8+A3io8+A3io8+A3io8+C4io8+C4io8+C4io8+C4io8+D4io8+A3io8+A3io8+G3io8+A3io8
+C43o8+rest1o4+rest1o8+E4io8+E4io4+E4io4+D4io8+C4io8+C4io8+D4io8+D4io4+D4io8+E4io8+D4io8+C4io8
+A3io8+G3io8+C4io8+A3io8+A3io8+G3io8+C4io8+A3io8+A3io8+G3io8+A3io8+C43o8+A4io6+A4io6+A4io6+A4io4
+rest1o4+C5io4+rest1o4+D4io12+E4io12+D4io12+D4io8+C4io8+D4io12+E4io12+D4io12+D4io8+C4io8+D4io12
+C4io12+A3io12+A3io4+rest1o4+rest1o8+C4io8+D4io12+C4io12+A3io12+A3io4+A4io6+A4io6+A4io6+A4io4
+rest1o4+C5io4+rest1o4+D4io12+E4io12+D4io12+D4io8+C4io8+D4io12+E4io12+D4io12+D4io8+C4io8+D4io12
+C4io12+A3io12+A3io4+rest1o4+rest1o8+C4io8+D4io12+C4io12+A3io12+A3io4+rest1o2+rest4o4+D4io12+E4io12
+D4io12+D4io8+C4io8+D4io12+E4io12+D4io12+D4io8+C4io8+D4io12+C4io12+A3io12+A3io4+rest1o2+rest1o2
+rest1o4+rest1o8+E4io8+E4io4+E4io4+D4io8+C4io8+C4io8+D4io8+D4io4+D4io8+E4io8+D4io8+C4io8+A3io8
+G3io8+C4io8+A3io8+A3io8+G3io8+C4io8+A3io8+A3io8+G3io8+C5io8+C5io8+C5io8+D5io8+E5io8+C5io8+D5io4
+C5io8+A43o16+rest1o32+rest1o4+C5io8+D5io8+C5io8+E5io8+E5io4+C4io4+A3io8+A3io4+C4io8+C4io4+C4io8
+C4io8+C4io8+A3io8+A3io8+A3io8+C4io8+C4io8+C4io8+C4io8+D4io8+A3io8+A3io8+G3io8+A3io8+C43o8+rest1o2
+rest1o4+E4io4+D4io8+C4io8+C4io8+D4io8+D4io4+D4io8+E4io8+D4io8+C4io8+A3io8+A3io8+C4io8+A3io8+A3io8
+G3io8+C4io8+A3io8+A3io8+G3io8+A3io8+C43o8+rest1o2+rest1o4+C4io4+A3io8+A3io4+C4io8+C4io4+C4io8+C4io8
+C4io8+A3io8+A3io8+A3io8+C4io8+C4io8+C4io8+C4io8+D4io8+A3io8+A3io8+G3io8+A3io8+C43o8+rest1o4+rest1o8
+E4io8+E4io4+E4io4+D4io8+C4io8+C4io8+D4io8+D4io4+D4io8+E4io8+D4io8+C4io8+A3io8+G3io8+C4io8+A3io8
+A3io8+G3io8+C4io8+A3io8+A3io8+G3io8+A3io8+C43o8+A4io6+A4io6+A4io6+A4io4+rest1o4+E5io2+D5io12+E5io12
+D5io12+D5io8+C5io8+D5io12+E5io12+D5io12+D5io8+C5io8+D5io12+C5io12+A4io12+A4io4+rest1o4+rest1o8+C5io8
+D5io12+C5io12+A4io12+A4io8+C5io8+rest1o4+rest1o8+E4io8+E4io4+E4io4+D4io8+C4io8+C4io8+D4io8+D4io4
+D4io8+E4io8+D4io8+C4io8+A3io8+G3io8+C4io8+A3io8+A3io8+G3io8+C4io8+A3io8+A3io8+G3io8+A3io8
+C43o8+rest1o2+rest4o4

play(DBM, bpm = 136)
```

Listing 6.9. Analyzing Original "Don't Blame Me"

```
from musicpy import *
import matplotlib.pyplot as plt

#Define the Rests and Notes.
rest1o32 = rest(1/32)
rest1o8  = rest(1/8)
rest1o4  = rest(1/4)
rest1o2  = rest(1/2)
rest4o4  = rest(4/4)
```

```
D31o8    = chord('D3', interval = [1/8])
D31o4    = chord('D3', interval = [1/4])
E31o4    = chord('E3', interval = [1/4])
G31o8    = chord('G3', interval = [1/8])
A33o16   = chord('A3', interval = [3/16])
A31o8    = chord('A3', interval = [1/8])
A31o4    = chord('A3', interval = [1/4])
A33o8    = chord('A3', interval = [3/8])
A31o12   = chord('A3', interval = [1/12])
B33o8    = chord('B3', interval = [3/8])
C41o8    = chord('C4', interval = [1/8])
C41o4    = chord('C4', interval = [1/4])
C43o8    = chord('C4', interval = [3/8])
C41o12   = chord('C4', interval = [1/12])
C41o6    = chord('C4', interval = [1/6])
D41o8    = chord('D4', interval = [1/8])
D41o4    = chord('D4', interval = [1/4])
D43o8    = chord('D4', interval = [3/8])
D41o12   = chord('D4', interval = [1/12])
D41o6    = chord('D4', interval = [1/6])
Eb41o4   = chord('Eb4', interval = [1/4])
E41o8    = chord('E4', interval = [1/8])
E41o4    = chord('E4', interval = [1/4])
E41o12   = chord('E4', interval = [1/12])
E41o6    = chord('E4', interval = [1/6])
G41o4    = chord('G4', interval = [1/4])
A43o16   = chord('A4', interval = [3/16])
A41o8    = chord('A4', interval = [1/8])
A41o4    = chord('A4', interval = [1/4])
A41o6    = chord('A4', interval = [1/6])
A41o12   = chord('A4', interval = [1/12])
C51o8    = chord('C5', interval = [1/8])
C51o4    = chord('C5', interval = [1/4])
C51o2    = chord('C5', interval = [1/2])
C51o12   = chord('C5', interval = [1/12])
D51o8    = chord('D5', interval = [1/8])
D51o4    = chord('D5', interval = [1/4])
D51o12   = chord('D5', interval = [1/12])
E51o8    = chord('E5', interval = [1/8])
E51o4    = chord('E5', interval = [1/4])
E51o2    = chord('E5', interval = [1/2])
E51o12   = chord('E5', interval = [1/12])

DBM = C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8+C41o8+A31o8+A31o8+A31o8+C41o8+C41o8+C41o8+C41o8+D41o8+
      A31o8
+A31o8+G31o8+A31o8+C43o8+rest1o2+rest1o4+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8
+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o2+rest4o4+D41o8+D43o8
+rest1o2+D41o8+C41o8+D41o8+C41o8+D41o4+Eb41o4+D41o8+D43o8+rest1o2+D41o8+C41o8+D41o8+C41o8+D41o8+C41o8
+E41o4+D41o8+D43o8+C41o8+C43o8+A31o8+A33o8+D31o8+D31o8+E31o4+rest4o4+rest4o4+D41o8+D43o8+rest1o2+D41o8
+C41o8+D41o8+C41o8+D41o4+Eb41o4+D41o8+D43o8+rest1o4+rest1o8+C41o8+D41o8+C41o8+D41o8+C41o8+D41o8+C41o8
+E41o4+D41o8+D43o8+C41o8+C43o8+A31o8+A33o8+D31o8+D31o8+E31o4+rest4o4+rest1o2+D41o8+E41o8+D41o8+C41o8
+A31o4+C41o4+A31o4+C41o4+A31o4+C41o4+D41o8+E41o8+D41o4+A31o4+C41o4+A31o4+C41o4+A31o4+C41o4+D41o8+E41o8
+D41o4+A31o4+C41o4+A31o4+C41o4+A31o4+C41o4+D41o8+E41o8+D41o4+rest1o4+rest1o8+C41o8+D41o8+C41o8+D41o8
+C41o8+D41o4+D41o4+rest1o4+C41o8+A33o16+rest1o32+rest1o4+C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8
+C41o8+A31o8+A31o8+A31o8+C41o8+C41o8+C41o8+C41o8+D41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o2+rest1o4
+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8
+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o2+rest1o4+C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8+C41o8
+A31o8+A31o8+A31o8+C41o8+C41o8+A31o8+G31o8+A31o8+C43o8+rest1o4+rest1o8+E41o8
+E41o4+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8
+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o2+D41o8+D43o8+rest1o2+D41o8+C41o8+D41o8+C41o8
+D41o8+C41o8+Eb41o4+D41o8+D43o8+rest1o2+D41o8+C41o8+D41o8+C41o8+D41o8+C41o8+E41o4+D41o8+D43o8+C41o8
+C43o8+A31o8+A31o4+A31o8+D31o4+E31o4+rest1o4+rest1o8+D41o8+D41o4+E41o4+rest4o4+D41o8+D43o8+C41o8+B33o8
+D41o8+C41o8+D41o8+C41o8+D41o8+C41o8+Eb41o4+D41o8+D43o8+rest1o2+D41o8+C41o8+D41o8+C41o8+D41o8+C41o8
+E41o4+D41o8+D41o8+D41o8+D41o8+C41o8+C41o4+C41o6+D41o6+E41o6+rest1o4+D51o8+E51o8+D51o4+C51o2
+G41o4+rest1o8+C41o8+D41o8+C41o8+D41o8+E41o8+D41o8+C41o8+A31o4+C41o4+A31o4+C41o4+A31o4+C41o4+D41o8
+E41o8+D41o4+A31o4+C41o4+A31o4+C41o4+A31o4+C41o4+D41o8+E41o8+D41o4+A31o4+C41o4+A31o4+C41o4+A31o4
+C41o4+D41o8+E41o8+D41o4+rest1o4+rest1o8+C41o8+D41o8+C41o8+D41o8+C41o8+D41o4+D41o4+rest1o4+C41o8
+A33o16+rest1o32+rest1o4+C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8+C41o8+A31o8+A31o8+A31o8+C41o8
+C41o8+C41o8+C41o8+D41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o2+rest1o4+E41o4+D41o8+C41o8+C41o8
+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8
+A31o8+C43o8+rest1o2+rest1o4+C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8+C41o8+A31o8+A31o8+A31o8
+C41o8+C41o8+C41o8+C41o8+D41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o4+rest1o8+E41o8+E41o4+E41o4
+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8
+A31o8+A31o8+G31o8+A31o8+C43o8+A41o6+A41o6+A41o6+A41o4+rest1o4+C51o4+rest1o4+D41o12+E41o12+D41o12
+D41o8+C41o8+D41o12+E41o12+D41o12+D41o8+C41o8+D41o12+C41o12+A31o12+A31o4+rest1o4+rest1o8+C41o8
+D41o12+C41o12+A31o12+A31o4+A41o6+A41o6+A41o6+A41o4+rest1o4+C51o4+rest1o4+D41o12+E41o12+D41o12
```

```
+D41o8+C41o8+D41o12+E41o12+D41o12+D41o8+C41o8+D41o12+C41o12+A31o12+A31o4+rest1o4+rest1o8+C41o8
+D41o12+C41o12+A31o12+A31o4+rest1o2+rest4o4+D41o12+E41o12+D41o12+D41o8+C41o8+D41o12+E41o12+D41o12
+D41o8+C41o8+D41o12+C41o12+A31o12+A31o4+rest1o2+rest1o2+rest1o4+rest1o8+E41o8+E41o4+E41o4+D41o8
+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8
+A31o8+G31o8+C51o8+C51o8+C51o8+D51o8+E51o8+C51o8+D51o4+C51o8+A43o16+rest1o32+rest1o4+C51o8+D51o8
+C51o8+E51o8+E51o4+C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8+C41o8+A31o8+A31o8+A31o8+C41o8+C41o8
+C41o8+C41o8+D41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o2+rest1o4+E41o4+D41o8+C41o8+C41o8+D41o8
+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+A31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8
+C43o8+rest1o2+rest1o4+C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8+C41o8+A31o8+A31o8+A31o8+C41o8
+C41o8+C41o8+C41o8+D41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o4+rest1o8+E41o8+E41o4+E41o4+D41o8
+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8
+A31o8+G31o8+A31o8+C43o8+A41o6+A41o6+A41o6+A41o4+rest1o4+E51o2+D51o12+E51o12+D51o12+D51o8+C51o8
+D51o12+E51o12+D51o12+D51o8+C51o8+D51o12+C51o12+A41o12+A41o4+rest1o4+rest1o8+C51o8+D51o12+C51o12
+A41o12+A41o8+C51o8+rest1o4+rest1o8+E41o8+E41o4+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8
+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8

play(DBM, bpm = 136)

# Given string
songstr = 'C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8+C41o8+A31o8+A31o8+A31o8+C41o8+C41o8+C41o8+C41o8+D41o8
+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o2+rest1o4+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8
+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o2+rest4o4+D41o8
+D43o8+rest1o2+D41o8+C41o8+D41o8+C41o8+D41o4+Eb41o4+D41o8+D43o8+rest1o2+D41o8+C41o8+D41o8+C41o8+D41o8
+C41o8+E41o4+D41o8+D43o8+C41o8+C43o8+A31o8+A33o8+D31o8+D31o8+E31o4+rest4o4+rest4o4+D41o8+D43o8+rest1o2
+D41o8+C41o8+D41o8+C41o8+D41o4+Eb41o4+D41o8+D43o8+rest1o4+rest1o8+C41o8+D41o8+C41o8+D41o8+C41o8+D41o8
+C41o8+E41o4+D41o8+D43o8+C41o8+C43o8+A31o8+A33o8+D31o8+D31o8+E31o4+rest4o4+rest1o2+D41o8+E41o8+D41o8
+C41o8+A31o4+C41o4+A31o4+C41o4+A31o4+C41o4+D41o8+E41o8+D41o4+A31o4+C41o4+A31o4+C41o4+A31o4+C41o4+D41o8
+E41o8+D41o4+A31o4+C41o4+A31o4+C41o4+A31o4+C41o4+D41o8+E41o8+D41o4+rest1o4+rest1o8+C41o8+D41o8+C41o8
+D41o8+C41o8+D41o4+D41o4+rest1o4+C41o8+A33o16+rest1o32+rest1o4+C41o4+A31o8+A31o4+C41o8+C41o4+C41o8
+C41o8+C41o8+A31o8+A31o8+A31o8+C41o8+C41o8+C41o8+C41o8+D41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o2
+rest1o4+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8
+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o2+rest1o4+C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8
+C41o8+A31o8+A31o8+A31o8+C41o8+C41o8+C41o8+C41o8+D41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o4+rest1o8
+E41o8+E41o4+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8
+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o2+D41o8+D43o8+rest1o2+D41o8+C41o8+D41o8+C41o8+D41o8
+C41o8+Eb41o4+D41o8+D43o8+rest1o2+D41o8+C41o8+D41o8+C41o8+D41o8+C41o8+E41o4+D41o8+D43o8+C41o8+C43o8
+A31o8+A31o4+A31o8+D31o4+E31o4+rest1o4+rest1o8+D41o8+D41o4+E41o4+rest4o4+D41o8+D43o8+C41o8+B33o8
+D41o8+C41o8+D41o8+C41o8+D41o8+C41o8+Eb41o4+D41o8+D43o8+rest1o2+D41o8+C41o8+D41o8+C41o8+D41o8+C41o8
+E41o4+D41o8+D41o8+D41o8+D41o8+D41o8+C41o8+C41o4+C41o6+D41o6+E41o6+rest1o4+D51o8+E51o8+D51o4+C51o2
+G41o4+rest1o8+C41o8+D41o8+C41o8+D41o8+E41o8+D41o8+C41o8+A31o4+C41o4+A31o4+C41o4+A31o4+C41o4+D41o8
+E41o8+D41o4+A31o4+C41o4+A31o4+C41o4+A31o4+C41o4+D41o8+E41o8+D41o4+A31o4+C41o4+A31o4+C41o4+A31o4
+C41o4+D41o8+E41o8+D41o4+rest1o4+rest1o8+C41o8+D41o8+C41o8+D41o8+D41o4+D41o4+rest1o4+C41o8
+A33o16+rest1o32+rest1o4+C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8+C41o8+A31o8+A31o8+A31o8+C41o8
+C41o8+C41o8+C41o8+D41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o2+rest1o4+E41o4+D41o8+C41o8+C41o8
+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8
+A31o8+C43o8+rest1o2+rest1o4+C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8+C41o8+A31o8+A31o8+A31o8+C41o8
+C41o8+C41o8+C41o8+D41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o4+rest1o8+E41o8+E41o4+E41o4+D41o8+C41o8
+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8
+A31o8+C43o8+A41o6+A41o6+A41o6+A41o4+rest1o4+C51o4+rest1o4+D41o12+E41o12+D41o12+D41o8+C41o8+D41o12
+E41o12+D41o12+D41o8+C41o8+D41o12+C41o12+A31o12+A31o4+rest1o4+rest1o8+C41o8+D41o12+C41o12+A31o12
+A31o4+A41o6+A41o6+A41o6+A41o4+rest1o4+C51o4+rest1o4+D41o12+E41o12+D41o12+D41o8+C41o8+D41o12+E41o12
+D41o12+D41o8+C41o8+D41o12+C41o12+A31o12+A31o4+rest1o4+rest1o8+C41o8+D41o12+C41o12+A31o12+A31o4+rest1o2
+rest4o4+D41o12+E41o12+D41o12+D41o8+C41o8+D41o12+E41o12+D41o12+D41o8+C41o8+D41o12+C41o12+A31o12+A31o4
+rest1o2+rest1o2+rest1o4+rest1o8+E41o8+E41o4+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8
+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C51o8+C51o8+C51o8+D51o8+E51o8
+C51o8+D51o4+C51o8+A43o16+rest1o32+rest1o4+C51o8+D51o8+C51o8+E51o8+E51o4+C41o4+A31o8+A31o4+C41o8
+C41o4+C41o8+C41o8+C41o8+A31o8+A31o8+A31o8+C41o8+C41o8+C41o8+C41o8+D41o8+A31o8+A31o8+G31o8+A31o8
+C43o8+rest1o2+rest1o4+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+A31o8
+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o2+rest1o4+C41o4+A31o8+A31o4
+C41o8+C41o4+C41o8+C41o8+C41o8+A31o8+A31o8+A31o8+C41o8+C41o8+C41o8+C41o8+D41o8+A31o8+A31o8+G31o8
+A31o8+C43o8+rest1o4+rest1o8+E41o8+E41o4+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8
+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8+A41o6+A41o6+A41o6
+A41o4+rest1o4+E51o2+D51o12+E51o12+D51o12+D51o8+C51o8+D51o12+E51o12+D51o12+D51o8+C51o8+D51o12
+C51o12+A41o12+A41o4+rest1o4+rest1o8+C51o8+D51o12+C51o12+A41o12+A41o8+C51o8+rest1o4+rest1o8
+E41o8+E41o4+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8
+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8'

songList = songstr.split("+")

# Filter out the 'rests'
filtered_items = [item for item in songList if 'rest' not in item]

# Joining the filtered items back into a string
result = '+'.join(filtered_items)
print(result)
```

```
DBMmr = C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8+C41o8+A31o8+A31o8+A31o8+C41o8+C41o8+C41o8
+C41o8+D41o8+A31o8+A31o8+G31o8+A31o8+C43o8+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8
+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8+D41o8
+D43o8+D41o8+C41o8+D41o8+C41o8+D41o4+Eb41o4+D41o8+D43o8+D41o8+C41o8+D41o8+C41o8+D41o8+C41o8
+E41o4+D41o8+D43o8+C41o8+C43o8+A31o8+A33o8+D31o8+D31o8+E31o4+D41o8+D43o8+D41o8+C41o8+D41o8
+C41o8+D41o4+Eb41o4+D41o8+D43o8+C41o8+D41o8+C41o8+D41o8+C41o8+D41o8+C41o8+E41o4+D41o8+D43o8
+C41o8+C43o8+A31o8+A33o8+D31o8+D31o8+E31o4+D41o8+E41o8+D41o8+C41o8+A31o4+C41o4+A31o4+C41o4
+A31o4+C41o4+D41o8+E41o8+D41o4+A31o4+C41o4+A31o4+C41o4+A31o4+C41o4+D41o8+E41o8+D41o4+A31o4
+C41o4+A31o4+C41o4+A31o4+C41o4+D41o8+E41o8+D41o4+C41o8+D41o8+C41o8+D41o8+C41o8+D41o4+D41o4
+C41o8+A33o16+C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8+C41o8+A31o8+A31o8+A31o8+C41o8+C41o8
+C41o8+C41o8+D41o8+A31o8+A31o8+G31o8+A31o8+C43o8+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8
+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8
+C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8+C41o8+A31o8+A31o8+A31o8+C41o8+C41o8+C41o8+C41o8
+D41o8+A31o8+A31o8+G31o8+A31o8+C43o8+E41o8+E41o4+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8
+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8
+D41o8+D43o8+D41o8+C41o8+D41o8+C41o8+D41o8+C41o8+Eb41o4+D41o8+D43o8+D41o8+C41o8+D41o8+C41o8
+D41o8+C41o8+E41o4+D41o8+D43o8+C41o8+C43o8+A31o8+A31o4+A31o8+D31o4+E31o4+D41o8+D41o4+E41o4
+D41o8+D43o8+C41o8+B33o8+D41o8+C41o8+D41o8+C41o8+D41o8+C41o8+Eb41o4+D41o8+D43o8+D41o8+C41o8
+D41o8+C41o8+D41o8+C41o8+E41o4+D41o8+D41o8+D41o8+D41o8+D41o8+C41o8+C41o4+C41o6+D41o6+E41o6
+D51o8+E51o8+D51o4+C51o2+G41o4+C41o8+D41o8+C41o8+D41o8+E41o8+D41o8+C41o8+A31o4+C41o4+A31o4
+C41o4+A31o4+C41o4+D41o8+E41o8+D41o4+A31o4+C41o4+A31o4+C41o4+D41o8+E41o8+D41o4
+A31o4+C41o4+A31o4+C41o4+A31o4+C41o4+D41o8+E41o8+D41o4+C41o8+D41o8+C41o8+D41o8+C41o8+D41o4
+D41o4+C41o8+A33o16+C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8+C41o8+A31o8+A31o8+A31o8+C41o8
+C41o8+C41o8+C41o8+D41o8+A31o8+A31o8+G31o8+A31o8+C43o8+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4
+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8
+C43o8+C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8+A31o8+A31o8+A31o8+C41o8+C41o8+C41o8
+C41o8+D41o8+A31o8+A31o8+G31o8+A31o8+C43o8+E41o8+E41o4+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4
+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8
+C43o8+A41o6+A41o6+A41o6+A41o4+C51o4+D41o12+E41o12+D41o12+D41o8+C41o8+D41o12+E41o12+D41o12
+D41o8+C41o8+D41o12+C41o12+A31o12+A31o4+C41o8+D41o12+C41o12+A31o12+A31o4+A41o6+A41o6+A41o6
+A41o4+C51o4+D41o12+E41o12+D41o12+D41o8+C41o8+D41o12+E41o12+D41o12+D41o8+C41o8+D41o12+C41o12
+A31o12+A31o4+C41o8+D41o12+C41o12+A31o12+A31o4+D41o12+E41o12+D41o12+D41o8+C41o8+D41o12+E41o12
+D41o12+D41o8+C41o8+D41o12+C41o12+A31o12+A31o4+E41o8+E41o4+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4
+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C51o8+C51o8
+C51o8+D51o8+E51o8+C51o8+D51o4+C51o8+A43o16+C51o8+D51o8+C51o8+E51o8+E51o4+C41o4+A31o8+A31o4
+C41o8+C41o4+C41o8+C41o8+C41o8+A31o8+A31o8+A31o8+C41o8+C41o8+C41o8+C41o8+D41o8+A31o8+A31o8+G31o8
+A31o8+C43o8+E41o4+D41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+A31o8+C41o8+A31o8
+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8+C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8+C41o8
+A31o8+A31o8+A31o8+C41o8+C41o8+C41o8+C41o8+D41o8+A31o8+A31o8+G31o8+A31o8+C43o8+E41o8+E41o4+E41o4
+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8
+A31o8+A31o8+G31o8+A31o8+C43o8+A41o6+A41o6+A41o6+A41o4+E51o2+D51o12+E51o12+D51o12+D51o8+C51o8
+D51o12+E51o12+D51o12+D51o8+C51o8+D51o12+C51o12+A41o12+A41o4+C51o8+D51o12+C51o12+A41o12+A41o8
+C51o8+E41o8+E41o4+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8
+A31o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8
```

```
#### Analysis of Original Song
###### Only the notes:

def PlotNoteFrequencies(NoteList):

    # Order notes
    note_order = ['A3', 'A4', 'B3', 'C4', 'C5', 'D3', 'D4', 'D5', 'Eb4', 'E3', 'E4', 'E5', 'G3', 'G4']

    # Creating an empty dictionary
    SongDict = {}

    # Set assign the notes as dictionary keys:
    for note in note_order:
        SongDict[note] = 0

    # Loop over the given note list again to give a value to the number of occurrences.
    for note in NoteList:
        snote = str(note)
        #print("note",note)
        if snote in SongDict:
            SongDict[snote] = SongDict[snote]+1

    print(SongDict)

    # Sort notes and frequencies
    Notes     = list(SongDict.keys())
    Frequency = [SongDict[note] for note in Notes]

    # creating the bar plot
    fig, ax = plt.subplots(figsize = (10, 5))
```

```
    bars = ax.bar(Notes, Frequency, color = "dimgray")

    for i in range(len(Notes)):
        ax.text(bars[i].get_x() + bars[i].get_width() / 2, bars[i].get_height(), str(Frequency[i]), ha='
            center', va='bottom')

    plt.xlabel("Note Used")
    plt.ylabel("Frequency in Song")
    plt.title("Frequency of Notes in Don't Blame Me")
    plt.xticks(rotation = 45)
    plt.show()
    fig.savefig("Original DBM Note Frequencies.png", bbox_inches='tight')


PlotNoteFrequencies(DBM.notes)

###### Notes and their respective length:

formatted_notes = ['-'.join([str(note), str(interval)]) for note, interval in zip(DBMnr, DBMnr.interval)]
print(formatted_notes)

def PlotSpecificNoteFrequencies(NoteList):

    # Order specific notes
    specific_note_order = ['A3-0.08333333333333333', 'A3-0.125', 'A3-0.1875', 'A3-0.25', 'A3-0.375', 'A4-
        0.08333333333333333', 'A4-0.125', 'A4-0.16666666666666666', 'A4-0.1875', 'A4-0.25', 'B3-0.375', '
        C4-0.08333333333333333', 'C4-0.125', 'C4-0.16666666666666666', 'C4-0.25', 'C4-0.375', 'C5-
        0.08333333333333333', 'C5-0.125', 'C5-0.25', 'C5-0.5', 'D3-0.125', 'D3-0.25', 'D4-
        0.08333333333333333', 'D4-0.125', 'D4-0.16666666666666666', 'D4-0.25', 'D4-0.375', 'D5-
        0.08333333333333333', 'D5-0.125', 'D5-0.25', 'Eb4-0.25', 'E3-0.25', 'E4-0.08333333333333333', 'E4-
        0.125', 'E4-0.16666666666666666', 'E4-0.25', 'E5-0.08333333333333333', 'E5-0.125', 'E5-0.25', 'E5-
        0.5', 'G3-0.125', 'G4-0.25']

    # Creating an empty dictionary
    SongDict = {}

    # Set assign the notes as dictionary keys:
    for note in specific_note_order:
        SongDict[note] = 0

    # Loop over the given note list again to give a value to the number of occurrences.
    for note in NoteList:
        snote = str(note)
#         print("note",note)
        if snote in SongDict:
            SongDict[snote] = SongDict[snote]+1

    print(SongDict)

    # Sort notes and frequencies
    Notes     = list(SongDict.keys())
    Frequency = [SongDict[note] for note in Notes]

    # creating the bar plot
    fig, ax = plt.subplots(figsize = (10, 5))
    bars = ax.bar(Notes, Frequency, color = "dimgray")

    for i in range(len(Notes)):
        ax.text(bars[i].get_x() + bars[i].get_width() / 2, bars[i].get_height(), str(Frequency[i]), ha='
            center', va='bottom')

    plt.xlabel("Specific Note Type Used")
    plt.ylabel("Frequency in Song")
    plt.title("Frequency of Specific Note Types in Don't Blame Me")
    plt.xticks(rotation = 88)
    plt.show()
    fig.savefig("Original DBM Specific Note Type Frequencies.png", bbox_inches='tight')


PlotSpecificNoteFrequencies(formatted_notes)

###### Length of all notes:

def PlotNoteLengthFrequencies(LengthList):

    # Order lengths
```

```
    length_order = ['0.08333333333333333', '0.125', '0.16666666666666666', '0.1875', '0.25', '0.375', '0.5
        ']

    # Creating an empty dictionary
    SongDict = {}

    # Set assign the rhythms as dictionary keys:
    for length in length_order:
        SongDict[length] = 0

    # Loop over the given rhythm list again to give a value to the number of occurrences.
    for length in LengthList:
        slength = str(length)
        #print("length",length)
        if slength in SongDict:
            SongDict[slength] = SongDict[slength]+1

    print(SongDict)

    # Sort notes and frequencies
    Notes     = list(SongDict.keys())
    Frequency = [SongDict[note] for note in Notes]

    # creating the bar plot
    fig, ax = plt.subplots(figsize = (10, 5))
    bars = ax.bar(Notes, Frequency, color = "dimgray")

    for i in range(len(Notes)):
        ax.text(bars[i].get_x() + bars[i].get_width() / 2, bars[i].get_height(), str(Frequency[i]), ha='
            center', va='bottom')

    plt.xlabel("Length of Note")
    plt.ylabel("Frequency in Song")
    plt.title("Frequency of Note Lengths in Don't Blame Me")
    plt.xticks(rotation = 45)
    plt.show()
    fig.savefig("Original DBM Length of Notes Frequencies.png", bbox_inches='tight')

PlotNoteLengthFrequencies(DBMnr.interval)
```

Listing 6.10. Analyzing Random "Don't Blame Me"

```
from musicpy import *
import random
import matplotlib.pyplot as plt

def ASCIItoNote(note):
    if note == 'rest1o32':
        return rest(1/32)
    if note == 'rest1o8':
        return rest(1/8)
    if note == 'rest1o4':
        return rest(1/4)
    if note == 'rest1o2':
        return rest(1/2)
    if note == 'rest4o4':
        return rest(4/4)
    if note == 'D31o8':
        return chord('D3', interval = [1/8])
    if note == 'D31o4':
        return chord('D3', interval = [1/4])
    if note == 'E31o4':
        return chord('E3', interval = [1/4])
    if note == 'G31o8':
        return chord('G3', interval = [1/8])
    if note == 'A33o16':
        return chord('A3', interval = [3/16])
    if note == 'A31o8':
        return chord('A3', interval = [1/8])
    if note == 'A31o4':
        return chord('A3', interval = [1/4])
    if note == 'A33o8':
        return chord('A3', interval = [3/8])
    if note == 'A31o12':
```

```
        return chord('A3', interval = [1/12])
    if note == 'B33o8':
        return chord('B3', interval = [3/8])
    if note == 'C41o8':
        return chord('C4', interval = [1/8])
    if note == 'C41o4':
        return chord('C4', interval = [1/4])
    if note == 'C43o8':
        return chord('C4', interval = [3/8])
    if note == 'C41o12':
        return chord('C4', interval = [1/12])
    if note == 'C41o6':
        return chord('C4', interval = [1/6])
    if note == 'D41o8':
        return chord('D4', interval = [1/8])
    if note == 'D41o4':
        return chord('D4', interval = [1/4])
    if note == 'D43o8':
        return chord('D4', interval = [3/8])
    if note == 'D41o12':
        return chord('D4', interval = [1/12])
    if note == 'D41o6':
        return chord('D4', interval = [1/6])
    if note == 'Eb41o4':
        return chord('Eb4', interval = [1/4])
    if note == 'E41o8':
        return chord('E4', interval = [1/8])
    if note == 'E41o4':
        return chord('E4', interval = [1/4])
    if note == 'E41o12':
        return chord('E4', interval = [1/12])
    if note == 'E41o6':
        return chord('E4', interval = [1/6])
    if note == 'G41o4':
        return chord('G4', interval = [1/4])
    if note == 'A43o16':
        return chord('A4', interval = [3/16])
    if note == 'A41o8':
        return chord('A4', interval = [1/8])
    if note == 'A41o4':
        return chord('A4', interval = [1/4])
    if note == 'A41o6':
        return chord('A4', interval = [1/6])
    if note == 'A41o12':
        return chord('A4', interval = [1/12])
    if note == 'C51o8':
        return chord('C5', interval = [1/8])
    if note == 'C51o4':
        return chord('C5', interval = [1/4])
    if note == 'C51o2':
        chord('C5', interval = [1/2])
    if note == 'C51o12':
        return chord('C5', interval = [1/12])
    if note == 'D51o8':
        return chord('D5', interval = [1/8])
    if note == 'D51o4':
        return chord('D5', interval = [1/4])
    if note == 'D51o12':
        return chord('D5', interval = [1/12])
    if note == 'E51o8':
        return chord('E5', interval = [1/8])
    if note == 'E51o4':
        return chord('E5', interval = [1/4])
    if note == 'E51o2':
        return chord('E5', interval = [1/2])
    if note == 'E51o12':
        return chord('E5', interval = [1/12])

# Transition Dictionary

#make the song into a sring and then a list
songstr = 'C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8+C41o8+A31o8+A31o8+A31o8+C41o8
+C41o8+C41o8+C41o8+C41o8+D41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o2+rest1o4+E41o4+D41o8
+C41o8+C41o8+D41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8
+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o2+rest4o4+D41o8+D43o8+rest1o2+D41o8+C41o8
+D41o8+C41o8+D41o4+Eb41o4+D41o8+D43o8+rest1o2+D41o8+C41o8+D41o8+C41o8+D41o8+C41o8+E41o4
```

```
+D41o8+D43o8+C41o8+C43o8+A31o8+A33o8+D31o8+D31o8+E31o4+rest4o4+rest4o4+D41o8+D43o8
+rest1o2+D41o8+C41o8+D41o8+C41o8+D41o4+Eb41o4+D41o8+D43o8+rest1o4+rest1o8+C41o8+D41o8
+C41o8+D41o8+C41o8+D41o8+C41o8+E41o4+D41o8+D43o8+C41o8+C43o8+A31o8+A33o8+D31o8+D31o8
+E31o4+rest4o4+rest1o2+D41o8+E41o8+D41o8+C41o8+A31o4+C41o4+A31o4+C41o4+A31o4+C41o4
+D41o8+E41o8+D41o4+A31o4+C41o4+A31o4+C41o4+A31o4+C41o4+D41o8+E41o8+D41o4+A31o4+C41o4
+A31o4+C41o4+A31o4+C41o4+D41o8+E41o8+D41o4+rest1o4+rest1o8+C41o8+D41o8+C41o8+D41o8
+C41o8+D41o4+D41o4+rest1o4+C41o8+A33o16+rest1o32+rest1o4+C41o4+A31o8+A31o4+C41o8
+C41o4+C41o8+C41o8+C41o8+A31o8+A31o8+A31o8+C41o8+C41o8+C41o8+C41o8+D41o8+A31o8+A31o8
+G31o8+A31o8+C43o8+rest1o2+rest1o4+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8
+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8
+rest1o2+rest1o4+C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8+C41o8+A31o8+A31o8+A31o8
+C41o8+C41o8+C41o8+C41o8+D41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o4+rest1o8+E41o8+E41o4
+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8
+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o2+D41o8+D43o8+rest1o2+D41o8+C41o8
+D41o8+C41o8+D41o8+C41o8+Eb41o4+D41o8+D43o8+rest1o2+D41o8+C41o8+D41o8+C41o8+D41o8+C41o8
+E41o4+D41o8+D43o8+C41o8+C43o8+A31o8+A31o4+A31o8+D31o4+E31o4+rest1o4+rest1o8+D41o8+D41o4
+E41o4+rest4o4+D41o8+D43o8+C41o8+B33o8+D41o8+C41o8+D41o8+C41o8+D41o8+C41o8+Eb41o4+D41o8
+D43o8+rest1o2+D41o8+C41o8+D41o8+C41o8+D41o8+C41o8+E41o4+D41o8+D41o8+D41o8+D41o8+D41o8
+C41o8+C41o4+C41o6+D41o6+E41o6+rest1o4+D51o8+E51o8+D51o4+C51o2+G41o4+rest1o8+C41o8+D41o8
+C41o8+D41o8+E41o8+D41o8+C41o8+A31o4+C41o4+A31o4+C41o4+A31o4+C41o4+D41o8+E41o8+D41o4+A31o4
+C41o4+A31o4+C41o4+A31o4+C41o4+D41o8+E41o8+D41o4+A31o4+C41o4+A31o4+C41o4+A31o4+C41o4+D41o8
+E41o8+D41o4+rest1o4+rest1o8+C41o8+D41o8+C41o8+D41o8+C41o8+D41o4+D41o4+rest1o4+C41o8+A33o16
+rest1o32+rest1o4+C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8+C41o8+A31o8+A31o8+A31o8+C41o8
+C41o8+C41o8+C41o8+D41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o2+rest1o4+E41o4+D41o8+C41o8
+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8
+A31o8+G31o8+A31o8+C43o8+rest1o2+rest1o4+C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8+C41o8
+A31o8+A31o8+A31o8+C41o8+C41o8+C41o8+C41o8+D41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o4
+rest1o8+E41o8+E41o4+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8
+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8+A41o6+A41o6+A41o6
+A41o4+rest1o4+C51o4+rest1o4+D41o12+E41o12+D41o12+D41o8+C41o8+D41o12+E41o12+D41o12+D41o8
+C41o8+D41o12+C41o12+A31o12+A31o4+rest1o4+rest1o8+C41o8+D41o12+C41o12+A31o12+A31o4+A41o6
+A41o6+A41o6+A41o4+rest1o4+C51o4+rest1o4+D41o12+E41o12+D41o12+D41o8+C41o8+D41o12+E41o12
+D41o12+D41o8+C41o8+D41o12+C41o12+A31o12+A31o4+rest1o4+rest1o8+C41o8+D41o12+C41o12+A31o12
+A31o4+rest1o2+rest4o4+D41o12+E41o12+D41o12+D41o8+C41o8+D41o12+E41o12+D41o12+D41o8+C41o8
+D41o12+C41o12+A31o12+A31o4+rest1o2+rest1o2+rest1o4+rest1o8+E41o8+E41o4+E41o4+D41o8+C41o8
+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8
+A31o8+G31o8+C51o8+C51o8+C51o8+D51o8+E51o8+C51o8+D51o4+C51o8+A43o16+rest1o32+rest1o4+C51o8
+D51o8+C51o8+E51o8+E51o4+C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8+C41o8+A31o8+A31o8+A31o8
+C41o8+C41o8+C41o8+C41o8+D41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o2+rest1o4+E41o4+D41o8
+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+A31o8+C41o8+A31o8+A31o8+G31o8+C41o8
+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o2+rest1o4+C41o4+A31o8+A31o4+C41o8+C41o4+C41o8+C41o8
+C41o8+A31o8+A31o8+A31o8+C41o8+C41o8+C41o8+D41o8+A31o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o4
+rest1o8+E41o8+E41o4+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8
+C41o8+A31o8+A31o8+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8+A41o6+A41o6+A41o6+A41o4+rest1o4
+E51o2+D51o12+E51o12+D51o12+D51o8+C51o8+D51o12+E51o12+D51o12+D51o8+C51o8+D51o12+C51o12+A41o12
+A41o4+rest1o4+rest1o8+C51o8+D51o12+C51o12+A41o12+A41o8+C51o8+rest1o4+rest1o8+E41o8+E41o4
+E41o4+D41o8+C41o8+C41o8+D41o8+D41o4+D41o8+E41o8+D41o8+C41o8+A31o8+G31o8+C41o8+A31o8+A31o8
+G31o8+C41o8+A31o8+A31o8+G31o8+A31o8+C43o8+rest1o2+rest4o4'

songList = songstr.split("+")
# print(songList)

song = songList

states = []
DBMD = {}

for notetype in song:
    if notetype not in states:
        states.append(notetype)
        DBMD[notetype] = []

for i, notetype in enumerate(song):
    #print("notetype",notetype,"i",i)
    if (i < len(song) - 1):
        DBMD[notetype].append(song[i + 1])

print(DBMD)

#Pick the random state to start from.
keyList = []

for i, key in enumerate(DBMD):
    keyList.append(key)
#       print("key", i, " = ", key)
#       print("        ", DBMD[key])
```

```
possible_options = keyList
# print(possible_options)

#Initialize a random song.
state = random.choice(possible_options)
newsong = ASCIItoNote(state)

#Make a random song.

for i in range(47):
    state = random.choice(DBMD[state])
    myNote = ASCIItoNote(state)
    print("state", i,':',  myNote)
    newsong += myNote
introchorus = newsong
newsong = 0  #Reset newsong for the next iteration

for i in range(66):
    state = random.choice(DBMD[state])
    myNote = ASCIItoNote(state)
    print("state", i,':',  myNote)
    newsong += myNote
verse1 = newsong
newsong = 0  #Reset newsong for the next iteration

for i in range(40):
    state = random.choice(DBMD[state])
    myNote = ASCIItoNote(state)
    print("state", i,':',  myNote)
    newsong += myNote
prechorus = newsong
newsong = 0  #Reset newsong for the next iteration

for i in range(96):
    state = random.choice(DBMD[state])
    myNote = ASCIItoNote(state)
    print("state", i,':',  myNote)
    newsong += myNote
chorus = newsong
newsong = 0  #Reset newsong for the next iteration

for i in range(80):
    state = random.choice(DBMD[state])
    myNote = ASCIItoNote(state)
    print("state", i,':',  myNote)
    newsong += myNote
verse2 = newsong
newsong = 0  #Reset newsong for the next iteration

for i in range(28):
    state = random.choice(DBMD[state])
    myNote = ASCIItoNote(state)
    print("state", i,':',  myNote)
    newsong += myNote
riff = newsong
newsong = 0  #Reset newsong for the next iteration

for i in range(19):
    state = random.choice(DBMD[state])
    myNote = ASCIItoNote(state)
    print("state", i,':',  myNote)
    newsong += myNote
solo = newsong
newsong = 0  #Reset newsong for the next iteration

for i in range(38):
    state = random.choice(DBMD[state])
    myNote = ASCIItoNote(state)
    print("state", i,':',  myNote)
    newsong += myNote
bridge = newsong
newsong = 0  #Reset newsong for the next iteration

for i in range(30):
    state = random.choice(DBMD[state])
```

```
    myNote = ASCIItoNote(state)
    print("state", i,':',  myNote)
    newsong += myNote
diffriff = newsong
newsong = 0  #Reset newsong for the next iteration

for i in range(24):
    state = random.choice(DBMD[state])
    myNote = ASCIItoNote(state)
    print("state", i,':',  myNote)
    newsong += myNote
outro = newsong
newsong = 0  #Reset newsong for the next iteration

DBMR = introchorus+ verse1 + prechorus + chorus + verse2 + prechorus + chorus + riff + riff + solo +
     bridge + chorus + diffriff + outro

# Print the length of the final random song
print(len(DBMR))

for i in range(len(DBMR)):
    print("state", [i] ,"note=", DBMR.notes[i], "interval=", DBMR.interval[i])

play(DBMR, bpm = 136)


#### Analysis of Random Song
###### Only the notes:

def PlotNoteFrequencies(NoteList):

    # Order notes
    note_order = ['A3', 'A4', 'B3', 'C4', 'C5', 'D3', 'D4', 'D5', 'Eb4', 'E3', 'E4', 'E5', 'G3', 'G4']

    # Creating an empty dictionary
    SongDict = {}

    # Set assign the notes as dictionary keys:
    for note in note_order:
        SongDict[note] = 0

    # Loop over the given note list again to give a value to the number of occurances.
    for note in NoteList:
        snote = str(note)
        #print("note",note)
        if snote in SongDict:
            SongDict[snote] = SongDict[snote]+1

    print(SongDict)

    # Sort notes and frequencies
    Notes     = list(SongDict.keys())
    Frequency = [SongDict[note] for note in Notes]

    # creating the bar plot
    fig, ax = plt.subplots(figsize = (10, 5))
    bars = ax.bar(Notes, Frequency, color = "silver")

    for i in range(len(Notes)):
        ax.text(bars[i].get_x() + bars[i].get_width() / 2, bars[i].get_height(), str(Frequency[i]), ha='
            center', va='bottom')

    plt.xlabel("Note Used")
    plt.ylabel("Frequency in Song")
    plt.title("Frequency of Notes in Newly Generated Don't Blame Me")
    plt.xticks(rotation = 45)
    plt.show()
    fig.savefig("Random DBM Note Frequencies.png", bbox_inches='tight')

PlotNoteFrequencies(DBMR.notes)

###### Notes and their respective length:

formatted_notes = ['-'.join([str(note), str(interval)]) for note, interval in zip(DBMR, DBMR.interval)]
# print(formatted_notes)
```

```
def PlotSpecificNoteFrequencies(NoteList):

    # Order specific notes
    specific_note_order = ['A3-0.08333333333333333', 'A3-0.125', 'A3-0.1875', 'A3-0.25', 'A3-0.375', 'A4-
        0.08333333333333333', 'A4-0.125', 'A4-0.16666666666666666', 'A4-0.1875', 'A4-0.25', 'B3-0.375', '
        C4-0.08333333333333333', 'C4-0.125', 'C4-0.16666666666666666', 'C4-0.25', 'C4-0.375', 'C5-
        0.08333333333333333', 'C5-0.125', 'C5-0.25', 'C5-0.5', 'D3-0.125', 'D3-0.25', 'D4-
        0.08333333333333333', 'D4-0.125', 'D4-0.16666666666666666', 'D4-0.25', 'D4-0.375', 'D5-
        0.08333333333333333', 'D5-0.125', 'D5-0.25', 'Eb4-0.25', 'E3-0.25', 'E4-0.08333333333333333', 'E4-
        0.125', 'E4-0.16666666666666666', 'E4-0.25', 'E5-0.08333333333333333', 'E5-0.125', 'E5-0.25', 'E5-
        0.5', 'G3-0.125', 'G4-0.25']

    # Creating an empty dictionary
    SongDict = {}

    # Set assign the notes as dictionary keys:
    for note in specific_note_order:
        SongDict[note] = 0

    # Loop over the given note list again to give a value to the number of occurances.
    for note in NoteList:
        snote = str(note)
#         print("note",note)
        if snote in SongDict:
            SongDict[snote] = SongDict[snote]+1

    print(SongDict)

    # Sort notes and frequencies
    Notes     = list(SongDict.keys())
    Frequency = [SongDict[note] for note in Notes]

    # creating the bar plot
    fig, ax = plt.subplots(figsize = (10, 5))
    bars = ax.bar(Notes, Frequency, color = "silver")

    for i in range(len(Notes)):
        ax.text(bars[i].get_x() + bars[i].get_width() / 2, bars[i].get_height(), str(Frequency[i]), ha='
            center', va='bottom')

    plt.xlabel("Specific-Note-Type-Used")
    plt.ylabel("Frequency-in-Song")
    plt.title("Frequency-of-Specific-Note-Types-in-Newly-Generated-Don't-Blame-Me")
    plt.xticks(rotation = 88)
    plt.show()
    fig.savefig("Random-DBM-Specific-Note-Type-Frequencies.png", bbox_inches='tight')

PlotSpecificNoteFrequencies(formatted_notes)

###### Length of all notes:

def PlotNoteLengthFrequencies(LengthList):

    # Order lenths
    length_order = ['0.08333333333333333', '0.125', '0.16666666666666666', '0.1875', '0.25', '0.375', '0.5
        ']

    # Creating an empty dictionary
    SongDict = {}

    # Set assign the rhythms as dictionary keys:
    for length in length_order:
        SongDict[length] = 0

    # Loop over the given rhythm list again to give a value to the number of occurances.
    for length in LengthList:
        slength = str(length)
        #print("length",length)
        if slength in SongDict:
            SongDict[slength] = SongDict[slength]+1

    print(SongDict)

    # Sort notes and frequencies
    Notes     = list(SongDict.keys())
    Frequency = [SongDict[note] for note in Notes]
```

```
    # creating the bar plot
    fig, ax = plt.subplots(figsize = (10, 5))
    bars = ax.bar(Notes, Frequency, color = "silver")

    for i in range(len(Notes)):
        ax.text(bars[i].get_x() + bars[i].get_width() / 2, bars[i].get_height(), str(Frequency[i]), ha='
            center', va='bottom')

    plt.xlabel("Length-of-Note")
    plt.ylabel("Frequency-in-Song")
    plt.title("Frequency-of-Note-Lengths-in-Newly-Generated-Don't-Blame-Me")
    plt.xticks(rotation = 45)
    plt.show()
    fig.savefig("Random-DBM-Length-of-Notes-Frequencies.png", bbox_inches='tight')

PlotNoteLengthFrequencies(DBMR.interval)
```