Characteristical Beatmap Analysis of Mobile Rhythm Video Games

Hannah Garcia
Voiland College of Engineering and Architecture
Washington State University

I. Abstract

Rhythmic video games lack in-depth research, especially in the technological field, despite the capabilities achievable through modern day algorithms. While there are a few existing studies on rhythm games, they particularly focus on desktop games such as *osu!* and arcade-style games such as *Dance Dance Revolution.* This study pursues a brand new direction, diving into mobile rhythm games. Specifically, *Hatsune Miku: Colorful Stage's* song data, including BPM and playback, and beatmaps (SVG) files are parsed in order to analyze various difficulty level characteristics through feature selection, principal component analysis, and k-means clustering. Results demonstrate how high difficulty charts are mainly characterized by a large number of notes and a high note density, how fast a song is, and its beat layer ratios. Further investigation of deterministic features, such as note types, still need to be researched.

II. Introduction

The broad spectrum of video games varies from genre to genre. Rhythm games in particular capture one of the most expressive modes of art: music. While studies of the applications of technology have advanced across every field, video games fall into a specific category, or niche, that has not yet been explored. Rhythm games specifically have not been researched as much as other video game genres, despite the large fanbase of millions of players on different platforms, from the popular desktop game osu!, to the infamous Dance Dance Revolution arcade game. With the rising use, accessibility, and advancement of mobile devices in recent years, the era of mobile gaming has become widely popularized worldwide.

Hatsune Miku: Colorful Stage is one of the mobile rhythm games that has practically exploded in popularity. Initially released in Japan on September 30th, 2020 [1], the game was later localised globally, having a total of 39 million users as of March 27, 2024 [2]. Its core mechanics follow common gameplay that other mobile rhythm games adopt, such as *BanG Dream!*, *LOVE LIVE!*, or *Rhythm Hive*.

Hatsune Miku: Colorful Stage gameplay is described to consist of tapping notes as they slide toward the bottom of a mobile device's screen, or the judgment line, according to the rhythm of a song. Note types include *tap*, where players simply tap the note; *hold*, where players tap the note first, then hold until it passes the judgment line; *flick*, where players flick towards a note's arrow direction; and *trace*, where players "graze" over notes [3]. Beatmaps are the final result of all the notes compiled together to fit according to a song's rhythm and in-game playback time. Each beatmap can contain as few as 100 notes to over thousands of notes, usually within a 2-3 minute playback duration.

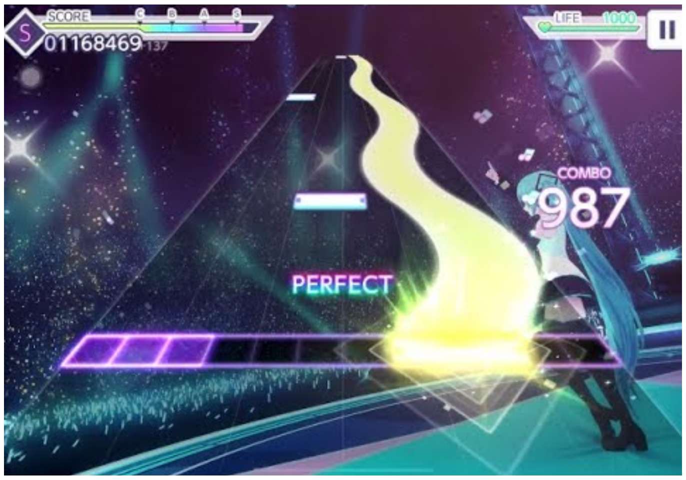


Figure 1. An image of actual gameplay footage, showcasing the player's point of view. *Tap* notes and a *slide* note is displayed.

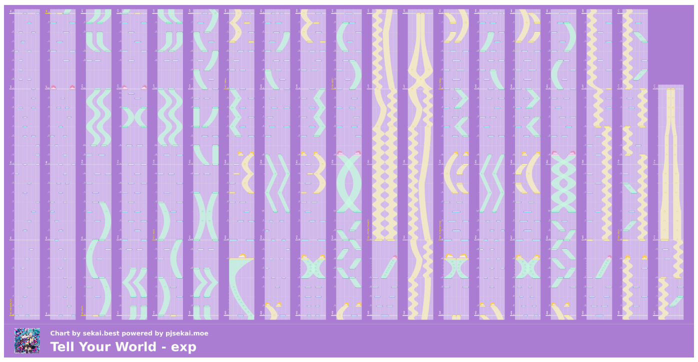


Figure 2. A SVG file depicting a beatmap's full layout, containing all notes and components.

This type of gameplay is commonly followed for mobile rhythm games due to the capabilities of mobile devices compared to other platforms. However, there have been no further studies on the difficulty level scaling methods for this specific genre.

The "rhythm game" genre itself is defined as "a genre of music-themed action … that challenges a player's sense of rhythm" [4]. Players seek out rhythm games as a source of entertainment because of these mechanics. But, if there is no way to properly identify and categorize mechanics accurately, how can a company ensure that their players are able to play these games fairly? Not only that, but ensuring that the difficulty level scale stays consistent across the entire game is important in correctly informing players about the expectations each gameplay mode may have. Tsujino et. al's study, "The Characteristics Study of Dance-charts on Rhythm-based Video Games," and Liang et. al's study, "Procedural content generation of rhythm games using deep learning methods," dive further into the analysis of rhythm game mechanics, highlighting the importance of understanding the characteristics that determine the difficulty of a beatmap [5][6]. From a developer's view, fully analyzing beatmap characteristics and showing a meaningful connection between particular features and beatmap components can lead to advancements in the rhythm game field. From being able to create an efficient and objective evaluation system for difficulty levels, to the generation of new beatmaps through the use of LSTM models, new tools can be explored from a deeper understanding of beatmap characteristics.

The approach for this project is through unsupervised learning using feature selection, principal component analysis, and k-means clustering. Gathering the pre-existing song data according to all the existing songs in the game - including BPM, playback, and total note count - serves as the first step for our approach. Next, by parsing the SVG files of each beatmap in the game, we can derive characteristics such as the number of notes per second, calculated from note positions in an individual file along with general song information (such as BPM). K-means clustering will then be conducted after feature selection and principal component analysis (PCA). These features will reveal any core characteristics that are commonly found in specific clusters, often mimicking the already existing difficulty scales in-game. Different k values, PCs, and other parameters will be tested to see the impact of certain features, from note density to the tempo of a song, while showing any correlations between features and a higher or lower difficulty level.

III. Problem Statement

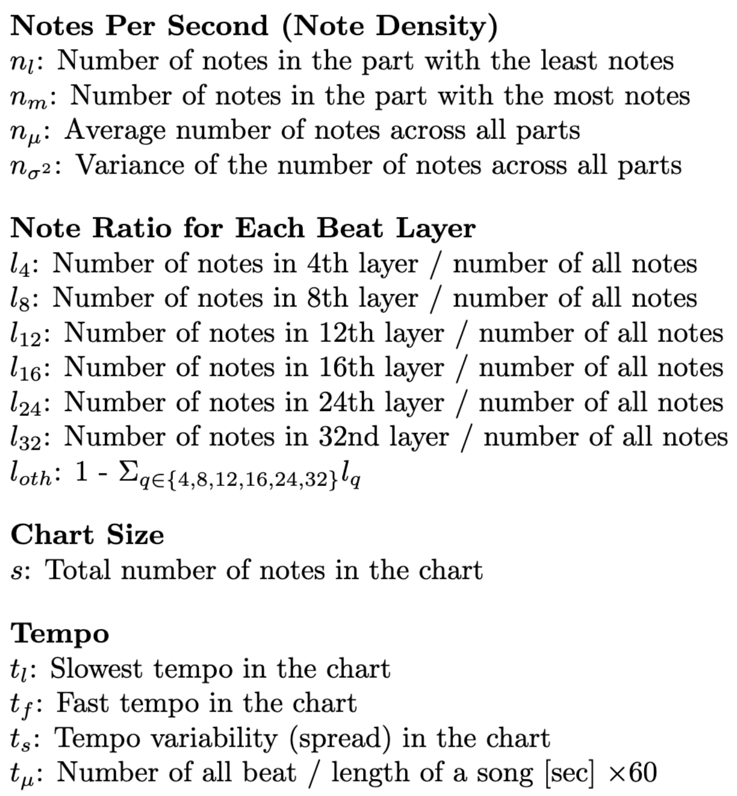
What exactly defines the difficulty of a rhythm game in the first place? Is it the complexity of note combinations, the number of notes (note density), the speed of notes, timing strictness, or the number of misses permitted? (Liang et. al) The definition of difficulty varies from game to game. For *Hatsune Miku: Colorful Stage*, its in-game difficulties are already categorized by two types: difficulty type (categorical) and difficulty level (numerical). This includes EASY, ranging from levels 5 to 10; NORMAL, ranging from levels 10 to 16; HARD, ranging from 15 to 25; EXPERT, ranging from levels 21 to 32; MASTER, ranging from 25 to 37; and APPEND, ranging from levels 23 to 38. It's easy to differentiate between EASY levels against APPEND levels - difficulty types from EASY to MASTER are required to be "playable" with two fingers, while APPEND mode - officially added to the game in September 2023 - introduced a new minimum of 3 fingers or more [7]. "Playable" refers to being able to clear and full combo a specific beatmap. However, difficulty level categorization often gets contrived from an in-depth view. Would an EXPERT level 32 chart be considered more "difficult" than a MASTER level 31 chart? Or, does the difficulty type take further priority when putting these two charts head-to-head? Even if it does, what then would be the difference between an EXPERT level 30 and a MASTER level 30 chart? Are there other characteristics that play a part in determining the difficulty?

Hatsune Miku: Colorful Stage's developers themselves have not given a defined criteria of what determines the difficulty level of a beatmap. They instead state that "the difficulty levels are decided based on how difficult it is to clear the song. For example, songs with a lot of flick notes and difficulty-looking note placements may be set at a higher difficulty. For some songs where there's a big gap between how difficult it is to clear the song and how difficult it is to [full combo] the song, the decision may be based on how difficult it is to [full combo] the song instead" [8][9]. "Full combo" or "FC" refers to the achievement when a player does not miss any notes in a beatmap while clicking each note on time. While this offers context about the types of criterion that determine the final difficulty level, this answer does not give us definite quantitative nor qualitative characteristics found in beatmaps and the songs they correspond to. Without a definite benchmark, how can we ensure the evaluation method used for assigning difficulty levels stays consistent and uniform? Having a simple baseline would not only assist with the future implementation of charts but also allow players to be aware of characteristics they can watch out for or even specifically practice in order to improve steadily at their chosen pace.

With this context in mind, I decided to explore a variety of different score features and song features that may contribute to the difficulty level of a beatmap. There is a wide range of attributes that belong to each beatmap. Does the total number of notes in a chart matter? Or do song attributes play a bigger part in difficulty level? Should we consider one type of note more difficult than another? Is there a standardized way we can consider pattern-based features? There is much to consider when analyzing a beatmap, making it difficult to address every existing component. Thus, this project aims to address a few of the most meaningful yet foundational characteristics.

IV. Models, Algorithms, and Measures

An unsupervised learning approach is used for this study. Feature selection is used to curate a set of designed features, chosen based on characteristics often correlated with a game's "difficulty." Two types of features are used: *score features*, which differ by a chart's difficulty, and *song features*, which differ by song. Song features include the number of notes per second (note density): *n*, the note ratio for each beat layer (corresponding to the complexity of a rhythm based on timing divisions such as 1/4 or 1/8 notes): *l*, and the size or total chart magnitude (number of total notes): *s*. Song features include tempo (BPM) characteristics: *t*. Chart gimmicks, such as note speed or BPM changes, are another feature I aimed to implement. However, due to time constraints, chart gimmicks were handled in data analysis rather than treated as song features. More details on the calculations and data handling are provided in the *Implementation and Analysis* section.


Notes Per Second (Note Density)
n_l: Number of notes in the part with the least notes
n_m: Number of notes in the part with the most notes
n_μ: Average number of notes across all parts
n_{σ^2}: Variance of the number of notes across all parts

Note Ratio for Each Beat Layer
l_4: Number of notes in 4th layer / number of all notes
l_8: Number of notes in 8th layer / number of all notes
l_{12}: Number of notes in 12th layer / number of all notes
l_{16}: Number of notes in 16th layer / number of all notes
l_{24}: Number of notes in 24th layer / number of all notes
l_{32}: Number of notes in 32nd layer / number of all notes
l_{oth}: 1 - $\Sigma_{q\in\{4,8,12,16,24,32\}} l_q$

Chart Size
s: Total number of notes in the chart

Tempo
t_l: Slowest tempo in the chart
t_f: Fast tempo in the chart
t_s: Tempo variability (spread) in the chart
t_μ: Number of all beat / length of a song [sec] $\times 60$

Figure 3. Song and score features computed and used for PCA and clustering.

Note ratio for each beat layer is a unique feature based on Tsujino et. al's implementation of the analysis of the open-sourced game *Stepmania* [5][10]. Tsujino details the concept of a "beat layer," defined as the "set of timing abstained by dividing a bar into *q* equal parts ($q \geq 4$) [that is] defined as '*q*th beat layer.'" The lower layer contains the timing of a note belonging to the *q*th layer, with the specific note being defined as the *"q*th note." "The lowest" refers to the least *q*. Notes that belong to a higher layer mean a more difficult and complex series of notes. To determine the beat layers, we use the formulas detailed in Figure 3.

In order to derive these features, we must collect the appropriate data to compute them. Unfortunately, despite the millions of *Hatsune Miku: Colorful Stage* players, there were no existing datasets satisfying the requirements for this project's objective. This led to heavy importance being placed on the data collection and cleaning portion of the data science process.

For song features, we must collect data containing relevant musical features, including BPM and beatmap duration - not the actual song's duration. For score features, we must collect data differing from chart to chart. This is where the SVG files are used. The game community created these SVG files through official game assets and sources, allowing us to access raw data that can later be cleaned and parsed accordingly.

The data analysis step comes next. With our gathered features, we are now able to use principal components analysis (PCA) to produce a low-dimensional representation of our wide range of features. Lastly, using the top PCs (by variance), we perform k-means clustering, using the elbow plot method to determine the most optimal k to use in analysis.

V. Implementation and Analysis

V. I. Dataset

With no existing dataset readily available for this project, I pulled and compiled the relevant attributes for computing each feature. After thorough research online, the sources I was able to locate include the community pages: Sekai World (the primary GitHub database for the community resource, sekai.best) and Sekaipedia.org (containing other song data missing from Sekai World) [11][12][13].

The song features include BPM and the duration of the song in-game. As *Sekai World* did not contain this data, I was forced to pull from the web-based *Sekaipedia* site. The site used dynamic JavaScript to display song data in a tabular format, including song ID, BPM, duration (game-size), and other attributes. Attempting to initially web scrape with Python and with BeautifulSoup failed despite specialized handling [14][15]. This may be due to the fact that the database was not publicly available from *Sekaipedia* nor could be found in *Sekai World,* making it difficult to iteratively collect the data via web scraping scripts. While not the most efficient method, I copied and pasted the tabular contents into a Google Sheets file named *song_ID_bpm_playback.csv*, allowing me to organize the data in *songID, bpm,* and *playback* columns. Note that songID is the unique ID identifier given to every song in the game. BPM noticeably had special data that required later handling. For example, songID 103 contained a BPM of 115-230 due to tempo changes. Lastly, the playback was stored in a minutes:seconds format. Since much of our feature computation requires the use of total seconds instead of the minutes:seconds format, I handle this by creating a new column in the next data table.

song_metadata.csv is the main data set called in our implementation. Using the previous *songID_bpm_playback.csv* file, plus *musics.json* (English and Japanese versions) and *musicDifficulties.json* from *Sekai World*, the JSON data is fetched through the Requests Python library and stored using the Pandas Python. It is important to note that each song has its own EASY, NORMAL, HARD, EXPERT, MASTER, and possibly APPEND beatmap, meaning that each songID in *musics.json* (containing general song data such as song ID song title, composer, etc.) contained another 5-6 data entries in *musicDifficulties.json* (containing *beatmap* ID, song/music ID, difficulty type, and difficulty level). The end result of the table contained *song_id*; *difficulty* (type) and *difficulty_level*, used for later data analysis; *note_count*, *song_title_jpn* and *song_title_eng*, used for identifying certain songs in data analysis; and *bpm, playback_time_minutes,* and *playback_time_seconds,* pulled and calculated from the previously compiled dataset.

The last and most important dataset used were the SVG files corresponding to each beatmap. From the sekai.best website and the Requests Python library, each SVG file can directly be accessed through the link: *storage.sekai.best/sekai-music-charts/jp{song_id}/{difficulty}.svg*. We can use this uniform structure to our advantage using Requests, where we iterate through for all song IDs (from 1 to 700) to fetch and download each SVG into our local repository. Currently, the *chart_downloads.py* script handles these actions and takes about 20 to 40 minutes to make 4000+ requests. Each SVG file is saved in the format of *songID_difficultyType*. For example, *1_append.svg*.

V. II. Data Preprocessing: SVG Parsing

Data preprocessing began with the exploration of SVG file parsing. Using the *svgpathtools* library and one chart for testing, we first had to discover what components we are able to pull from the SVG files. *Svgpathtools* contains functions that allow us to easily read SVG files and use geometrically-oriented tools to transform and analyze path elements. This project primarily focused on reading the SVG files first in order to derive the score features needed. The *svg2paths()* function was used to read the path objects and the list of dictionaries of attributes from an SVG file. Each path would usually follow the format {'class': 'bar-line', 'x1': '40', 'x2': '232', 'y1': '2670', 'y2': '2670'}, where the *class* corresponds to one of the path attribute classes, *x1* and *x2* correspond to the starting and ending x-values, and *y1* and *y2* correspond to the starting and ending y-values.

The unique path attribute classes that we are able to read from our data set include:

- *background, meta, meta-line*: Formatting components.
- *bar-count-flag*: Background rectangles for bar counts in a chart, contained within lanes and broken down by #'s
- *bar-line*: Lines corresponding to a musical measure contained within a lane
- *beat-line*: Lines for timings within each bar. Each line roughly represents quarter notes in a musical measure.
- *decoration*: Background elements or decorative paths (unclickable)
- *event-flag:* Flags for in-game events (skill activation, Super Fever, etc.)
- *lane:* Main lanes containing all other components, such as bar-lines. Each chart usually contains about 13 lanes.
- *lane-line:* Breaks down each lane into sections.
- *slide* and *slide-critical*: Geometric attributes corresponding to slide notes.
- *speed-line*: Markers for speed or tempo changes.
- *tick-line:* Tick lines for note timing.

With the above given information and exploratory mapping and graphing, we can deduce the following class hierarchy and relationships: lanes consist of several bars that are separated by a

#. Each bar contains beat lines (about 3 per bar) for separation. Breaking down the structure further, each bar has tick-lines corresponding to the note and tick timings of a chart.

We also discovered a couple of major issues: While the SVG files look as if they are spread out across the x-axis via lanes, the components only contain varying y-positions. This meant we would have to consider vertical positions, with a higher y showing earlier song timing, while a lower y showing later timing. Thus, sorting notes from the highest Y to the lowest Y recreates the temporal order of the song. This is also shown in the five unique bar-lines parsed from SVG files, corresponding to the four bars per lane that repeat in a cycle. *x1* and *x2* always equaled 40 and 232 respectively. However, y-values ranged from 32, 692, 1351, 2011, and 2670. This is an issue, as we need to calculate the number of notes per second, where we should divide charts into parts for each second and the note ratio for the beat layer, based on timing divisions.

Because of this layering, we must handle each relevant path attribute accordingly. We begin with our bar lines. We only keep one bar-line per unique y-position to represent the measure, avoid zero-height bars, and avoid confusion when mapping notes to bars. We additionally extend the first and last bars to include notes outside the bar range in the case of outliers. This way, although we have a small number of bar lines, we can still cover all notes in the chart. Then, using the bar-lines, we create our bars - defined by the space between two consecutive bar-lines, or top Y and bottom Y bar-lines. This gives us the vertical range of each bar for mapping notes.

This leads us to the next major issue: parsing the SVG files does not give us any playable notes to work off of. We are only given slide-notes and tick-lines, which often contain a much higher count than the real total note count due to song tempo. However, tick-lines are our closest way of mimicking “real” notes in a beatmap. A high tick-line may also indicate a stricter check when players perform slide notes (for example, if the position of the slide note changes instead of remaining straight). Thus, tick-lines serve as a solid and usable basis for this project. Using only the 157 unique y-values correlated to the tick-lines, we map notes to the previously composed bars. For each tick-line, we look at its y-position and find the bar whose y-range contains the note. Similarly, we map slide notes to each bar.

Next is the mapping of beat-lines, which assists with accurately mapping out a tick-line in its bar based on song timestamps. There are 15 unique beat-lines with y-values ranging from -463, -298, -133, 197, 363, 527, 856, 1021, 1186, 1516, 1681, 1846, 2176, 2340, and 2505. Each bar may have multiple beat-lines representing the subdivisions of a bar. Thus, for each beat line, we find the closest bar by y-position, using bar midpoints. We create a dictionary of a bar index and its list of beat-line positions. Cleaning up the mapping, we sort beats from earliest to latest by y-position and handle any single-beat bars to avoid zero-division in later computation.

Now, we can calculate the timestamps for our tick-lines and slide notes. We use song duration in seconds, beats per minute (BPM), beats per bar (4), and seconds per beat (60 / BPM) for mapping. We first attempt beat-line interpolation, where we find two consecutive beat lines a note falls between, compute its relative position between those beats (in other words, how far between the beats the note lies), and convert that position into time using BPM. If a bar has zero beat lines or notes outside a beat-region, we must approximate the time stamp. We use linear mapping from the top of the bar to the bottom of the bar. Lastly, we scale all timestamps calculated to ensure the maximum timestamp matches the song's real playback time. Figure 4 simply visualizes the mapping of each class. Slide notes appear close together vertically due to their structure, as they occur within the same beats or bar (vertical space). Tick notes are more spread out since they represent timing points.

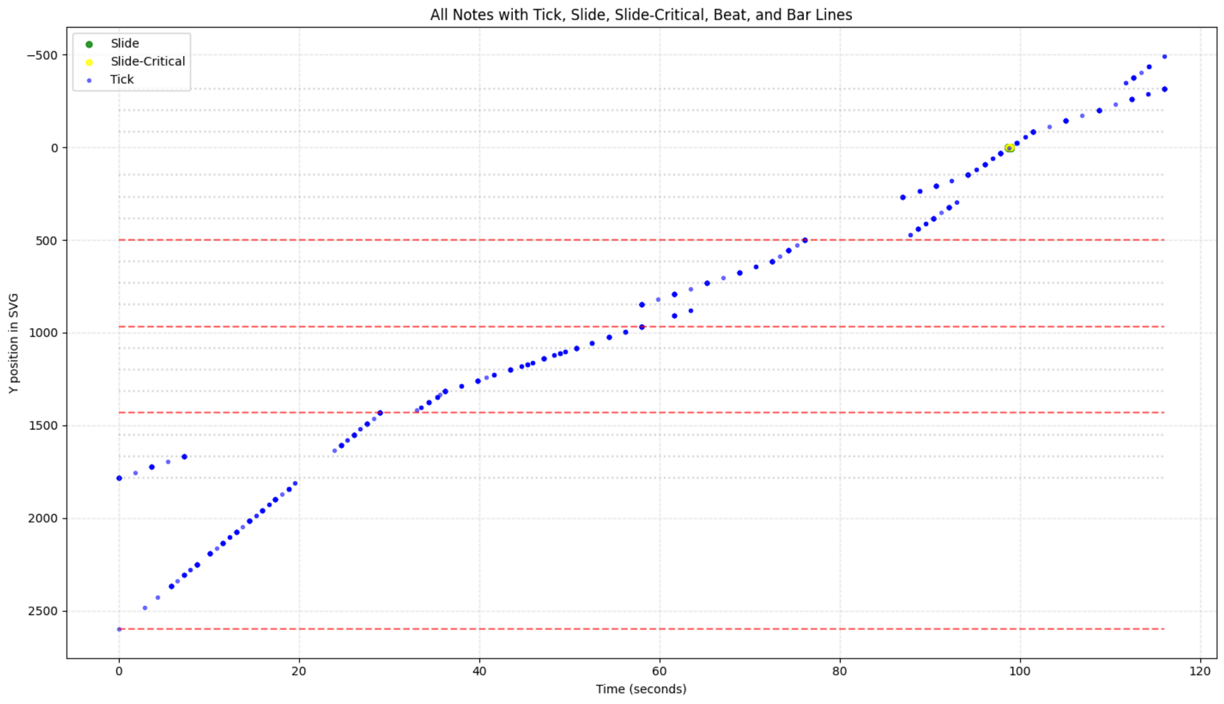


Figure 4. Visual mapping of all notes with tick-line, beat-line, bar-line, slide, and slide-critical.

V. III. Feature Extraction

With data preprocessing complete, we can proceed with the computations required for the selected features in Figure 3.

Beginning with the note ratio for each beat layer, we first must create a synthetic beat grid, which serves as a normalized timeline for each bar. We first ensure we have a scale factor that is consistent with our previous implementations, normalizing note times so all bars fit within the song's duration. Then, when building the synthetic beat grid, we divide each bar into its multiple *q* subdivisions, mapping notes to positions relative to the bar in a structured, standardized way. Here, we divide each bar into fractions, or layers, such as quarter notes or eighth notes. Then, we classify a single note or tick-line into a *q*-layer using the synthetic grid. We handle notes that may fall slightly outside of the grid to ensure we are still mapping them, then assign each note to each subdivision, starting from the smallest *q* to the largest *q*. If no *q* is matched, we save the tick or note as *oth*. Lastly, we compute counts and ratios after classification for each *l*. This method is in accordance with the definition of Tsujino et. al's beat layers.

Next, we compute the notes per second, or note density *n*, from Figure 3. Using the note and tick-line times, song duration, and a bin size of 1.0 per second with the Numpy Python library, we count notes in each 1-second bin and compute n_l, n_m, n_μ, and n_{σ^2}.

Since the scale *s* of a beatmap is already stored in the *song_metadata.csv* file, we can compute our final feature: tempo. Due to time limitations on this project, we do not handle any special BPM events during feature extraction. However, we parse special BPM by handling ranges, such as 150-170 through its average value, and parenthesis such as 92 (184) by using the first numerical value. For the other tempo characteristics, we use the song's BPM, beat count, and the song duration to compute t_l, t_f, t_s, and t_μ.

V. IV. Principal Component Analysis (PCA) and K-Means Clustering

For the following methods, we use the numpy and sklearn Python libraries. We perform principal component analysis (PCA) to capture the most meaningful features according to variance. We do this by collecting all features into a data frame and standardizing feature groups to reduce noise. Optionally, we can downweight rare layers to ensure outliers do not skew data and clusters significantly. For example, layers l_4 or l_8. We then apply PCA per group. For example, note density has n_l, n_m, n_μ, and n_{σ^2}. PCA is applied to each variable, resulting in the variance explained by PC1: 0.8760 (87.6%), PC2: 0.0987 (9.9%), PC3: 0.0253 (2.5%), and PC4 (n_var): 0.0 (0.0%). Using PCs demonstrates how the most meaningful variation in note-density patterns can be represented with the first two principal components.

Lastly, we perform k-means clustering with the principal components. After compiling the final feature matrix by concatenating PCs from each group, we achieve the final feature matrix shape in the form of *(number of total charts analyzed*, *number of selected features*), i.e., (3099, 15). Using a *k*-range from 2 to 35, we perform clustering for each *k* in order to generate an elbow plot through the sum of squares. PCA coordinates *pca_x* and *pca_y* from clustering are used for distance calculations, where centroids are computed for the sum of square distances.

VI. Results and Discussion

The feature t_μ is excluded from the following clustering methods due to possible distortions in data, as the computation of beat lines used for tempo is conducted manually. Computed tempo values seem to be empirically underestimated - while expected to be within 100 to 200 BPM, they result in values between 7 to 10. Separate invesigations were conducted while including t_µ, and results produce a similar outcome as the following, detailed results: lower-difficulty charts (EASY, NORMAL, HARD) naturally cluster together, and higher-difficulty charts (EXPERT, MASTER) cluster together depending on difficulty level. However, including for k = 5 fails to accurately group MASTER and APPEND beatmaps together. Thus, t_µ is excluded to avoid including any possible bias from unreliable tempo estimates in the following analysis and discussion.

Figure 5 depicts the elbow plot generated for clusters 2 to 10, observing a smaller range of values from the top-level. Figure 6 depicts the elbow plot for clusters 2 to 35, observing clusters closer.

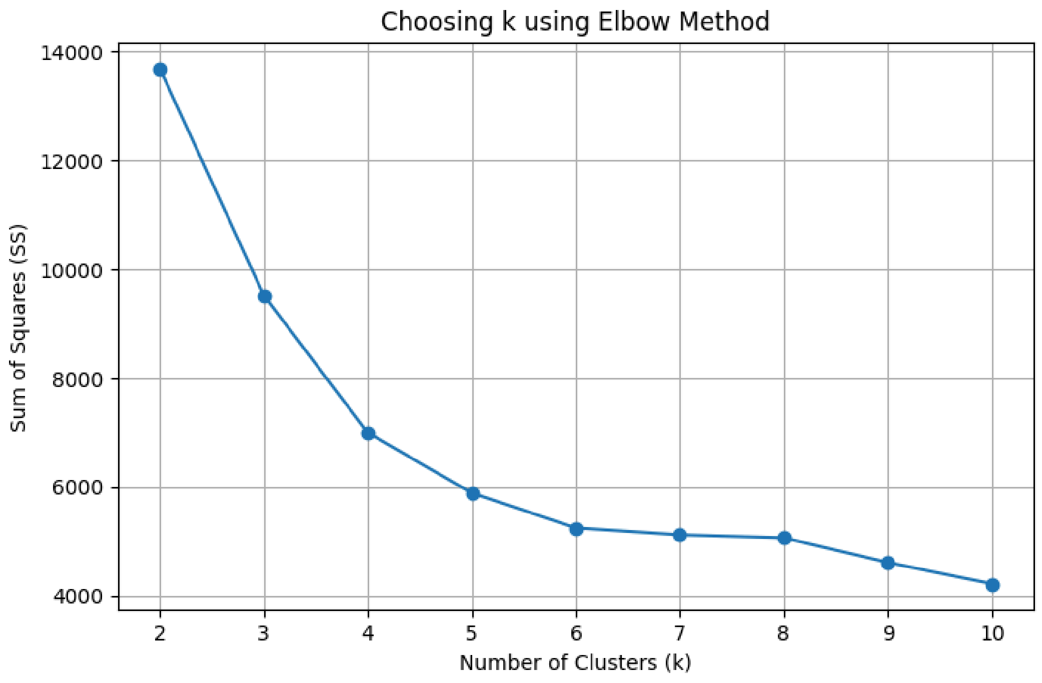


Figure 5. Elbow plot for *k = 2* to *k = 10*.

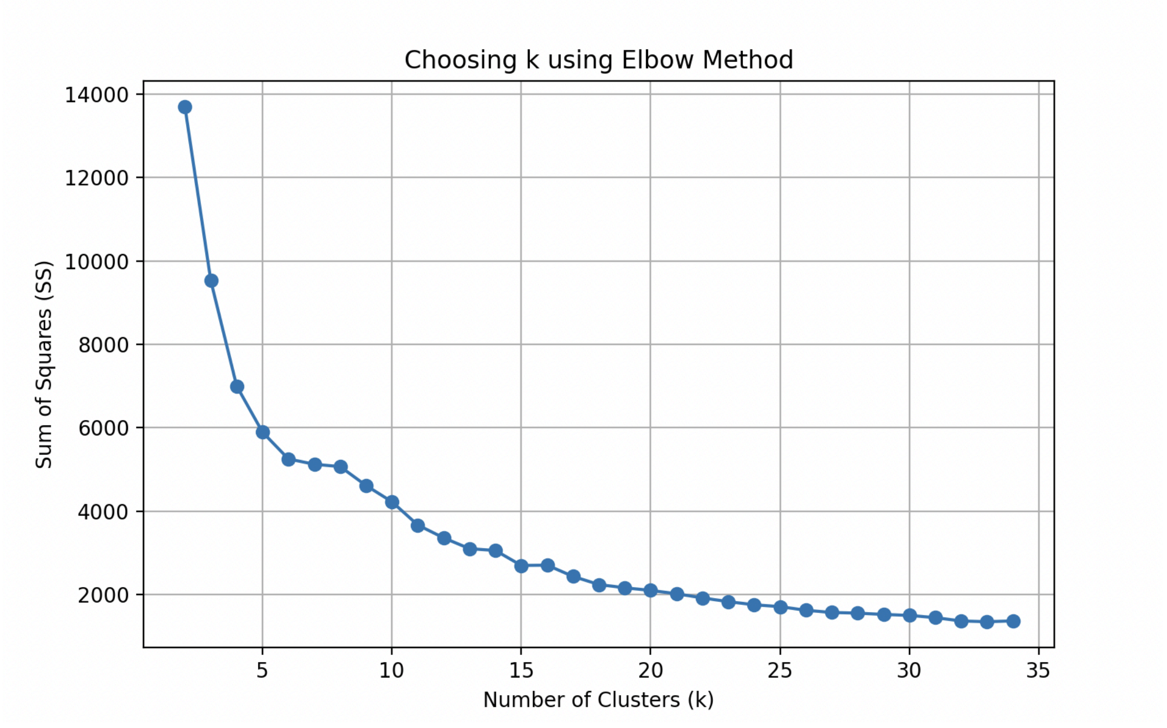


Figure 6. Elbow plot for *k = 2* to *k = 35*.

This paper will mainly analyze the results for *k* = 5, according to the optimal *k* value from the plot. We later observe various, higher *k* values briefly. Tables 1 to 4 depict the average metric per cluster for *k = 5,* while Figure 7 depicts the results of *k = 5* clustering.

V.I I. *k* = *5* Clustering

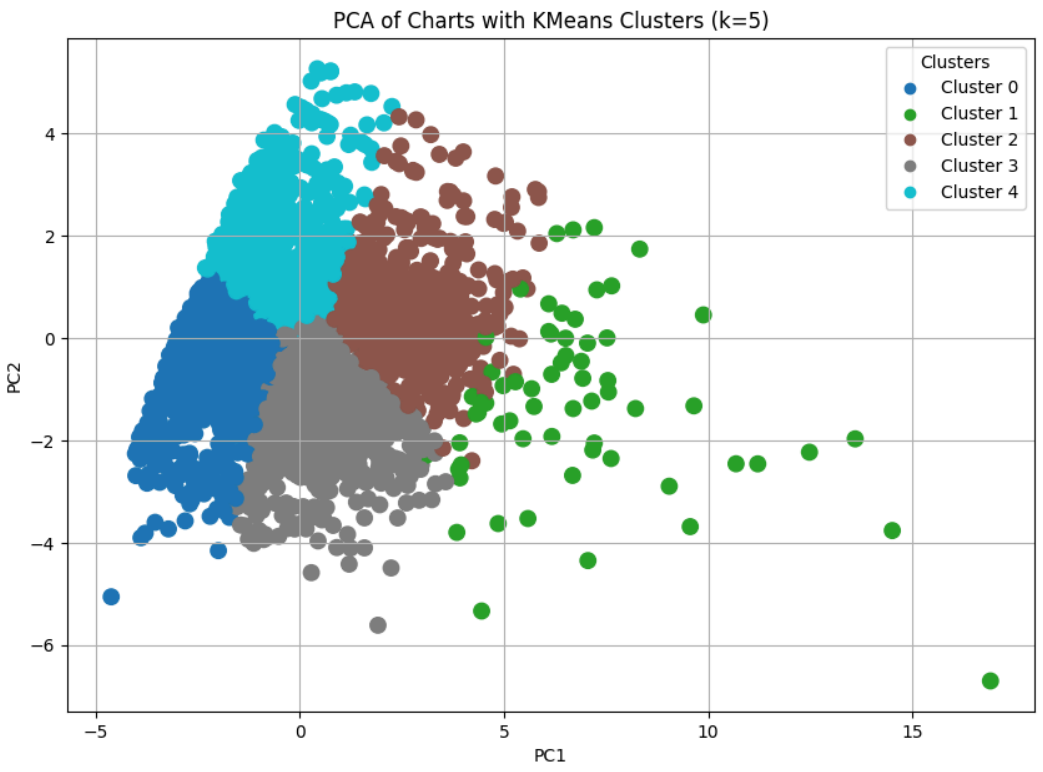


Figure 7. Resulting graph of *k* = *5* clustering, with PC1 and PC2 used for the 2D axis.

cluster	**# of charts**	***n_l***	***n_m***	***n_*μ**	***n_*σ^*2***
0	926	0.0	122.73	7.1353	279.23
1	64	0.0	114.50	6.883	254.69
2	579	0.0	123.71	7.121	289.29
3	781	0.0	123.43	7.086	280.92
4	749	0.0	122.32	7.169	286.71

Table 1. Note density features *n* for *k* = 5 clustering.

cluster	**# of charts**	***l_4***	***l_8***	***l_12***	***l_16, l_24, l_32, l_oth***
0	926	0.9990	0.0005364	0.0004357	0.0
1	64	0.9977	0.0001915	0.002115	0.0

2	579	0.9991	0.0003607	0.0004895	0.0
3	781	0.9993	0.0003449	0.0003983	0.0
4	749	0.9989	0.0004387	0.0006292	0.0

Table 2. Note ratio for each beat layer *l* for *k* = 5 clustering.

cluster	**# of charts**	***t_l***	***t_s***	***t_f***	***t_*μ**
0	926	154.9	154.9	154.9	8.786
1	64	156.6	156.6	156.6	8.397
2	579	157.1	157.01	157.01	8.895
3	781	155.8	155.8	155.8	9.054
4	749	152.6	152.6	152.6	8.531

Table 3. Tempo *t* for *k* = 5 clustering.

cluster	**# of charts**	***total_notes_svg***	***total_notes_csv***
0	926	898.3	689.2
1	64	878.34	670.3
2	579	905.8	700.2
3	781	900.9	692.7
4	749	899.4	682.1

Table 4. Scale *s* for *k* = 5 clustering, from total notes (tick-lines and slide notes) from the SVG and the actual total playable notes from the CSV.

Overall, notice how all clusters often fall under *l_4* (number of notes in the 4th layer / number of all notes), with little to no notes in *l_8* and beyond. Other metrics varied in value. The total chart count is 3099, with 600 EASY charts, 593 NORMAL, 595 HARD, 598 EXPERT, 594 MASTER, and 119 APPEND, as not every chart has an APPEND mode. Going over the clusters from most to least difficult reveals the following:

Cluster 0 contains 926 charts. Relevant metrics include the third smallest *n_m* (number of notes in the part with the most notes), second smallest *n_*σ*^2* (variance of the number of notes across all parts), second lowest *t_l, t_s,* and *t_f* (slowest, variability, and fastest tempos, respectively), and second smallest *s* (total number of notes in the SVG chart). Cluster 0 additionally had the largest *l_4* value. Breaking down the cluster further shows the cluster consists of 334 EASY charts, 326 NORMAL, 231 HARD, 25 EXPERT, 9 MASTER, and 1 APPEND - mainly EASY, NORMAL, and HARD - depicted in Figure 8. Difficulty levels mainly range from 6 to 17, with the most common difficulty levels being Level 6 (184 charts, or 20% of the cluster), Level 12 (156 charts, or 17%), and Level 17 (86 charts, or 9%), depicted in Figure 9.

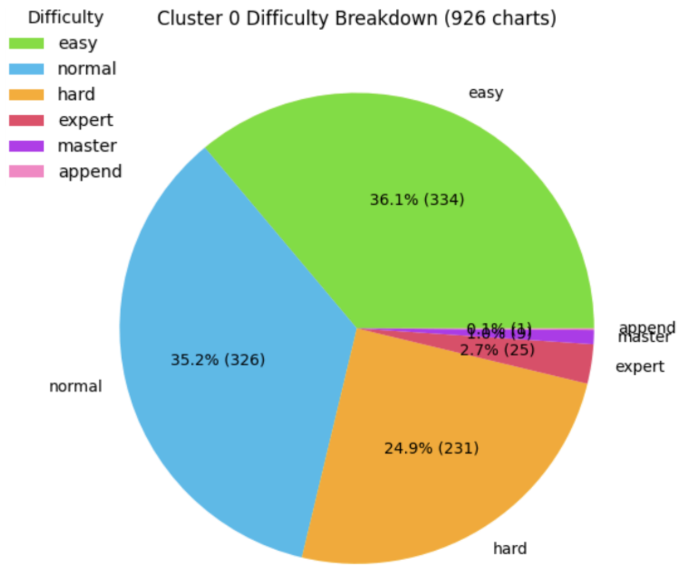


Figure 8. Cluster 0's difficulty type breakdown.

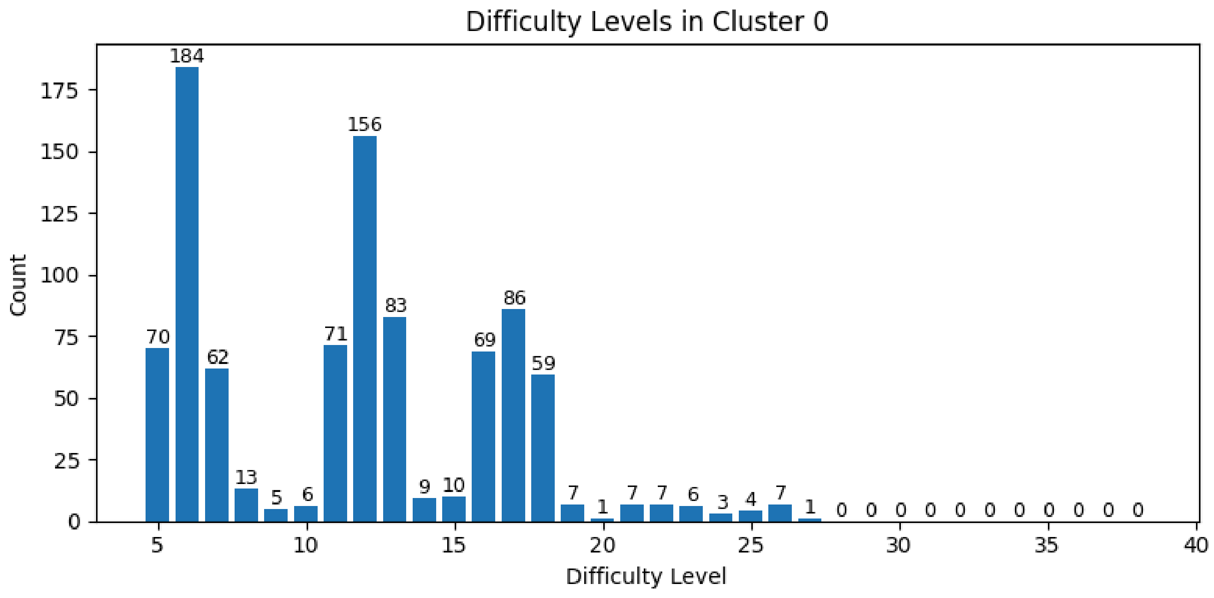


Figure 9. Cluster 0's difficulty level breakdown.

Cluster 4 contains 749 charts. Relevant metrics include the second smallest *n_m*, lowest *t_l, t_s,* and *t_f*), and the midpoint average *s*. Cluster 4 had the highest *l_12* value. The cluster consists of 267 NORMAL charts, 266 EASY, 204 HARD, 12 EXPERT, and 0 MASTER and APPEND - mainly EASY, NORMAL, and HARD - depicted in Figure 10. Difficulty levels mainly range from 8 to 18, with the most common difficulty levels being Level 13 (126 charts, or 17% of the cluster), Level 12 (107 charts, or 14%), and Level 9 (88 charts, or 11%), depicted in Figure 11.

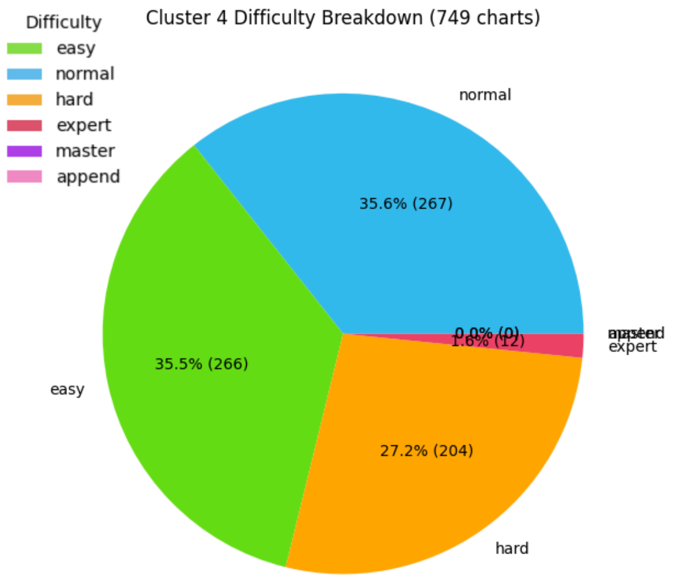


Figure 10. Cluster 4's difficulty type breakdown.

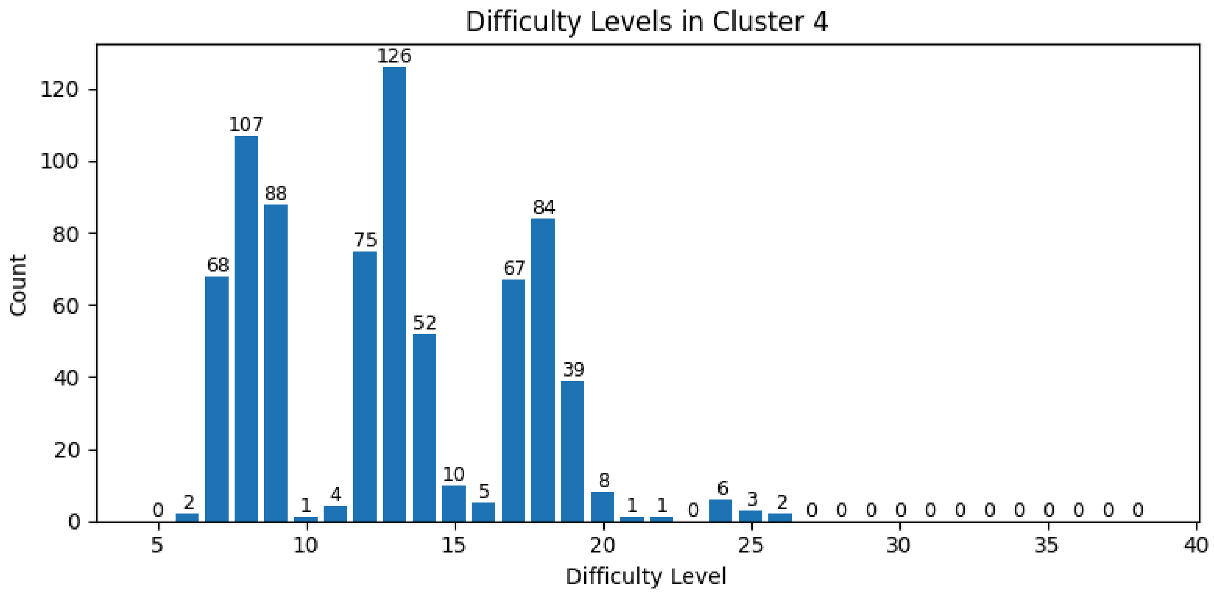


Figure 11. Cluster 4's difficulty level breakdown.

Cluster 3 contains 781 charts. Relevant metrics include the second largest *n_m,* the second largest *n_σ^2*, and the second largest *t_l, t_s,* and *t_f*. Breaking down the cluster shows the cluster consists of 318 EXPERT charts, 311 MASTER, 101 HARD, 51 APPEND, and 0 EASY and NORMAL - mainly EXPERT and MASTER - depicted in Figure 12. Difficulty levels jump to a range from 24 to 29, with the most common difficulty levels being Level 24 (109 charts, or 14% of the cluster), Level 28 (106 charts, or 14%), and Level 29 (93 charts, or 12%), depicted in Figure 13.

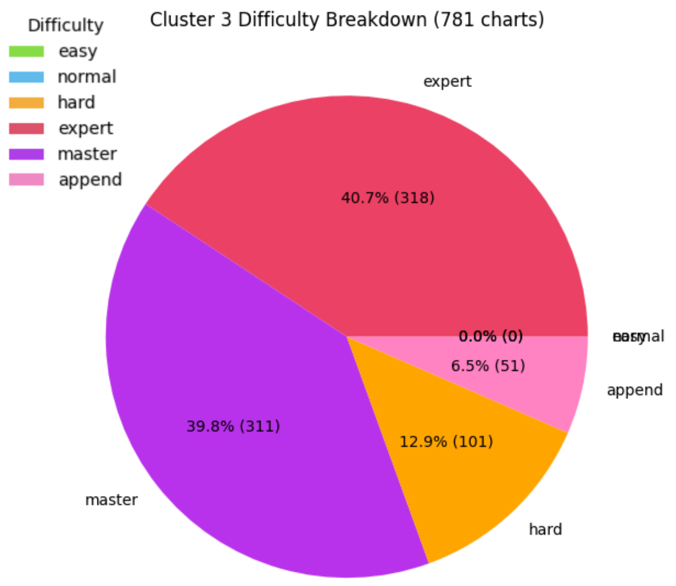


Figure 12. Cluster 3's difficulty type breakdown.

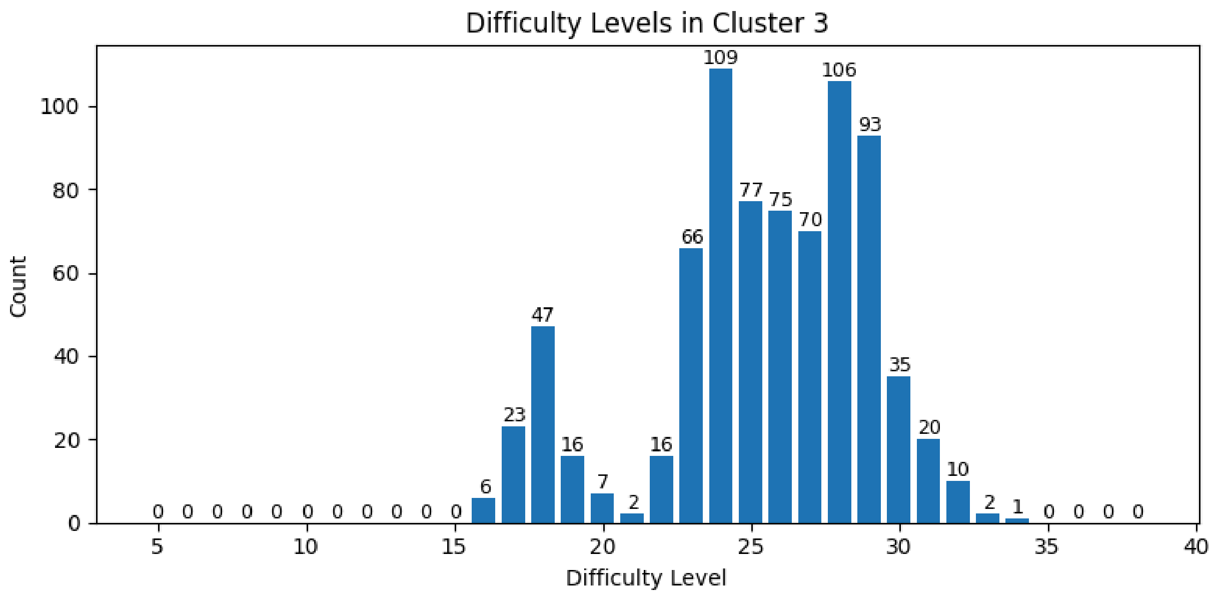


Figure 13. Cluster 3's difficulty level breakdown.

Cluster 2 contains 579 charts. Relevant metrics include the largest *n_m,* the largest *n_*σ^*2*, and the largest *t_l, t_s,* and *t_f* , and the largest *s*. Observing cluster 2 further shows the cluster consists of 247 MASTER charts, 236 EXPERT, 59 HARD, 37 APPEND, and 0 EASY and NORMAL - mainly EXPERT and MASTER, with slightly fewer APPEND charts than cluster 3 - depicted in Figure 14. Difficulty levels slightly shift to mainly 25 to 30, with the most common difficulty levels being Level 30 (79 charts, or 14% of the cluster), Level 25 (75 charts, or 13%), and Level 31 (75 charts, or 13%), depicted in Figure 15. Cluster 2 is very similar to cluster 3, but what sets the two apart is a greater number of MASTER charts in cluster 2. Additionally, Cluster 2 has the greatest metric values.

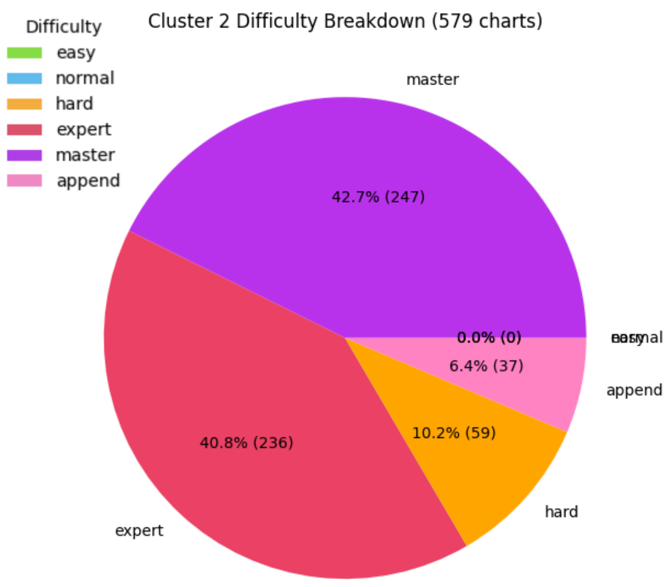


Figure 14. Cluster 2's difficulty type breakdown.

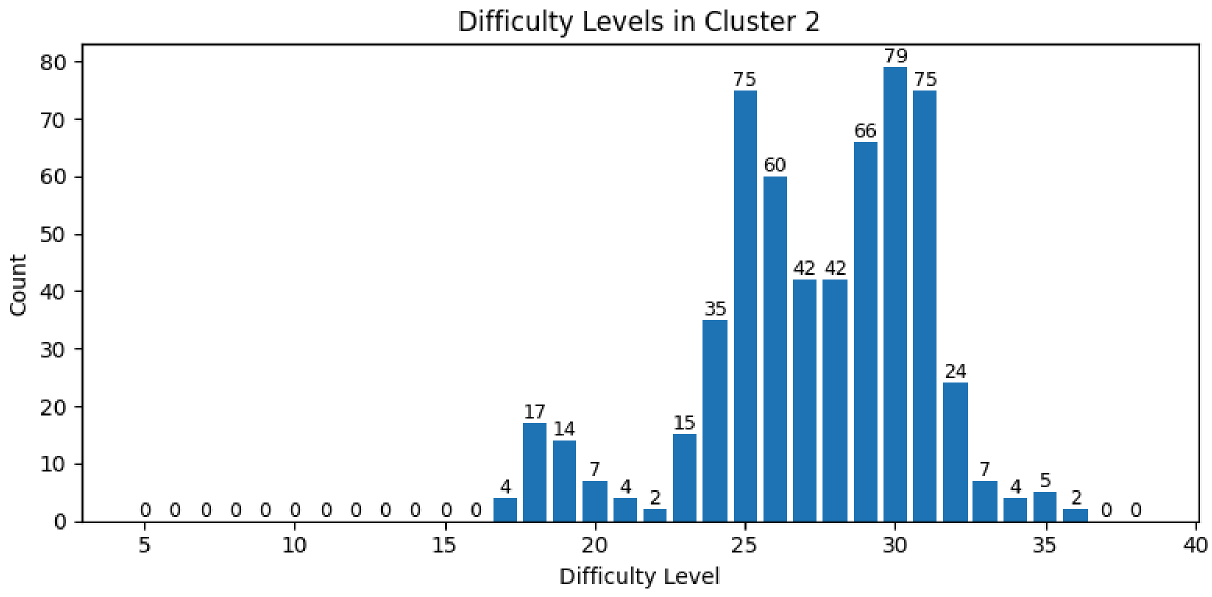


Figure 15. Cluster 2's difficulty level breakdown.

Lastly, cluster 1 contains 64 charts, which is a significantly lower count than the other clusters. Despite the cluster having the lowest averages of each metric, clustering categorized the most difficult types and levels accurately. Figure 16 depicts cluster 1 containing 30 APPEND charts, 27 MASTER, 7 EXPERT, and 0 EASY, NORMAL, and HARD charts. Figure 17 depicts how cluster 1 contains the highest difficulty levels compared to previous clusters, mainly ranging from 31 to 37. Note that these higher difficulty levels are uncommon - i.e., there is only 1 chart of difficulty level 38, 11 charts of difficulty level 35, and 6 charts of difficulty level 36 in the game. Cluster 1 successfully shows the grouping of such unique, rare, and difficult charts. For comparison, Figure 18 visualizes the range of difficulty types across each cluster.

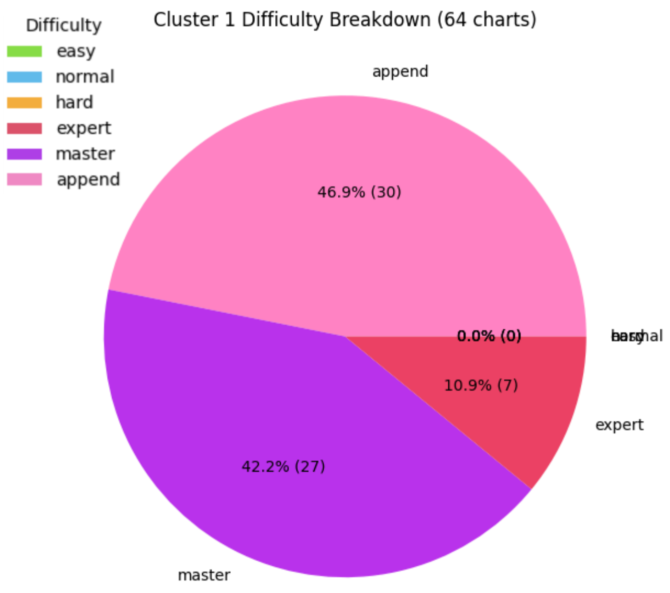


Figure 16. Cluster 1's difficulty type breakdown.

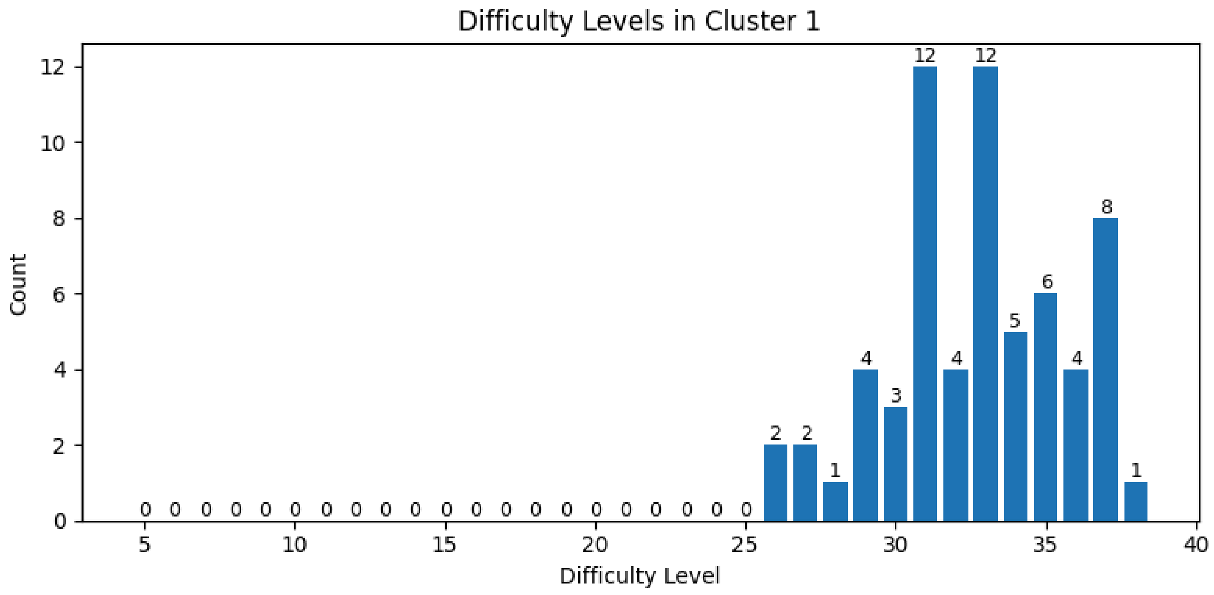


Figure 17. Cluster 1's difficulty level breakdown.

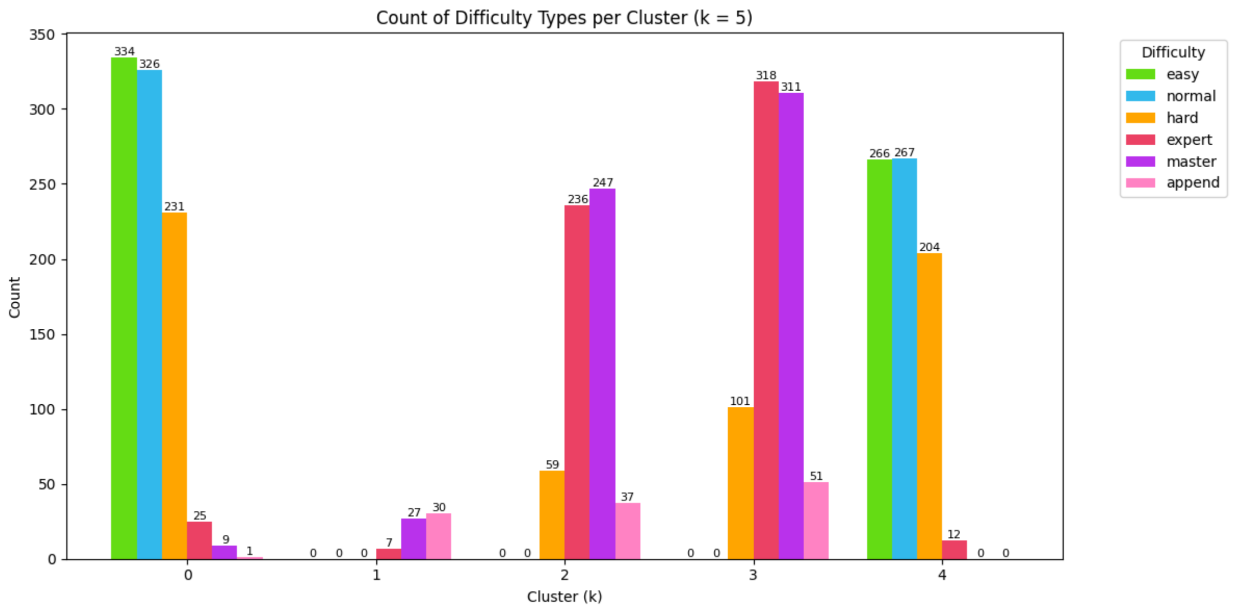


Figure 18. Difficulty type breakdown across all clusters.

Lastly, Figure 19 shows the note range across each cluster. Notice how cluster 0, which has been attributed with a low note density, tempo, and chart size, contains the least amount of notes - usually under 600 or 800 notes. Clusters 2 and 3, attributed with a high note density, tempo, and chart size, contain the largest amount of notes - thousands of notes.

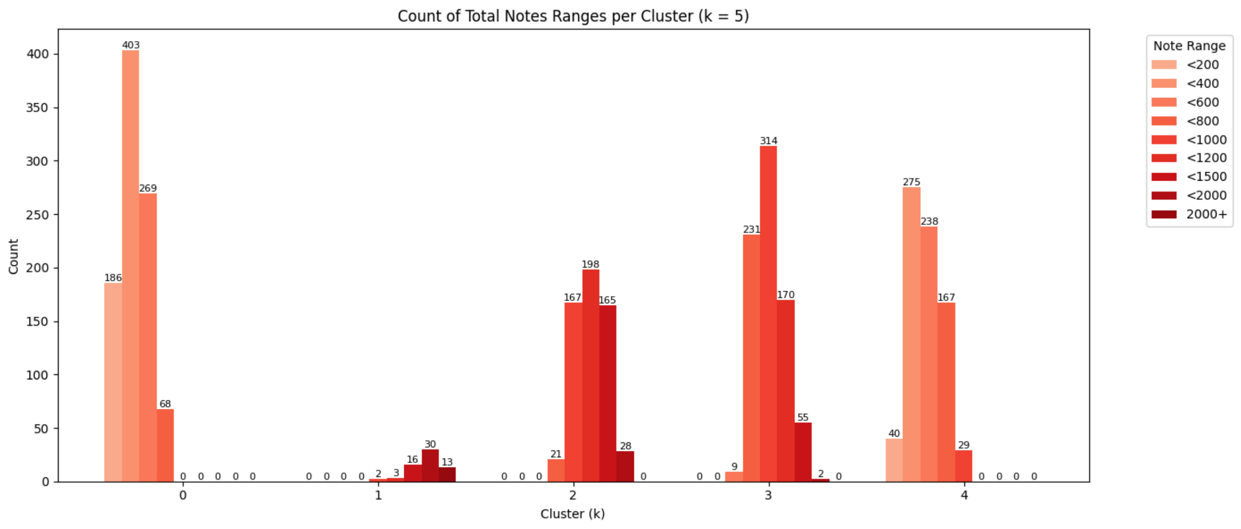


Figure 19. Count of total note ranges per cluster.

V.I II. *k = 10* Clustering

Choosing a larger *k* value, such as 10, comparatively splits up clusters further, getting rid of large “jumps” in difficulty (i.e., difficulty level jumps between cluster 3 and 4 with *k* = 5).

Additionally, we see how outliers get grouped together more, such as Cluster 4 in Figure 20. Similar links between higher difficulty levels through metrics such as a high note density are also supported with *k = 10* clustering.

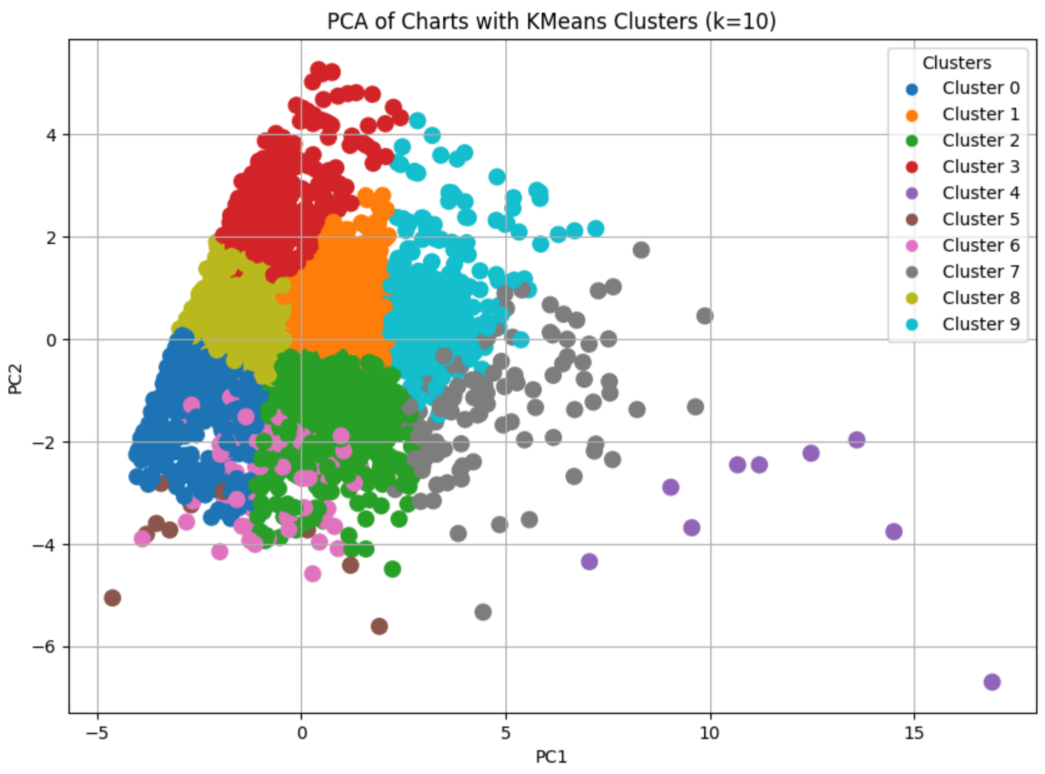


Figure 20. Results of *k = 10* clustering.

V.I III. Overall Discussion

Analyzing the overall results leads to high difficulty charts often being linked to a large number of notes, a high note density, and how fast a song is. The beat layer did not seem to have a strong correlation between a beatmap chart and its difficulty level. However, the estimations made for note and tick-line mapping during data pre-processing may have contributed to less impactful metrics. Tsujino et. al's works showed a higher beat layer corresponding to a higher difficulty level; a more complex and accurate mapping for q-layers may be required to fully capture this metric. However, it is noticeably more common for notes to fall within the 4th layer compared to other beat layers even in Tsujino et. al's work.

With the amount of tick-lines used for computation during feature computation, this algorithm ends up placing great emphasis on musical-related components and the number of notes as well. This can serve as both a strength and weakness when analyzing beatmap difficulty levels, depending on the features we would like to take into heavy consideration. Due to SVG parsing limitations, it is difficult to incorporate features accurately. Future investigation in SVG parsing methods can assist with optimal feature selection.

VII. Related Work

Yudai Tsujino, Ryosuke Yamanishi, and Yoichi Yamashita's study, *"Characteristics Study of Dance-charts on Rhythm-based Video Games*," was the main paper referenced throughout the implementation of this study [5]. Using an existing dataset for *Stepmania,* they study the characteristics of dance-charts through clustering by using the k-means method with a set of designated features. Their results demonstrate that clusters were composed mostly based on step frequency and the complexity of a rhythm. In "*Dance Dance Gradation: a generation of fine-tuned dance charts,*" by Yudai Tsujino and Ryosuke Yamanishi, further explored the concept, implementation, and use of beat layers and their step ratios [10]. My approach was different during the feature computation step: due to the creation of the dataset through SVG file parsing, I mapped beatmap characteristics manually. Additionally, *Stepmania* has different core gameplay mechanics as a desktop rhythm game as compared to *Hatsune Miku: Colorful Stage*'s mobile format. Thus, Tsujino et. al investigated a variety of other features, such as the "tf-idf of 3-gram steps," or the frequency of arrows for continuous 3 steps. Here, they categorize a specific type of pattern that can be found in *Stepmania*'s beatmaps, which has not been done in this study due to SVG file limitations.

VIII. Conclusion

This study investigated and designed specialized song and score features according to the characteristics found in beatmaps through k-means clustering. Results of data analysis confirm that difficulties are mainly characterized by the features that have been selected: note density, chart size, and tempo. Beat layer ratios are not the most accurate due to SVG parsing and manual note mapping, with only some correct patterns. However, overall results lead to high difficulty charts having a large number of notes, a high note density, and a fast tempo.

Future improvements and investigation that can be conducted in the future include specialized handling for songs with BPM changes and beatmaps that contain particular "gimmicks" such as note speed changes or chart freezes [16]; attempting different methods of parsing for more accurate note mappings and identifying specific types of notes for feature selection; investigations of charts that fail to be parsed or read correctly; and the optimization of data collection for CSV and SVG files.

IX. Bibliography

[1] "Hatsune Miku: Colorful Stage!," *Wikipedia*, Mar. 26, 2023.
https://en.wikipedia.org/wiki/Hatsune_Miku:_Colorful_Stage

[2] “HATSUNE MIKU: COLORFUL STAGE! on Instagram”, *Instagram*, March 27, 2024. https://www.instagram.com/p/C5BMKEnLMqG/.

[3] Sekaipedia, “Rhythm game mechanics - Sekaipedia,” *Sekaipedia*, Aug. 16, 2022. https://www.sekaipedia.org/wiki/Rhythm_game_mechanics

[4] “Rhythm game,” *Wikipedia*, Dec. 12, 2019. https://en.wikipedia.org/wiki/Rhythm_game

[5] Y. Tsujino, R. Yamanishi, and Y. Yamashita, “Characteristics Study of Dance-charts on Rhythm-based Video Games,” *IEEE Xplore*, Aug. 01, 2019. https://ieeexplore.ieee.org/document/8848126/.

[6] Y. Liang, W. Li, and K. Ikeda, “Procedural Content Generation of Rhythm Games Using Deep Learning Methods,” *Entertainment Computing and Serious Games,* pp. 134–145, 2019, doi: https://doi.org/10.1007/978-3-030-34644-7_11.

[7] “APPEND,” *Project SEKAI Wiki*, 2025. https://projectsekai.fandom.com/wiki/Category:APPEND (accessed Dec. 04, 2025).

[8] “プロジェクトセカイ カラフルステージ！ feat.初音ミク,” プロジェクトセカイ カラフルステージ！ feat.初音ミク, 2025. https://pjsekai.sega.jp/news/article/index.html?hash=f9e1eb72ee92df7c4812fd16160b459eb8aa09e3 (accessed Dec. 04, 2025).

[9] *X (formerly Twitter)*, 2025. https://x.com/pjsekai_eng/status/1691383595343245312 (accessed Dec. 04, 2025).

[10] Y. Tsujino and R. Yamanishi, “Dance Dance Gradation: A Generation of Fine-Tuned Dance Charts,” *Lecture Notes in Computer Science*, pp. 175–187, 2018, doi: https://doi.org/10.1007/978-3-319-99426-0_15.

[11] “Sekai World,” *GitHub*, Dec. 07, 2025. https://github.com/Sekai-World

[12] “Sekai Viewer,” *sekai.best*. https://sekai.best/

[13] “Sekaipedia,” *Sekaipedia*, Sep. 06, 2023. https://www.sekaipedia.org/wiki/Main_Page

[14] KOrfanakis, “Web Scraping With Python,” *GitHub*, 2025. https://github.com/KOrfanakis/Web_Scraping_With_Python/tree/main/

[15] oxylabs, “Scraping Dynamic JavaScript Ajax Websites With BeautifulSoup,” *GitHub*, 2025. https://github.com/oxylabs/Scraping-Dynamic-JavaScript-Ajax-Websites-With-BeautifulSoup?tab=readme-ov-file.

[16] “Songs with gimmicks,” *Project SEKAI Wiki*, 2023. https://projectsekai.fandom.com/wiki/Category:Songs_with_gimmicks

X. Appendix

Appendix A: GitHub Repository
https://github.com/hkgarcia/prsk-difficulty-clustering