– Independent Work Report Spring, 2021 –

Music Notation Analysis Through Music21

Justin Bi ’22
Adviser: Brian Kernighan

Abstract

As computer technologies and algorithms improve, the music field continues to leverage computer science in their research. However, much of the intersect between music and computer science focuses more on music’s auditory aspect than its theory aspect. This project proposes an alternative method of using computers to analyze specifically music notation both by demonstrating the plausibility of computer assisted analysis and by creating analysis tools using the music21 Python library.

1. Introduction

With the field of computer science constantly expanding, many other fields of research have adopted and utilized computers to help improve their own unique goals. These range from smaller scale tools like word counters for authors and image editing software for artists to large scale operations such as user data analysis for advertisers and diagnosis programs for doctors. Music is another such field that benefits from computer science, with programs helping composers write pieces in notation software or performers record and edit in digital audio workstations. But one area of music where computer science has not been too impactful is music analysis.

Music theory research is still largely done without computers. Of course, musicologists might use computers to download files or look up background knowledge, but generally, if a music researcher has a question about a score, the most common method is not to boot up a program or application, but rather to analyze the music manually.

It is not as if music notation cannot be effectively digitized. Methods for digitally storing music files have been around for decades [1], and analysis programs on these file formats have also existed for years, as will be detailed in the background portion of this paper. And even if computers cannot

decipher the exact intricacies of a complex humanities field like music, they would still be extremely helpful aids, as they could do laborious tasks such as finding and counting note occurrences in a piece more quickly than humans. So why does there exist a dearth of computer usage in music analysis?

There are two plausible reasons. First, a lack of publicity. When it comes to composition or audio editing, there are many applications which are relatively well known. Programs like MuseScore, Finale, Audacity, and Reaper are all popular and useful applications that help make these music tasks simpler and more efficient, and they are very likely to be recognized by members of the music community. Meanwhile, code libraries like music21 that aid music analysis do exist [2], but they certainly are not as well known to the general public.

The second reason is a lack of availability. The aforementioned popular programs are all designed to be easy to use. While some of the apps might have a bit of a learning curve, at the very least none of them require users to learn a completely different skill before learning how to use the application itself. Meanwhile, applications that exist for music analysis are not apps, but rather Python libraries, meaning if someone wanted to use this library, they would have to learn Python before even getting to the music analysis portion. And though there may exist musicologists who know Python or have the time to learn it, the average person may not be willing to spend their time learning both Python and the library.

This project tackles both of the above problems which may deter musicologists from using computers in music analysis. One of this paper's goals is demonstrating the prospect of using computers to push the boundaries of music analysis by leveraging music21. The other goal is to make some of this library's features available online for others to use in a manner which abstracts away the code complexities. Accomplishing these goals would help demonstrate to more people the benefits of computer music analysis as well as provide them with accessible methods to do so.

2. Background and Related Work

2.1. Music Notation Refresher

While the more complex musical aspects in this paper will be abstracted away, some rudimentary music knowledge will be explained in this subsection for those who are unfamiliar with it. Readers who are comfortable with music basics can skip to section 2.2.

At its core, music notation transcribes the audio we hear into notes that we can read, similar to how a stenographer would record the conversations we hear into words that we read. Everything in notation form conveys some sort of meaning which allows musicians to recreate the audio of a piece just by reading sheet music. Take Figure 1 below.

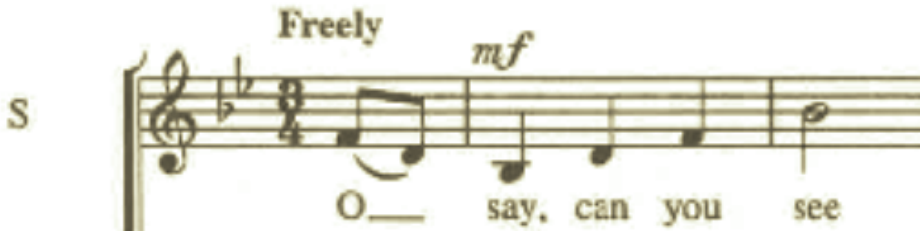

Figure 1: The Star Spangled Banner in Notation Form [3]

The above shows the beginning words of the national anthem as it would appear on sheet music. For this paper, the only two things that need to be understood are pitch and duration, so we will examine these two items in the context of figure 1.

The pitch of a note determines how high or low it should be sung or played. For example, the notes above the words "can you see" are rising, just like how each successive lyric should be sung higher than the next. For our purposes, we say there are unique 12 pitches, denoted by the letters A to G, sometimes with a sharp (#) or flat (b) symbol appended to it. As an example, the score above would have the pitches: F D Bb D F Bb.

The duration of the notes dictates how long they should be played. In the lyric "can you see", the words "can" and "you" are sung for a shorter time than "see", and thus are denoted with quarter notes while "see" is denoted with a half note. There are many durations a note can have, but a good rule of thumb is that the smaller the fraction, the shorter the length. For example, an eighth note

would be played for less time than a quarter, and both of these durations are shorter than a half or whole note.

2.2. Music Notation Analysis

While music analysis with computers is not the most popular thing in the musical field, people have certainly done studies combining the two. What is interesting, however, is that these studies chose to use music audio files over music notation files. In contrast, the majority of non-computer music analysis is done by looking over scores. Why this difference in approach?

One likely explanation is that many computation techniques base their analysis off of data statistics, thus requiring lots of data that can be easily used as program inputs. If one only compares the availability between notation data and audio data, audio seems the clear choice, as while it's hard to come by notation datasets which are both sizeable and well documented, there is no shortage of music audio datasets, with possible sources ranging from audio databases to Spotify.

However, there is a reason why many musicologists study notation, and that is because it is the easiest way to learn about the theory behind the music. Examining a score provides all the information in one static and controlled environment. If someone wanted to figure out exactly what rhythm some part plays in a given measure range, they could easily search for and locate the relevant passage in the score. On the other hand, with just an audio file, one would first have to listen through the piece until the relevant portion came into question, and then try and listen for that specific part under potentially many other instruments, each playing their own interweaving part. While hearing the notes played aloud might help people get a feel for the music, when it comes down to specific details, the score is simply much easier to use.

Of course, the point is not to figure out what humans find easy, but rather what computers find easy. But aside from the difference in data availability, using music notation data seems to be the better choice, for similar reasons as to why humans find it better. Not only do you get cleaner, more exact data (recorded audio runs the possibility of having more human errors like background noise, tuning errors, incorrect speeds, wrong notes, etc, while a digital score would consist of purely

discrete values), but you can also discern between notes that sound the same but mean different things. As an example, the notes Bb and A# have the same pitch when played aloud, but they often have different meanings. If this pitch appears as the 7th of a C7 chord, then the 7th often resolves a step downwards, and will almost always be written as Bb. On the other hand, if this pitch appears as the third of a V chord, then this is most likely the key's leading tone which resolves a step upwards, and will almost always be written as A#. If a computer only listened to the audio, it would hear the same pitch but not see their different labels, meaning it would have a harder time discerning the two functions of the same pitch.

And while there exist software tools to represent and analyze audio waves, the same can be said for music notation. There are a variety of notation file formats available across the web, including MusicXML, MIDI, MUS, and more. Additionally, the Python library music21 can help standardize a variety of formats into an intuitive object oriented format (more on this in section 2.3). Indeed, it is actually the case that computers can very easily operate on notation data, as evidenced by the entire library of functionality built around doing exactly that.

Of course, the lack of good notation datasets remains an obstacle. However, this does not detract from the advantages of analyzing specifically notation files, and while there may not be too many notation datasets currently, as tools which transform music into notation formats improve, such datasets will become easier to build. We hope that by showing the possibilities of computer analysis on notation files, it will encourage researchers to compile more notation datasets.

2.3. Music Analysis Tools and Their Availability

The other part of this project involves developing tools for others to use in order to further promote this approach to computer music analysis.

As far as analysis tools go, the aforementioned music21 library is a great option. Not only does it provide a consistent, programmer friendly way to represent all types of music objects from notes to chords to scores, it also provides intuitive methods to access relevant information, which you can then use to build your own functions.

But even though music21 is useful, it does have quite a bit of a learning curve and can require a large time investment to truly learn. Even with a background in Python, it took around half of a semester to learn all the required information for this paper's goals, which focus on just a fraction of music21's total functionality. Things would be even more difficult for people who have no background in programming to begin with. If someone with no coding experience wanted to take advantage of music21's features, it would take quite a bit of time to familiarize themselves with both Python as well as the library, and while programming is a useful skill to learn, it is completely reasonable to want to skip the time consuming portions and access the features immediately. This led to the question, were there tools like music21 available online, or in otherwise accessible manners? Perhaps there would not be tools as in-depth as music21, but maybe there would exist a few simple functions that people could readily use.

Surprisingly, the web is almost completely devoid of tools that help analyze notation files. Take a relatively simple task, like counting the number of C's in a score. It is not insurmountable to manually scan through the every part in the sheet music and tally the occurrences, but if you had a MIDI file of the score, then the information you want is already stored in there and in an easily workable format. If there was a website that had the tools to execute such a request, then this problem is shortened and simplified by a significant factor. Unfortunately, such an application on the internet is nowhere to be found.

This lack of tools seems odd given the circumstances. There are a wide variety of tasks which musicologists might want to do, but would be tedious to do by hand. There exist formats and tools that can be used to greatly simplify said tasks. Yet, no one seems to have made any of these tools more accessible, an action that would benefit many researchers.

Here is where this project steps in. By providing music analysis functions online in a manner that is intuitive for people unfamiliar with computer programming, these tools would become more widely available and accessible for everyone.

3. Approach

The approach for this project consists of four main parts: learning music21, finding appropriate data, processing the data for analysis, and deploying functionality online.

3.1. Music21

Music21 was very much a keystone in this project, in the sense that it is a well maintained library that already has helpful features which are ready to be used. One of the key things about the library is the ability to easily find music objects and to extract information about said objects. Most of this project's work is focused on notes and chords, both of which contain a variety of data to extract, including pitch, duration, name, and more. These are all useful and fundamental traits for any music analysis in general, so it is essential to be able to access these. Since music21 simplified collecting such data, it was clear that using music21 would help make the analysis much more efficient.

3.2. Dataset

As mentioned prior, notation datasets are not the most prevalent on the internet. There were a few options as to what data to use for this project, and each of them had their advantages and drawbacks.

The first possibility was using music21's own corpus, which can be found online [2]. The works present in this corpus are neatly organized and documented, and there is a sufficient amount of data that can be used for analysis. The trouble is that the scope of this corpus is a bit limited. The only prevalent genres are folk and pre 18th century classical; if you wanted to analyze jazz, rock, game soundtracks, or even just lesser known composers from the classical era, then music21's corpus likely does not have what you are looking for.

Another option is to collect data from online. Many internet users will independently upload music files for a variety of pieces, and you could form your own dataset composed of these files by collecting them into one place. Certainly, the expanse of the internet will have a lot more variety to work with when compared to music21's much smaller corpus. The problem is that the data on the internet is prone to being messy and unstructured. Unlike music21, there does not exist a singular

professional group that is making and uploading all these files; instead, there are often just amateur individuals, each uploading files with their own distinct notation styles. What this means is that the data from the internet has a higher chance of being inconsistent or faulty.

One last possibility is to simply create all the music files yourself. This elicits the greatest amount of control for your dataset; you can dictate exactly what is in it, and if a piece is not readily available anywhere online, you can always make it yourself. Additionally, you hold the standard for how pieces should be notated, so the notation style can be exactly how you want it to be. The downside is that this would be a monumental amount of work compared to the other two options. There is a reason why notation datasets are not more common, and it is because at the moment, the technology is not there to efficiently and easily automate this process, meaning a lot of time consuming manual transcription.

Of these options, this project went with using music21's corpus, specifically the Palestrina dataset. Not only is this approach simpler than the alternatives, but Palestrina also composed lots of wonderful pieces, and there are many potentially intriguing things to explore in his compositions.

3.3. Analysis

With the data acquired, we can now decide how we want to analyze it. To give some background, the Palestrina dataset consists of over 100 church masses composed by Palestrina, dating from around the mid to late 1500s [4]. Palestrina was a major part of this musical era, and his counterpoint music would lay the groundwork for much of the future's musical innovation. Besides following a basic mass structure, the pieces in this dataset do not seem to have any clear patterns or groupings, not from a quick glance over the scores nor from online research. When trying to find potential patterns in a dataset, one very common idea is to use a clustering algorithm. Thus, the general goal for this dataset was to collect the relevant data features and run it through kmeans++ clustering to analyze the results.

3.4. Deployment

As for making some analysis tools available for the general public to use, the best approach was to create a website. Deploying online means people could access the functions from any device that could connect to the internet, allowing for more availability. By abstracting away the more intricate coding aspects, people would only need to do simplistic tasks to use the tools, making it both easier and quicker to use music21's features.

Additionally, because this website is meant to be viewed by anyone, care should be put in to make things look interesting. While raw data is useful, having just tables of numbers might not make for the most appealing or informative experience. Thus, the idea was had to use the D3.js library to plot data in an attractive and interactive manner [5]. This serves not only to make the data easier to digest, but it also allows some degree of user control, meaning people can play around with the data and form the visualization that best suits their needs.

4. Implementation

4.1. Organizing and Collecting Palestrina Score Data

While the Palestrina dataset is neatly written, its pieces are structured not as masses, but rather as the individual movements of the masses. In short, a mass usually has around four to six individual movements, which together form the actual mass. Having the music stored as movements would be okay if the goal was to analyze the movements, but this particular project is focused on the masses; thus, the dataset must first be reorganized.

Fortunately, many music21 scores have a metadata object associated with it, which stores information like the composer, the title of the piece, and more. With this dataset, each movement has its parent mass title in its metadata object's title field, which saves us the trouble of having to manually sort the scores together. Note that for our purposes, the order of the internal movements is not critical, though other projects using this dataset may wish to find the proper way to order these files.

With the mass data correctly structured, now we must decide what aspects of the scores should serve as input for the clustering algorithm. Here, any interesting feature would suffice, including things like the pitch distribution, the presence of certain chords, or even Parsons code to examine melodic structure. This project ended up focusing on rhythmic distribution. In essence, the data to be extracted would tally how many of each rhythmic duration (eighth note, half note, etc) existed within each mass. The aim was to see whether there was perhaps some sort of pattern within Palestrina's compositions related to rhythmic distribution. Perhaps certain masses are more rhythmically active with many short duration notes, while others are more relaxed with long whole notes. Or maybe some are more rhythmically static, being largely comprised of one type of duration, while others are more varied with rhythms across the spectrum.

To obtain this specific information, we need to extract note durations from each piece. For that to happen, deeper understanding of how music21 stores musical data is necessary. One of the core classes in music21 is the stream class. A stream itself is not attached to a specific musical concept, but instead is similar to a nestable container. For example, a score containing all the music in an entire piece would be a stream. Each of the parts within the score would store its information in a stream as well. Each measure in each part is also a stream. This is how music21 stores scores, and it provides users with a musical hierarchy that stores all the relevant information in the piece. For this project, we need to be able to access all the notes in the piece, so given this structure we would need to access each part of every movement.

Note that if you tried to iterate through each part stream to get the notes, you would find measure streams instead. While one could theoretically just iterate within each measure stream to get access to the notes, this is a bit tedious, and there exists a simpler method, which is to leverage the `.flat` property of a stream object. In essence, this is a flattened version of the stream which removes all stream classes within it and compiles the remaining musical objects into a single stream. Not only is easier to iterate over a cleaner stream, but this also helps solve a problem with musical ties detailed below.

When a music note extends over a bar line, composers have to write this as a tied note, which

indicates that the two notes are connected even though they are separated by a measure line. For example, if a half note started on beat four of a measure that was four quarter notes long, then this would be written as a quarter note on beat four tied to a quarter note on beat one of the next measure. Here lies the potential snag in our program. Music21 stores tied notes as if they were separate notes, just with a tie between them. In other words, iterating over a stream that contained a tie like the one in our example would incorrectly count two notes of quarter note duration instead of correctly counting one note of half note duration.

Fortunately, there exists the function `stripTies()`, which when called on a stream will automatically remove all such tied notes and replace them with a single note of their combined duration, including edge cases such as notes with multiple ties. Note that this was another reason why we needed the part flattening described a paragraph prior, as without the flattened stream, `stripTies()` would only strip the ties within a single measure, whereas we specifically want it to strip all ties, including those that span outside of a single measure.

Thus, with this final modification, the score is in a desirable format. Simply access each part stream of a score, flatten its contents, remove the ties, and then get the duration of each object. To collect the duration, we first make sure that the item that we are accessing is either a note or a chord by checking its variable type, as streams can contain objects such as key signatures or tempo markers which are of no concern to us and should be ignored. Further, though a rest is a musical object, we specifically want note rhythms, so rest objects are ignored as well. Once it is ascertained that an item is either a note or the chord, we can access the duration property, which is equivalent to its rhythm. This information is stored in a Counter object for each score, and thus the data from every score has been collected.

With the rhythm information collected, there is just one more thing to take care of before we can begin clustering. One requirement for kmeans++ clustering is that the input length for each data point must be the same across all points. Since our data for each score was collected using Python's counter object, not every score's counter object has all the possible rhythms in the entire dataset. For example, it is possible that not every score would contain 16th notes. What this means is that

the input length for each point has the potential to be of varying length. Thus, to make sure each input is of the same length, we restructure the counter objects into lists of equal length that count each of the seen rhythms during data collection, which ensures the input size is the same across all points.

4.2. Clustering Results

With the data in this collected and processed, we can begin using our clustering algorithm. First, we use the elbow method to determine how many clusters to use, picking the number of clusters around the bend of the curve. In this case, the best number of clusters worked out to be four. After running the algorithm with this number of clusters, we use a principal component analysis (PCA) function to project the multi-dimensional data we collected from the scores into 2-dimensions, in order to visualize the results better. Both the PCA and kmeans methods were sourced from the sklearn library [6].

With the PCA and kmeans++ results, we can plot the data to visualize our clusters. By using the labels obtained from our clustering algorithm, we can filter out exactly which of our PCA indices belong to which class, since the indices of both align to the input data we fed them. The plotted results are shown in figure 2.

These reduced clusters are relatively good. Perhaps the rightmost cluster has too few points, maybe some of these points can be seen as outliers, but in general the clusters are visually clear and distinct. When plotted like this, the data does not really tell much besides the fact that there may be some relation between the mass data, but we can use this as a starting point to find out more about these scores.

One idea is to see whether these clusters correspond with the date at which they were published. Palestrina composed for almost half a century, and his pieces were published in batches over that time period [4]. While not all the pieces are dated, enough are such that we can still gain some insight by plotting the same data colored by year instead of by class.

One thing to note is that the Palestrina dataset does not contain years attached to each mass, nor

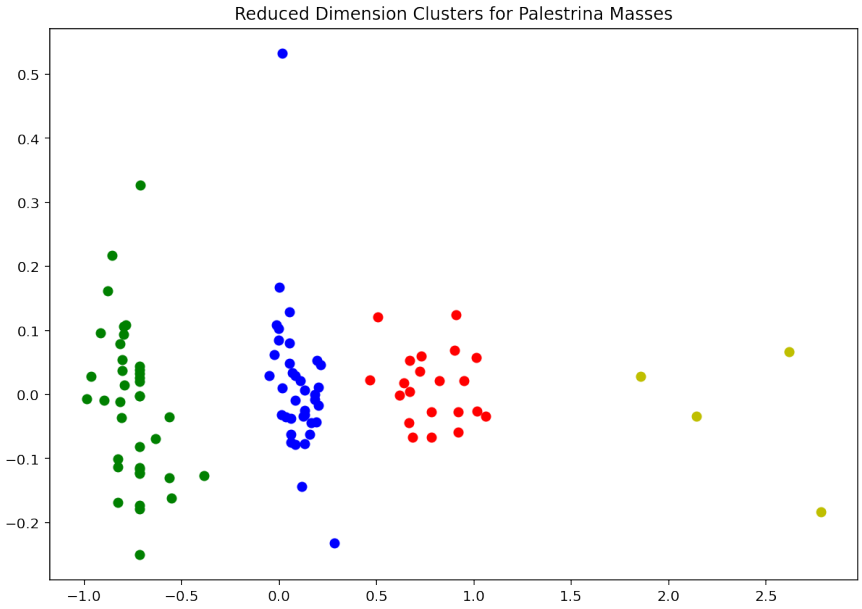

Figure 2: Palestrina clusters

to each movement. Thus, we used the publication dates from Wikipedia and manually attached them to each score, which, while a bit repetitive, was doable given that there were only about 100 entries to go through.

With the years attached to the pieces, we can now color each data point by publication date. A diverging color map was used, where earlier years are colored blue and later years are colored red, with the years spanning from 1554 to 1601. If a score does not have an attached date, it was represented as a black X instead of a colored dot. This makes it easy to see which scores were published around which time. The results are shown in figure 3.

From this, there emerges a potential pattern in the clusters. While the cluster on the left to have the largest mix between early and late dates, the other three clusters are more skewed more towards later years. In fact, when calculating the average years for the clusters, we see that going from left to right, the average years for the clusters are 1583, 1588, 1594, and 1597, which supports the above observation.

Another interesting detail is the location of the pieces with unknown publication dates, the ones marked by the black X symbols. Most of them tend to congregate in the middle two clusters, which together contain 20 of the 22 undated scores. Them being clustered together may indicate that these

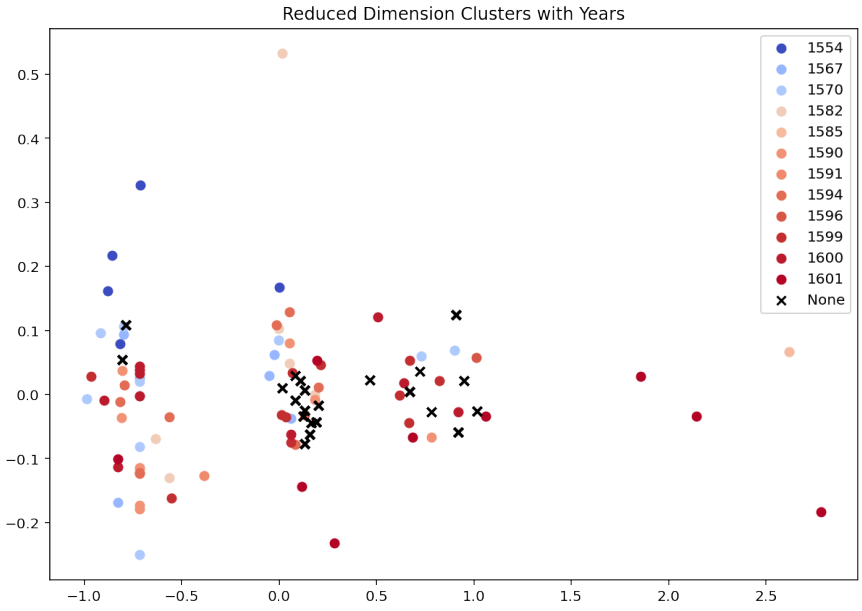

Figure 3: Palestrina clusters by year

pieces are similar in style to the other pieces in this cluster, which could be due to the fact that they were potentially composed in a similar time range. In essence, because many of these undated pieces are clustered with many pieces composed during later years, perhaps these undated pieces were also composed during later years.

Of course, this is just a hypothesis, and more testing would be needed to find support either for or against this specific proposition. However, the main takeaway from this is that the full process of music analysis was completed from end to end, leveraging computer software techniques to help simplify and accelerate the process. This illustrates how computer-aided analysis is both possible and promising.

4.3. Deploying the Website

After the analysis portion was completed, the focus of the project shifted to deploying analysis tools on a website. This goal mainly consisted of setting up the basic framework, designing the functions to publish, and displaying the data in an appealing manner.

4.3.1. Website Framework

While the basic website structure does not need to be too complex, at the bare minimum it needs to handle file uploads and look presentable. The presentation was relatively simple, as this was

largely resolved by using a Bootstrap template from online [7]. Handling the file uploads was a bit more involved.

With any website, one must be careful when allowing users to upload data. Specifically with music21, if the uploaded file is not an accepted music notation file, then music21 will not be able to parse the file correctly. While music21 does accept a variety of notation files, for now only MIDI files are accepted, as they are both widely available and simple to handle. Additionally, most MIDI files are very small, with the vast majority being under 1MB, meaning less concern about the amount of necessary storage. Nonetheless, it is always possible to create extremely large MIDI files (some can go up to terabytes in size), so a file size limit of 1 MB was implemented.

The next step was to manage the files once they were uploaded. This led to a design decision: should these files persist forever in a database, or should they be deleted after a certain amount of time? Allowing files to stay up forever would mean users could just revisit the same link to view the results again, rather than reupload the file. On the other hand, deleting the files after a while would reduce the need for a dedicated database, and data storage would be less of an issue.

In the end, it was decided that the website would only store the files for a limited amount of time. To do this, while a user session is active, the file will remain. Then, once the session is no longer active, the file will be deleted after an hour has passed.

4.3.2. Website Functionality

Because of the wide variety of things music21 can do, it would be nearly impossible to put all of its features online. Instead, the website focused on having a smaller number of well implemented functionalities.

The first function was a piano roll visualization. To give a brief definition, a piano roll displays all the notes on the screen, not as musical symbols but rather as blocks. The horizontal offset indicates when a note starts and ends, and the vertical offset indicates the pitch of the note. One optional feature that often makes the piano roll cleaner is to color each note based on which instrument plays it.

Given the general features of a piano roll, the data collection for this function was as follows.

For each note in each part, we need to collect the pitch, start position, and end position of the note. We also need to collect the part names for labelling and coloring purposes. Music21 always parses its incoming files into a hierarchy of streams, meaning we can repurpose the code used to collect rhythm information for the clustering algorithm and have it collect piano roll information instead. What remains is to collect the name of each part, which requires a bit more delicacy.

Because the user can write anything in their uploaded score, we do not know what the part names in the score are going to be. In fact, the score might not even provide valid part names. Since we would like to color code and label our piano roll, we need to make sure that whatever the contents of the uploaded piece, we can still provide valid names to each part.

To do this, we first make use of the `bestName()` function from music21. What this does is examine the relevant metadata information, including provided name, instrument, etc, to determine the best label for this instrument. However, this function still throws an exception when encountering a nameless part. In such a case, since we do not have any other information to go on, the part is simply named unnamed_part_1, with incrementing numbers should the need arise. With these two additions, the function is now able to correctly collect the piano roll information for each part and color the blocks accordingly.

The other function that was created was a pitch class visualizer. Recall from section 2.1 that music is made up of various pitches, like C, A#, Eb, etc. While notes are a combination of pitches and octaves, when separating into pitch class, we focus only on the pitch aspect. For our purposes, we say there are 12 unique pitch classes that notes can fall into, which is the general number of accepted pitches.

However, there exists a problem when dealing with these pitch classes, which is how to handle pitches that are named differently but sound the same. For example, we previously touched on how even though Bb and A# sound the same, their different labels often convey different meanings. Given this, what should a Bb or an A# be counted as when determining its class?

In the end, the choice was made to include both labels as separate classes. By using this type of structure, we do not lose the nuance between the differently labeled notes, and users can easily sum

the counts of both labels if they truly only want to focus on the pitch frequency. Though this seems like we include the same pitch twice, each pitch is only counted for the label it matches, meaning no note is ever double counted. Note that this only applies to pitches with exactly one accidental and whose counterpart also has exactly one accidental. For example, the pair Eb and D# would be included, but not the pair Eb and Fbb or E# and F.

Similar to the piano roll function, we iterate through the streams to collect the pitch name data, except this time we are not concerned about the individual parts. Note that music21 has its own way of storing flats, often replacing the flat symbol with a minus (for example, Bb would become B-). Because when this data is eventually presented we want to represent this with the flat symbol rather than a minus, we use the unicode name feature to extract the unicode version of these pitch names instead of music21's version.

This is how the data is collected for each of the two implemented functions. After the collection is complete, the data is formatted into CSV files, which are then stored inside a folder specific to the requesting user. These files will then be used for the visualization portion described below.

4.3.3. Website Visualization

With the data collected, now the task is to present this data in an interesting way. To do this, we make use of D3.js, which gives us a wide variety of ways to both plot data and interact with said plots.

For the piano roll, we begin by simply having the notes appear in the correct position based on their pitch, offset, and duration. Much of the base idea made use of code from an example project available online [8]. By providing a CSV with the relevant data, one could plot the elements by using D3.js's functions.

While this baseline was able to display a piano roll, at the moment the visualization was extremely static, and upon testing the functionality, it was quickly discovered that there were many improvements that could be made.

The first improvement concerned longer scores. Some of the longer pieces simply contained too many notes to fit on the screen at once, resulting in the data overflowing off the page. Scaling

down the width or the height of the piano roll contents would not suffice, as this could lead to scores looking extremely cramped as we try to cram hundreds of measures onto the screen at once.

The solution was to allow users more control when viewing the data. First, the piano roll was edited such that users could drag the piano roll around. This way, blocks would appear at a reasonable width and height, and people would still be able to access all the information by dragging to whatever part of the piece they want. It would also be useful to focus in on a specific section or view more of the score at a time, so a zoom feature was implemented as well. Finally, sometimes extremely fast rhythms like 16th notes or extremely long duration notes like tied whole notes might appear a bit too thin or wide to work with. Thus, sliders were implemented to adjust the scale of the block dimensions. Should a block be, say, too thin to easily see, one could increase the block scale to view larger blocks without affecting the correct relative placement of each block.

The next issue was that while the piano roll provides axes denoting the pitch number of each note, this was lacking in two regards. First, because the collected pitch data was stored as MIDI pitch numbers (which was necessary to get each block's correct y-axis placement), it is not immediately obvious which pitch each block corresponds to, as it might take a while to realize that, say, a pitch number of 60 corresponds to middle C. Second, once you drag the piano roll far enough right, the y-axis disappears from the screen, meaning you would have to drag back left to see the pitch number of a block.

To solve this problem, the visualization was edited to display the pitch information in its text format when hovering over a block. This solves both issues, as now you would not need to translate a number to a pitch, and you do not have to drag back just to find the pitch of a note. Note that this was specifically implemented as a hover tooltip, as this allows the user to see the data only when they want to and not have it remain cluttered on the screen all the time.

Speaking of clutter, there was one final feature that was added, which was a way to hide certain parts in the score. Take an example piece where multiple parts inhabited the same general pitch area at similar times. Even though each block is labeled by color, having so many blocks all in one single space may still make it hard to decipher exactly what is going on with each part.

Thus, the ability to hide certain parts in the score was added. As each block was tagged to enable color coding the notes, it was straightforward to hide these notes on a button trigger. Now, users can clean up the piano roll and only view the parts they deem necessary by clicking the names of the parts they wish to hide. Should they ever want to make the hidden parts visible again, they can simply press the name again.

These are the additional features the piano roll was fleshed out with. A visual of the finished piano roll function with sample data is shown in figure 4.

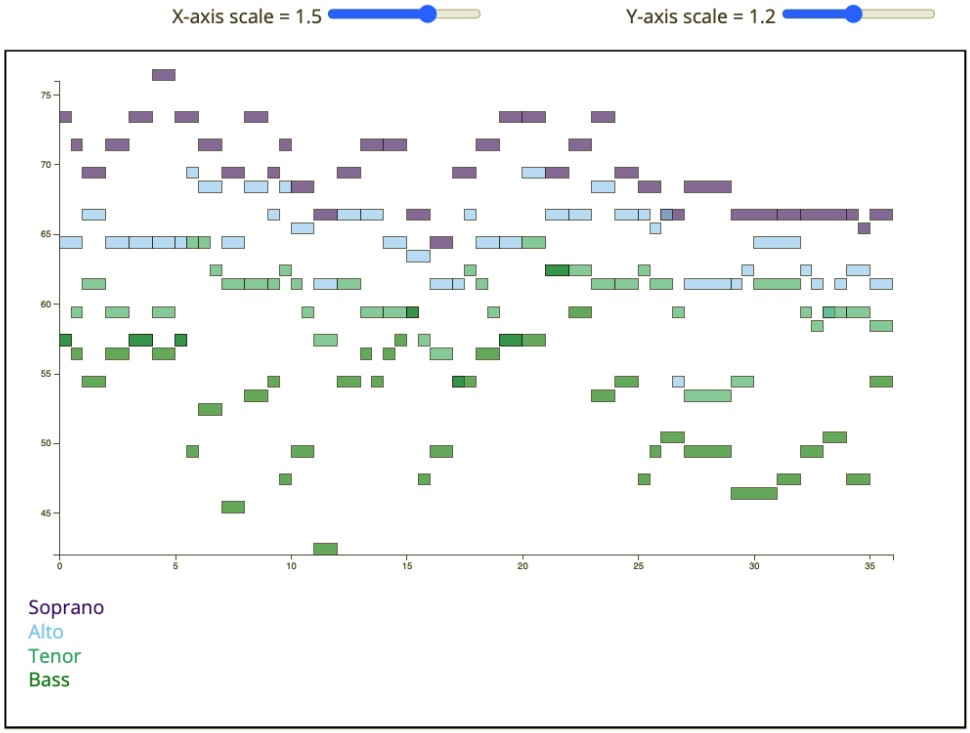

Figure 4: An example piano roll visualization

The other function to be visualized was the pitch class bar graph. This function was simpler to design, as at its core it is just a common bar graph, unlike the more complicated piano roll described above. Nonetheless, there were still some helpful additional features to add.

First was the ability to see the exact counts of each bar. Often in static bar charts, one would be able to get the general picture about the data distribution, but have a much more difficult trying to get the exact count of a bar. So in this graph, we added the ability to get the exact count of a pitch class by hovering over its bar.

Specifically regarding music, it would also be helpful to see the pitch distribution for each individual part instead of only for the whole score. For example, one might be curious to see if the bass part in an SATB score sings more root notes than the other parts, which would require examining the distribution for each part. Thus, the option to change the bar graph and only view the notes in a specific part was implemented.

With the above addition came another question: how should the y-axis act when changing the display option between parts? Keeping it constant did not seem like the best choice, since individual parts naturally have fewer notes than the score, and if the part counts are too low in comparison to the score counts, the bar heights could become too small to work with. This led to the initial idea of scaling the y-axis to whichever part was being viewed. While it solved the problem of the bars being too short, this presented another potential problem, in the sense that if you wanted to visually compare the note count between two parts, the scaling axis makes this more difficult. For example, a bass part and a tenor part might contain the same number of Bb's, but if the axis is different for each of them, the different bar heights might incorrectly convey that the counts are different. Given all these factors, the end product was designed to let users control the y-axis themselves. Should the bars be too small for their liking, they can adjust the y-axis scale. Should they want to keep the scale constant to compare between parts, they can do this as well. This grants more control to the user, allowing them to get more use out of the tool.

This describes the full functionality of the pitch class tool. An example instance is shown in figure 5.

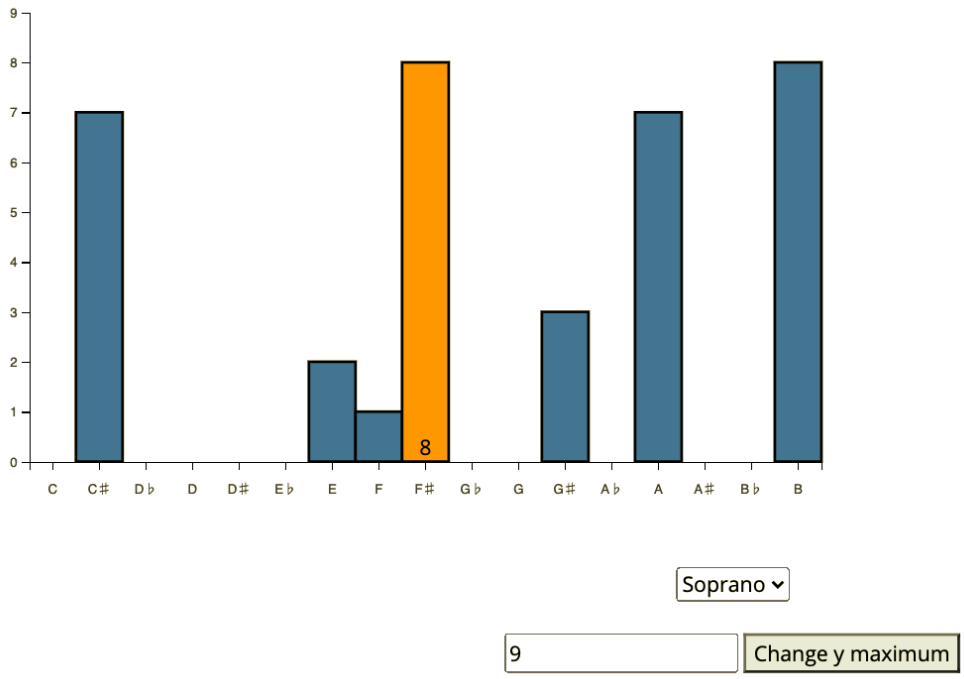

Figure 5: An example pitch class visualization

5. Evaluation

The goal at the beginning of the semester was to just do an analysis project. This goal was effectively completed, as using music21 and the Palestrina datasets, we were able to produce interesting results by using kmeans++ clustering, demonstrating the ability and benefits of using computer science algorithms on music notation data.

Later on, an additional goal was added: creating a website to make music analysis tools more accessible. This goal was also accomplished, and the website is available at https://music21-online.herokuapp.com/ for the public to use. Feedback given by peers and colleagues was used to improve the website's features and designs, and a form is available on the website for future comments and suggestions.

6. Conclusions and Future Work

This project was able to demonstrate the possibility and usefulness of doing music notation analysis using computer science. It demonstrated the basics of music21 and showed how to collect and use

the data for a clustering algorithm, and then used the results to formulate possible hypotheses about the Palestrina dataset, spurring further analysis.

Midway through the semester, an additional goal was added to create an intuitive website for those unfamiliar with computer science to use. The website allows users to upload their own music files and view data about them in an engaging way.

There are a variety of ways one can continue this work. The first and possibly most important option is to organize datasets or create efficient ways to collect data. As mentioned in the background section, there do not exist many well curated notation datasets, meaning there is not that much data that researchers can download and immediately use. Whether this means manually cleaning notation files or improving music transcription methods, creating better datasets would be an immense help.

Another idea would be to use music21 for more analysis work. Future steps for this project in particular would be to use different features for clustering, or to perhaps try a different algorithm. More generally, there are a wide variety of paths one could take. Even just in the machine learning area, there are many other interesting experiments one could try, such as music generation or classification. The methods of collecting data from music files has been detailed in the implementation section, and once the information has been collected, a large variety of programs and algorithms would be able to operate on it.

One final area of improvement is to make these kinds of tools more accessible. While the website provides some basic functions online, more can always be provided, which would allow those who do not have much coding experience to still take advantage of programming libraries for their own work. Whether this means expanding more on this specific project's website or creating other applications, making tools and knowledge more accessible would always be a helpful contribution.

7. Acknowledgements

I would like to thank Professor Brian Kernighan, Professor Zoe LeBlanc, and Vivien Nguyen for their continuous guidance throughout the semester. I would also like to thank my classmates in the IW06 seminar for providing helpful feedback and tips about my topic. Additionally, I would

like to thank my friends for their peer review, in particular Suava Sanjay. Lastly, I would like to thank Princeton University and the IW staff for providing me with the opportunity to pursue my own independent work project.

8. Honor Code

This paper represents my own work in accordance with University regulations.

- Justin Bi

References

[1] "30 Years of MIDI: a Brief History," https://www.musicradar.com/news/tech/30-years-of-midi-a-brief-history-568009.

[2] "Music21," http://web.mit.edu/music21/.

[3] "Star Spangled Banner Excerpt," https://www.jwpepper.com/The-Star-Spangled-Banner/3083938.item#.YJCkEX1KhQI.

[4] "Giovanni Pierluigi da Palestrina," https://en.wikipedia.org/wiki/Giovanni_Pierluigi_da_Palestrina.

[5] "D3.js," https://d3js.org/.

[6] "sklearn Library," https://scikit-learn.org/.

[7] "Bootstrap Template," https://bootstrapmade.com/mentor-free-education-bootstrap-theme/.

[8] "Piano Roll Helper," http://bl.ocks.org/indirajhenny/552a25ff5e000e55f64ec8cdeaadf72b.