MCDI

MAESTRÍA EN CIENCIA DE DATOS E INFORMACIÓN

INFOTEC CENTRO DE INVESTIGACIÓN E INNOVACIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIÓN

DIRECCIÓN ADJUNTA DE INNOVACIÓN Y CONOCIMIENTO

GERENCIA DE CAPITAL HUMANO

POSGRADOS

NEURAL NETWORK ASSISTED COMPOSITION FOR PIANO IN JAZZ

Propuesta de Intervención
Que para obtener el grado de MAESTRO EN CIENCIAS DE DATOS E INFORMACIÓN

Presenta:

Ismael Medina Muñoz

Asesor:

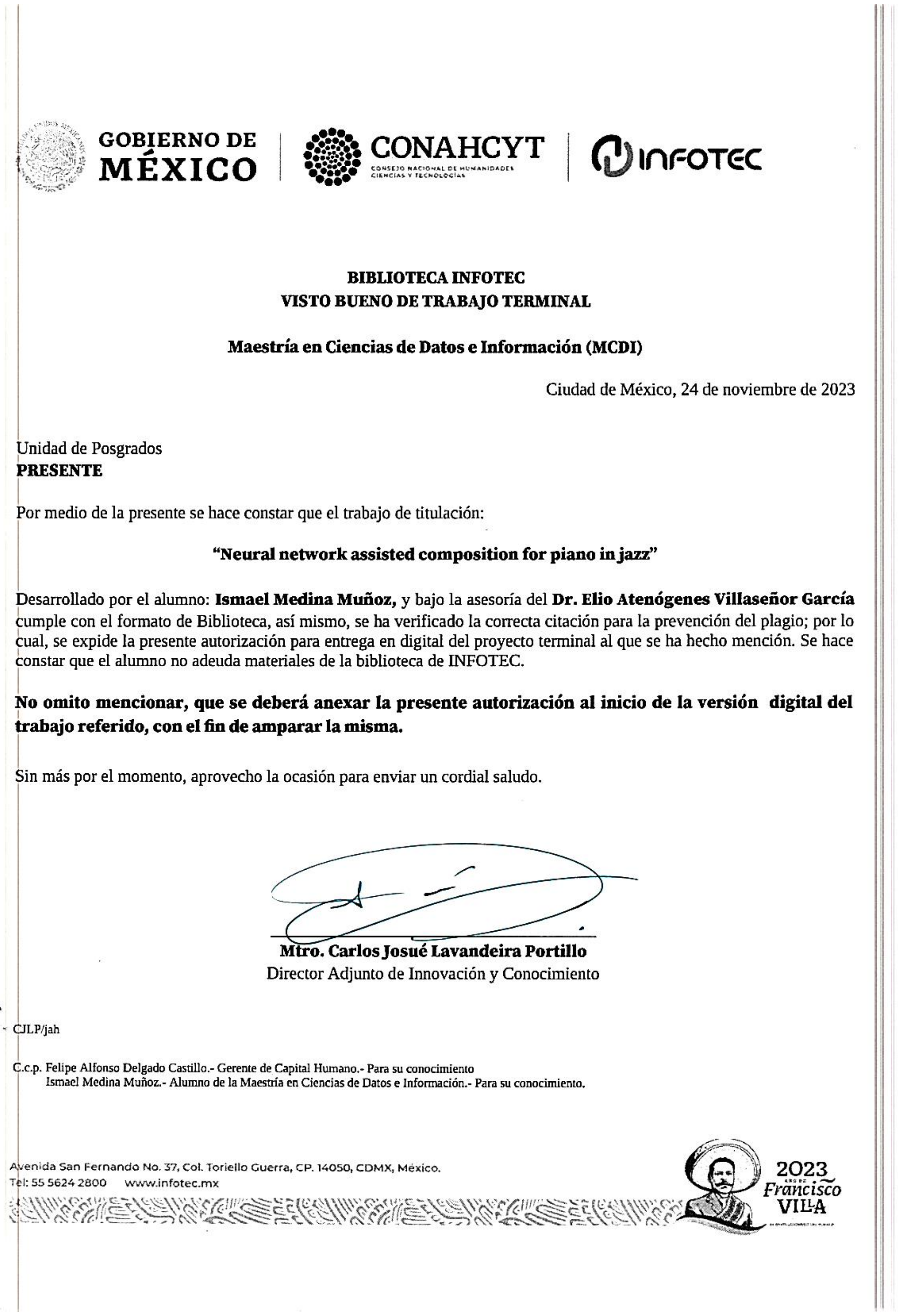
Dr. Elio Atenógenes Villaseñor García

Ciudad de México, Febrero, 2023

CONACYT

INFOTEC POSGRADOS

Printing Authorization



GOBIERNO DE MÉXICO | CONAHCYT CONSEJO NACIONAL DE HUMANIDADES CIENCIAS Y TECNOLOGÍAS | INFOTEC

BIBLIOTECA INFOTEC
VISTO BUENO DE TRABAJO TERMINAL

Maestría en Ciencias de Datos e Información (MCDI)

Ciudad de México, 24 de noviembre de 2023

Unidad de Posgrados
PRESENTE

Por medio de la presente se hace constar que el trabajo de titulación:

"Neural network assisted composition for piano in jazz"

Desarrollado por el alumno: **Ismael Medina Muñoz,** y bajo la asesoría del **Dr. Elio Atenógenes Villaseñor García** cumple con el formato de Biblioteca, así mismo, se ha verificado la correcta citación para la prevención del plagio; por lo cual, se expide la presente autorización para entrega en digital del proyecto terminal al que se ha hecho mención. Se hace constar que el alumno no adeuda materiales de la biblioteca de INFOTEC.

No omito mencionar, que se deberá anexar la presente autorización al inicio de la versión digital del trabajo referido, con el fin de amparar la misma.

Sin más por el momento, aprovecho la ocasión para enviar un cordial saludo.

[illegible]

Mtro. Carlos Josué Lavandeira Portillo
Director Adjunto de Innovación y Conocimiento

Jah

CJLP/jah

C.c.p. Felipe Alfonso Delgado Castillo.- Gerente de Capital Humano.- Para su conocimiento
Ismael Medina Muñoz.- Alumno de la Maestría en Ciencias de Datos e Información.- Para su conocimiento.

Dedications

To my beloved wife, Eugenia. Your transformation inspires me to keep going. Thank you for teaching me that change is the only constant.

To Yen, Joshua, and Mateo, my dear sons. Knowledge will bring you closer to freedom. I love you.

To my mother, my father, my brother Edgar, and his family, thank you for bringing joy to my life.

To my family, for being my support and for understanding my absence.

Contents

List of Figures

List of Tables

Abbreviations and acronyms

(NLP):	Natural language processing
(GPT):	Generative pre-trained transformer
(pmf):	Probability Mass Function
(GPU):	Graphics processing unit
(vCPU):	virtual CPU

Glossary

analogical signal Continuous signal representing some other quantity, i.e., analogous to another quantity. For example, in an analog audio signal, the instantaneous signal voltage varies continuously with the pressure of the sound waves [16].

artificial intelligence It's the capability of a computer system to mimic human-like cognitive functions such as learning and problem solving [9].

corpus Musical Instrument Digital Interface. Is a technical standard that describes a communications protocol, digital interface, and electrical connectors that connect a wide variety of electronic musical instruments, computers, and related audio devices for playing, editing, and recording music [17].

corpus Large set of data with certain organization, it can be for papers, scientific texts, literary texts, etc. The corpus can be considered as the starting point for scientific investigation.

F clef Also known as bass clef, the F clef on fourth line is a referring point that sets the F note over the fourth line of a staff. All the notes can then be placed in relation to this position. So, G note will be in the space between fourth and fifth line.

G clef Also known as treble clef, the G clef on second line is a referring point that sets the G note over the second line of a staff. All the notes can then be placed in relation to this position. So, A note will be in the space between second and third line.

GPT Generative pre-trained transformer (GPT) stands for a series of pre-trained language models (PLM) developed by OpenAI..., which has been the most popular type of transformers in NLG tasks. PLMs are language models that have been trained with a large dataset of textual information and can be applied to deal with specific language-related tasks [18].

harmonic sequence In tonal music, the harmonic sequence, as accompaniment for a melody, is a motivic pattern of two or more harmonies in succession that is restated in transposition, usually twice or three times, preserving the same melodic shape (relative motion) of each part or voice. By creating harmonic and tonal variety with a unified pattern, the sequence serves as a means of musical development [2].

Jazz Jazz is a kind of music in which improvisation is typically an important part. In most jazz performances, players play solos which they make up on the spot, which requires considerable skill... In jazz, you may hear the sounds of freedom-for the music has been a powerful voice for people suffering unfair treatment because of the color of the skin, or because they lived in a country run by a cruel dictator [11].

JSON Java Script Object Notation (JSON). JSON is a format for storing and transporting data. Often used when data is sent from a server to a web page.

KDE A kernel density estimate (KDE) plot is a method for visualizing the distribution of observations in a dataset, analogous to a histogram. KDE represents the data using a continuous probability density curve in one or more dimensions. [15].

measures A measure is segment inside a staff that describes musical notation for a group of notes defined by the time signature. For example, a measure for a 4/4 time signature indicates that the measure will describe 4 notes with duration of 1/4 each inside it. All musical notation should be constrained inside the measure following the expected elements described by the time signature.

melodic sequence A melodic or chordal figure repeated at a new pitch level (that is, transposed), thus unifying and developing musical material [2].

music staves The main component for music notation. They consist of 5 lines and 4 spaces where musical notation is placed to describe a tune.

NLP Natural language processing (NLP) refers to the branch of computer science—and more specifically, the branch of artificial intelligence or AI—concerned with giving computers the ability to understand text and spoken words in much the same way human beings can [5].

pmf Probability Mass Function of a discrete random variable X gives the probability that the variable takes a value. The function p is defined as $p : \mathscr{R} \rightarrow [0,1]$.

regular expressions A regular expression is a pattern that the regular expression engine attempts to match in input text. A pattern consists of one or more character literals, operators, or constructs [7].

REST API In 2000, Roy Fielding proposed Representational State Transfer (REST) as an architectural approach to designing web services. REST is an architectural style for building distributed systems based on hypermedia. REST is independent of any underlying protocol and is not necessarily tied to HTTP. [8].

softmax Function that converts a vector into a probability distribution. It is used to produce a vector of probability values.

theme It is the main idea of a music tune. The theme makes a music tune a unique piece. Some jazz tunes themes were played based on how the player imagined the singer voice would play a written text but using the instrument voice. The theme will be heard as a repetition of the same music idea during the tune execution.

Introduction

Artificial Intelligence has taken an important role in activities that were once considered exclusively human. Generative AI is a vibrant area of research, with increasing interest in application fields related to the arts. The recent plethora of innovations in fields like visual arts and natural language processing, which are able to engage in dialogue with users, are just two examples of commercial applications that are driving innovation research for big tech giants. It would not be untrue to say that these innovations are shaping mankind's development.

Music is an investigative field that presents a challenge. Musical theory itself is challenging for humans, and music is as diverse and rich as the cultures in which it has evolved. This research and proposal is intended as a novel approach to creating a generative artificial intelligence that assists in piano composition for jazz tunes. This genre was selected because of the challenge that its richness and complexity for musical execution and interpretation pose.

By using a Recurrent Neural Net to create new sequences of n-notes from an initial n-note set and using a probabilistic approach to set the duration of each note in the produced n-notes set, the generative artificial intelligence described in this document is the piano composer assistant for jazz tunes.

Chapter 1

Music and Artificial Intelligence

1 Music and Artificial Intelligence

As Morán states, "The creation, performance, and appreciation of music obey the superior human ability to discover sound patterns and identify them on subsequent occasions. Without the biological processes of auditory perception and without a cultural consensus on what is perceived, among at least some listeners, neither music nor musical communication can exist" [10].

The human ability for communication is the origin of musical genres like Jazz. There are several efforts to bring artificial intelligence to the music industry. Creativity is an interesting field. Large projects like Google Magenta, which is intended to extend the musical creation process instead of replacing it [13], are some of the most cited example of those efforts. Another interesting example is the one called MuseNet from OpenAI based on GPT architecture. It has some interesting capabilities, like multi-instrument music generation based on existing music styles [12]. This project has the main purpose of assisting the creative process for jazz piano players.

1.1 Problem definition

As the development of audiovisual content for traditional and novel broadcasting platforms such as TV and the internet increases, musical support is required to achieve its purpose. The demand for new tunes has increased due to the rise in content creation. However, the high demand poses a challenge for musical creation due to human restrictions, such as the time required to create new musical tunes and the constant need for new music ideas to develop. The creative process can be slowed down by the lack of either of these two elements.

Having an artificial intelligence that can generate new ideas in a reduced amount of time would result in the immediate application of the project by accelerating musical creation based on chunks of melodic sequence and/or harmonic sequence to be used as a base in the composition process. Interaction with this assistant can be monetized for every single call that users trigger against it based on REST API architecture.

1.2 Objectives

1.2.1 General objectives

The objective is to create a generative artificial intelligence trained using already created jazz tunes to generate music staves for the piano. It will generate sequences for one staff in G clef and a second staff for F clef, as used by piano players. Music sequences will be described in several measures. The generative artificial intelligence will be deployed as a REST API. A further development could be the integration of a desktop and/or mobile app to consume the REST API.

The jazz tunes to be used to train the generative artificial intelligence are the ones publicly available at MuseScore for jazz tunes that contain piano parts.

1.2.2 Specific objectives

- To get a large score corpus to train the generative artificial intelligence.
- To clean the corpus to extract useful data for coherent training.
- To perform exploratory data analysis over score corpus.
- To train the generative artificial intelligence based in Recurrent Neural Networks as if it was a two-handed piano player.
- To publish the model as a REST API.
- To overcome already identified issues related to artificial intelligence's that creates music going rapidly into monotony and / or loops.
- To imitate aspects of human execution like improvisation.

1.3 Contribution

This project aims to create a foundation for others interested in creating a generative artificial intelligence for other musical genres or extending the capabilities of the generative artificial intelligence to improve its output.

Chapter 2

Theoretical Framework

2 Theoretical Framework

The second chapter aims to provide the reader with the foundational knowledge necessary to understand the elements of the proposed solution to the problem described in Chapter 1.

2.1 The problem of music generation as a NLP problem

Natural Language Processing (NLP) is a branch of artificial intelligence used to provide language processing capabilities to computers. NLP algorithms, techniques, and architectures are well-known for their powerful approach to solving some of the most challenging problems related to language. Text generation and word prediction based on a previous sequence of words are just two examples of problems solved by NLP.

Music is a language in itself, and the project uses the same approach used to predict the next word in a word sequence. Written music consists of a group of notes instead of a group of characters. Predicting the next word in a sequence is a task that Recurrent Neural Nets (RNN) are efficient for.

A convenient starting point is the general description of an RNN provided by Skansi [14], <<Feedforward neural networks can process vectors, and convolutional neural networks can process matrices (which are translated into vectors). How would we process sequences of unequal length? If we are talking about, e.g. images of different sizes, then we could simply re-scale them to match... Note, that if all matrices, we analyse are of the same size they can be represented by long vectors... If they vary in size, we cannot encode them as vectors and keep the nice properties... the real problem is how to fit vectors of different dimensions... in a neural network. Everything we have seen so far, needs a fixed-dimensional vectors. The problem of varying dimensionality can be seen as the problem of learning sequences of unequal length, and audio processing is a nice example of how we might need this, since various audio clips are necessarily of different

lengths. We could in theory just take the longest and then make all others of the same length as that one, but this is waste in terms of the space needed. But there is a deeper problem here. Silence is a part of language, and it is often used for communicating meaning, so a sound clip with some content labeled with the label 1 in the training set might be correct, but if add 10s of silence at the beginning or the end of the clip, the label 1 might not be appropriate anymore, since the clip with the silence may have a different meaning. Think about irony, sarcasm and similar phenomena. So the question is what we can do? The answer is that we need a different neural network architecture than we have seen before. Every neural network we have seen so far has connections which push the information forward, and this is why we have called them 'feedforward neural networks'. It will turn out that by having connections that feed the output back into a layer as inputs, we can process sequences of unequal length. This makes the network deep, but it does share weights so it partly avoids the vanishing gradient problem. Networks that have such feedback loops are called recurrent neural networks>>.

From this description, a detailed definition will be easier to understand. The definition provided by Goldberg [3] states, <<We use $\mathbf{x_{i:j}}$ to denote the sequence of vectors $\mathbf{x_i}, ..., \mathbf{x_j}$. On a high-level, the RNN is a function that takes as input an arbitrary length ordered sequence of n d_{in}-dimensional vectors $\mathbf{x_{1:n}} = \mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}$, $(\mathbf{x_i} \in \mathbb{R}^{d_{in}})$ and returns as output a single d_{out} dimensional vector $\mathbf{y_n} \in \mathbb{R}^{d_{out}}$:

$$\mathbf{y_n} = \mathrm{RNN}(\mathbf{x_{1:n}})$$

$$\mathbf{x_i} \in \mathbb{R}^{d_{in}} \qquad \mathbf{y_n} \in \mathbb{R}^{d_{out}}.$$

This implicitly defines an output vector $\mathbf{y_i}$ for each prefix $\mathbf{x_{1:i}}$ of the sequence $\mathbf{x_{1:n}}$. We denote by RNN* the function returning this sequence:

$$\mathbf{y_{1:n}} = \mathrm{RNN}^*(\mathbf{x_{1:n}})$$

$$\mathbf{y_i} = \mathrm{RNN}(\mathbf{x}_{1_i})$$

$$\mathbf{x_i} \in \mathbb{R}^{d_{in}} \qquad \mathbf{y_i} \in \mathbb{R}^{d_{out}}.$$

The output vector $\mathbf{y_n}$ is then used for further prediction. For example, a model for predicting the conditional probability of an event e given the sequence $\mathbf{x_{1:n}}$ can be defined as $p(e = j|\mathbf{x_{1:n}}) = \text{softmax}(\text{RNN}(\mathbf{x_{1:n}}) \cdot \mathbf{W} + \mathbf{b})_{[j]}$, the jth element in the output vector resulting from the softmax operation over a linear transformation of the RNN encoding $\mathbf{y_n} = \text{RNN}(\mathbf{x_{1:n}})$. The RNN function provides a framework for conditioning on the entire history $\mathbf{x_1}, \dots, \mathbf{x_i}$ without resorting to the Markov assumption which is traditionally used for modeling sequences... Indeed, RNN-based language models result in very good perplexity scores when compared to ngram-based models. Looking in a bit more detail, the RNN is defined recursively, by means of a function R taking as input a state vector $\mathbf{s_{i-1}}$ and an input vector $\mathbf{x_i}$ and returning a new state vector $\mathbf{s_i}$. The state vector $\mathbf{s_i}$ is then mapped to an output vector $\mathbf{y_i}$ using a simple deterministic function $O(\cdot)$... The base of the recursion is an initial state vector, $\mathbf{s_0}$, which is also an input to the RNN. For brevity, we often omit the initial vector $\mathbf{s_0}$, or assume it is the zero vector. When constructing an RNN, much like when constructing a feed-forward network, one has to specify the dimension of the inputs $\mathbf{x_i}$ as well as the dimensions of the outputs $\mathbf{y_i}$. The dimensions of the states $\mathbf{s_i}$ are a function of the output dimension...

$$\text{RNN}^*(\mathbf{x_{1:n}}; \mathbf{s_0}) = \mathbf{y_{1:n}}$$

$$\mathbf{y_i} = O(\mathbf{s_i})$$

$$\mathbf{s_i} = R(\mathbf{s_{i-1}}, \mathbf{x_i})$$

$$\mathbf{x_i} \in \mathbb{R}^{d_{in}} \qquad \mathbf{y_i} \in \mathbb{R}^{d_{out}} \qquad \mathbf{s_i} \in \mathbb{R}^{f(d_{out})}$$

The functions R and O are the same across the sequence positions, but the RNN keeps track of the states of computation through the state vector s_i that is kept and being passed across invocations of R.>>

Goldberg presents a figure to represent a recursive RNN graphically. This is figure number 2.1.

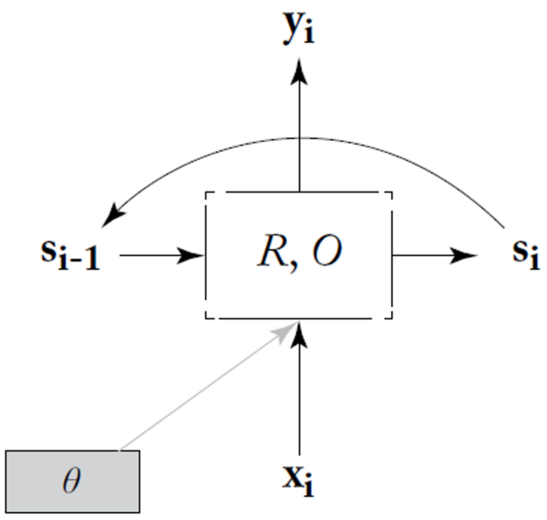


Figure 2.1: Graphical representation of an RNN (recursive).

Goldberg continues describing RNNs, stating that, <<This presentation follows the recursive definition, and is correct for arbitrarily long sequences. However, for a finite sized input sequence (and all input sequences we deal with are finite) one can *unroll* the recursion>>

The structure described by Goldberg is the following.

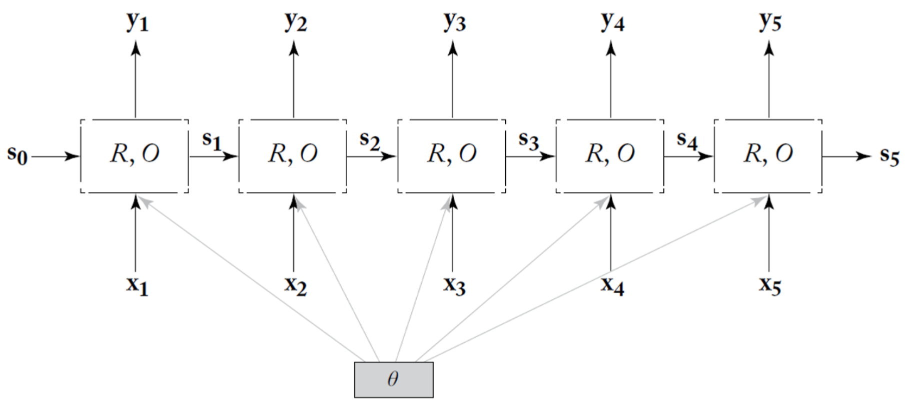


Figure 2.2: Graphical representation of an RNN (unrolled).

Lastly, Goldberg finalizes the description by stating that, <<While not usually shown in the visualization, we include here the parameters θ in order to highlight the fact that the same parameters are shared across all time steps. Different instantiations of R and O will result in different network structures, and will exhibit different properties in terms of their running times and their ability to be trained

effectively using gradient-based methods. However, they all adhere to the same abstract interface. We will provide details of concrete instantiations of R and O– the Simple RNN, the LSTM, and the GRU-... Before that, let's consider working with the RNN abstraction. First, we note that the value of $\mathbf{s_i}$ (and hence $\mathbf{y_i}$) is based on the entire input $\mathbf{x_1},...,\mathbf{x_i}$. For example, by expanding the recursion for $i = 4$ we get:

$$
\begin{aligned}
\mathbf{s_4} &= R(\mathbf{s_3},\mathbf{s_4}) \\
&= R(R(\mathbf{s_2},\mathbf{x_3}),\mathbf{x_4}) \\
&= R(R(R(\mathbf{s_1},\mathbf{x_2}),\mathbf{x_3}),\mathbf{x_4}) \\
&= R(R(R(R(\mathbf{s_0},\mathbf{x_1}),\mathbf{x_2}),\mathbf{x_3}),\mathbf{x_4})
\end{aligned}
$$

Thus, $\mathbf{s_n}$ and $\mathbf{y_n}$ can be thought of as *encoding* the entire input sequence... Is the encoding useful? This depends on our definition of usefulness. The job of the network training is to set the parameters of R and O such that the state conveys useful information for the task we are trying to solve>>.

For this project, a Long Short-Term Memory (LSTM) network and Gated Recurrent Unit (GRU) network were implemented. The explanation of them is described in the following subsections.

2.1.1 Long Short-Term Memory (LSTM) network

The synthesized definition of an LSTM given by Jurafsky [6] will be the starting point for the reader to identify its usefulness in solving the problem this project is for. He states that, <<LSTM divides the context management problem into two sub-problems: removing information no longer needed from the context, and adding information likely to be needed for later decision making. The key to solving both problems is to learn how to manage this context rather than hard-coding a strategy into the architecture. LSTMs accomplish this by first adding an explicit context layer to the architecture (in addition to the usual recurrent hidden layer), and through the use of specialized neural units that make use of gates to control the flow of infor-

mation into and out of the units that comprise the network layers. These gates are implemented through the use of additional weights that operate sequentially on the input, and previous hidden layer, and previous context layers. The gates in an LSTM share a common design pattern; each consists of a feedforward layer, followed by a sigmoid activation function, followed by a pointwise multiplication with the layer being gated. The choice of the sigmoid as the activation function arises from its tendency to push its outputs to either 0 or 1. Combining this with a pointwise multiplication has an effect similar to that of a binary mask. Values in the layer being gated that align with values near 1 in the mask are passed through nearly unchanged; values corresponding to lower values are essentially erased. The first gate we'll consider is the **forget gate**. The purpose of this gate to delete information from the context that is no longer needed. The forget gate computes a weighted sum of the previous state's hidden layer and the current input and passes that through a sigmoid. This mask is then multiplied by the context vector to remove the information from context that is no longer required.

$$f_t = \sigma(U_f h_{t-1} + W_f x_t)$$

$$k_t = c_{t-1} \bigodot f_t$$

The next task is compute the actual information we need to extract from the previous hidden state and current inputs — the same basic computation we've been using for all our recurrent networks.

$$g_t = \tanh(U_g ht - 1 + W_g x_t)$$

Next, we generate the mask for the **add gate** to select the information to add to the current context.

$$i_t = \sigma(U_i h_{t-1} + W_i x_t)$$

$$j_t = g_t \bigodot i_t$$

Next, we add this to the modified context vector to get our new context vector.

$$c_t = j_t + k_t$$

The final gate we'll use is the **output gate** which is used to decide what information is required for the current hidden state (as opposed to what information needs to be preserved for future decisions).

$$o_t = \sigma(U_o h_{t-1} + W_o x_t)$$

$$h_t = o_t \odot \tanh(c_t)$$

... Given the appropriate weights for the various gates, an LSTM accepts as input the context layer, and hidden layer from the previous time step, along with the current input vector. It then generates updated context and hidden vectors as output. The hidden layer, ht , can be used as input to subsequent layers in a stacked RNN, or to generate an output for the final layer of a network.>> Its capability to forget what is no longer relevant was used to produce a LSTM model to be compared against a GRU model. The model to be published in the web service would be the one with higher performance.

2.1.2 Gated Recurrent Unit (GRU) network

For GRU, a great description is the one from Jurafsky [6]. He states that, <<LSTMs introduce a considerable number of additional parameters to our recurrent networks. We now have 8 sets of weights to learn (i.e., the U and W for each of the 4 gates within each unit), whereas with simple recurrent units we only had 2. Training these additional parameters imposes a much significantly higher training cost. Gated Recurrent Units (GRUs)... ease this burden by dispensing with the use of a separate context vector, and by reducing the number of gates to 2 - a reset gate, r and an update gate, z.

$$r_t = \sigma(U_r h_{t-1} + W_r x_t)$$

$$z_t = \sigma(U_z h_{t-1} + W_z x_t)$$

As with LSTMs, the use of the sigmoid in the design of these gates results in a binary-like mask that either blocks information with values near zero or allows information to pass through unchanged with values near one. The purpose of the reset gate is to decide which aspects of the previous hidden state are relevant to the current context and what can be ignored. This is accomplished by performing an element-wise multiplication of r with the value of the previous hidden state. We then use this masked value in computing an intermediate representation for the new hidden state at time t.

$$\tilde{h}_t = \tanh(U(r_t \odot h_{t_1}) + W_{x_t})$$

The job of the update gate z is to determine which aspects of this new state will be used directly in the new hidden state and which aspects of the previous state need to be preserved for future use. This is accomplished by using the values in z to interpolate between the old hidden state and the new one.

$$h_t = (1 - z_t)h_{t-1} + z_t\tilde{h}_t$$

>>

Both LSTM and GRU are useful for model training, and either can be used to create a deployable model.

2.1.3 Comparing LSTM and GRU

LSTMs and GRUs have differences in the way they handle context retention and memory complexities. GRUs are lighter to process than LSTMs because of the context vector at the input and output. Figure 2.3 shows a basic comparison of the LSTM memory unit (c) and GRU (d), the basic feedforward unit (a), and the unit for a simple recurrent network (b), as described by Jurafsky [6].

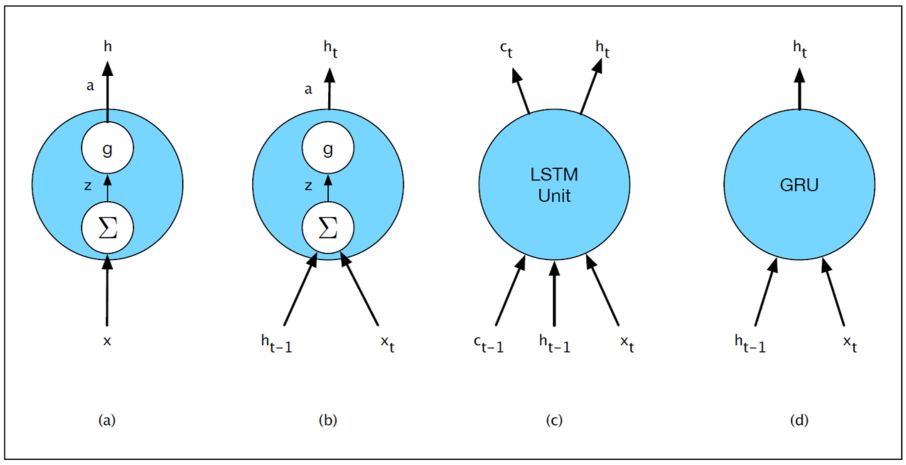


Figure 2.3: Basic neural units.

Chapter 3

Data Collection and Exploratory Data Analysis

3 Data Collection and Exploratory Data Analysis

The third chapter aims to explain to the reader how the corpus was built and the exploratory data analysis process. To address the problem described in Chapter 1, it was necessary to have a corpus of piano jazz music.

3.1 The Data Collection Challenge

To create a useful corpus of piano jazz tunes to train the neural net, the first requirement was to have a dataset. There are several possible sources for this, such as audio files, but this would raise the more challenging issue of separating the piano waves from waves of other instruments in the music tune, with a high chance of getting an output audio with low quality and with a high amount of noise and imperfection. Having the music in a digital representation was the best alternative data source.

A well-structured way to store data digitally is the corpus format. The main challenge there is that the structure is intended for producing audio with no need to be human-readable. Musical devices that create MIDI native data or devices that digitalize analogical signal to produce a MIDI representation are examples of how the representation of music can be created. MIDI is not constrained by music notation rules, and this increases the complexity for the task of corpus creation and analysis.

The third option was to collect data from music score representations created using specialized software. The following challenge was to identify specialized software for music writing that exposed some interface to programmatically read the files it produces and to get a huge amount of files to be digested. MuseScore.org created a free music composition and notation software built by developers, contributors, and a user community. The community supports MuseScore.com, which allows users to store and share music sheets created using the software. Subscribers of MuseScore.com can get a Pro subscription to download all public music sheets with no restriction.

The decision made to get a dataset to build the corpus to train the neural net was to create a web scraper to get music sheets using a Pro subscription based on the Python language. The described web scraper is based on the Musescore-Web-Crawler project from Arevalo [1]. On the MuseScore.com website, you are allowed to browse music sheets by music genre and instrument. The browser URL is defined by a set of parameters that can be introduced in web scraper calls.

Once you log in to the community website using an already created Pro subscription, you will be able to browse the music sheets shared by the community. This call is performed using the Selenium WebDriver library for Python. The login process implemented is able to click on the "login" button using the element id in the page content. The login data needs to be introduced manually. The code for the call is the following.

```
from selenium import webdriver
from selenium.webdriver.common.by import By
driver = webdriver.Edge()
button = driver.find_element(By.XPATH, "//button[@class='_30GTb␣
    ↪ _30GTb␣_39f0R␣_3Afie␣_11AeI␣_1ZXw8␣_3zmA3␣_2wqMT']")
button.click()
```

The browsing pages contain a list of music sheets that can be refined using several filters. Each browsing page shows this list, and you can retrieve the elements in the content page using regular expressions. Figure 3.1 shows a MuseScore.org browse page and the filters for jazz piano tunes and the music sheet list.

The code that shows the browsing page iteration and the regular expression to retrieve the collection of the music sheets is the following.

```
url_template_head = 'https://musescore.com/sheetmusic?genres=84&
    ↪ instrument=2&instrumentation=114&page='
url_template_tail = '&recording_type=public-domain'


driver.get("https://musescore.com/sheetmusic?genres=84&instrument=2&
    ↪ instrumentation=114&recording_type=public-domain")
```

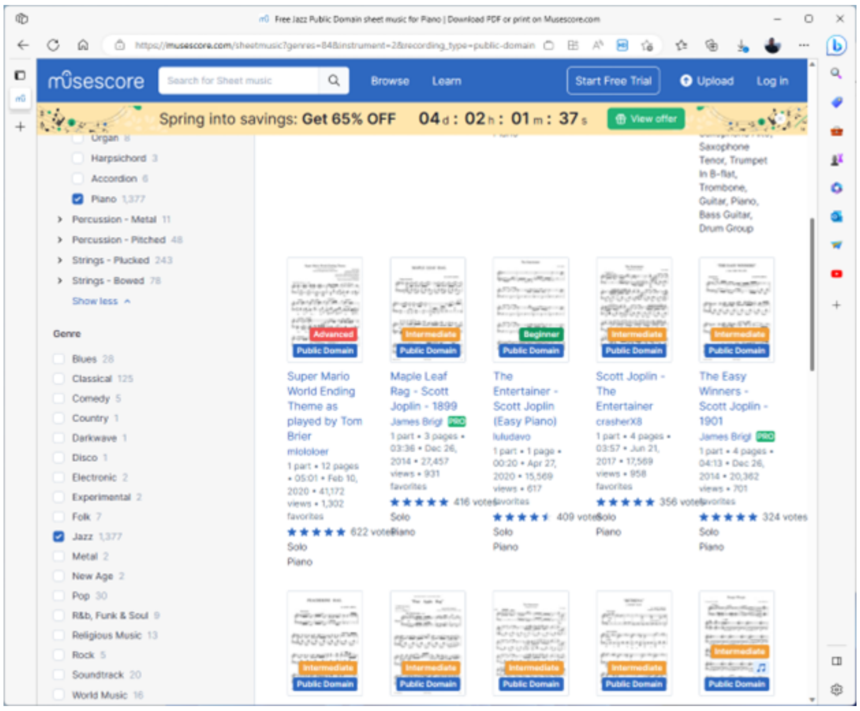

Free Jazz Public Domain sheet music for Piano | Download PDF or print on Musescore.com
https://musescore.com/sheetmusic?genres=84&instrument=2&recording_type=public-domain
musescore
Search for Sheet music
Browse
Learn
Start Free Trial
Upload
Log in
Spring into savings: Get 65% OFF
View offer
Organ
Harpsichord
Accordion
Piano 1,377
Percussion - Metal
Percussion - Pitched
Strings - Plucked
Strings - Bowed
Show less
Genre
Blues
Classical
Comedy
Country
Darkwave
Disco
Electronic
Experimental
Folk
Jazz 1,377
Metal
New Age
Pop
R&b, Funk & Soul
Religious Music
Rock
Soundtrack
World Music
Super Mario World Ending Theme as played by Tom Brier
Maple Leaf Rag - Scott Joplin - 1899
The Entertainer - Scott Joplin (Easy Piano)
Scott Joplin - The Entertainer
The Easy Winners - Scott Joplin - 1901
Public Domain
Advanced
Intermediate
Beginner

Figure 3.1: A MuseScore.org browse page and the filters for jazz piano tunes.

```
content = driver.page_source

score_urls = set(re.findall(r"https://musescore.com/user/\d+/scores/\
    ↪ d+", content))

for i in range(2, 39):
    url_to_save = url_template_head + str(i) + url_template_tail
    driver.get(url_to_save)
    content = driver.page_source
    temp_urls = set(re.findall(r"https://musescore.com/user/\d+/
        ↪ scores/\d+", content))
    score_urls.update(temp_urls)
```

Notice the URL query built on the `url_template_head` variable contains the `genres`, the `instrument`, the `instrumentation`, and the `page` parameters. The parameter `page` is dynamic and changes on each cycle. The last part of the URL query filters *public-domain* music sheets using the `url_template_tail` variable. The `content` variable is for the page content, this content is parsed to get the scores. The regular expression pattern to be captured is `r"https://musescore.com/user/d+/scores/d+"`.

Once the list is built using the `score_urls` variable, iterating each score URL is performed using the following code:

```
successes = []
for page in score_urls:
    try:
        driver.get(page)
        downloaddialog = driver.find_element(By.XPATH, "//button[
            ↪ @class='_30GTb␣_30GTb␣_3aHmn␣_17bLW␣_39f0R␣_3Afie␣
            ↪ _3CiIP␣_1ZXw8␣_33wzU␣_2wqMT']")
        downloaddialog.click()
        time.sleep(2)
        scorebutton = driver.find_element(By.XPATH, "//button[@class='
            ↪ _39f0R␣_3Afie␣_1EgpX␣_2zKSV␣_2wqMT']")
        scorebutton.click()
        print(".", end= '')
        time.sleep(3)
        if (driver.page_source.__contains__('Page␣not␣found') and
            ↪ driver.page_source.__contains__('Sorry␣about␣that.')):
            print("#", end= '')
            continue
        else:
            successes.append(page)
    except:
        print("!", end= '')
        time.sleep(2)
        continue
```

The previous code shows that the first step is to load the music sheet page, and the first activated button is the "Download" button referenced by the `downloaddialog` variable. Figure 3.2 shows the objects and the "Download" button inside the iterated page. Figure 3.3 shows the "download" dialog frame and the "MuseScore" file download button.

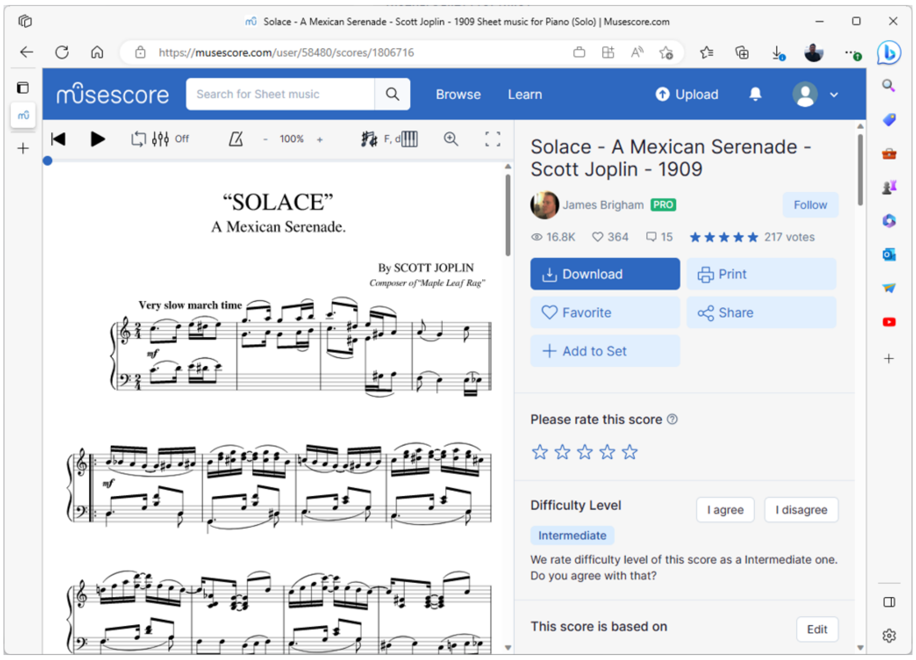

Solace - A Mexican Serenade - Scott Joplin - 1909 Sheet music for Piano (Solo) | Musescore.com
https://musescore.com/user/58480/scores/1806716
musescore
Search for Sheet music
Browse
Learn
Upload
Solace - A Mexican Serenade - Scott Joplin - 1909
James Brigham PRO
Follow
16.8K 364 15 217 votes
Download
Print
Favorite
Share
Add to Set
Please rate this score
Difficulty Level
I agree
I disagree
Intermediate
We rate difficulty level of this score as a Intermediate one. Do you agree with that?
This score is based on
Edit
"SOLACE"
A Mexican Serenade.
By SCOTT JOPLIN
Composer of "Maple Leaf Rag"
Very slow march time

Figure 3.2: Music sheet page and the "download" button.

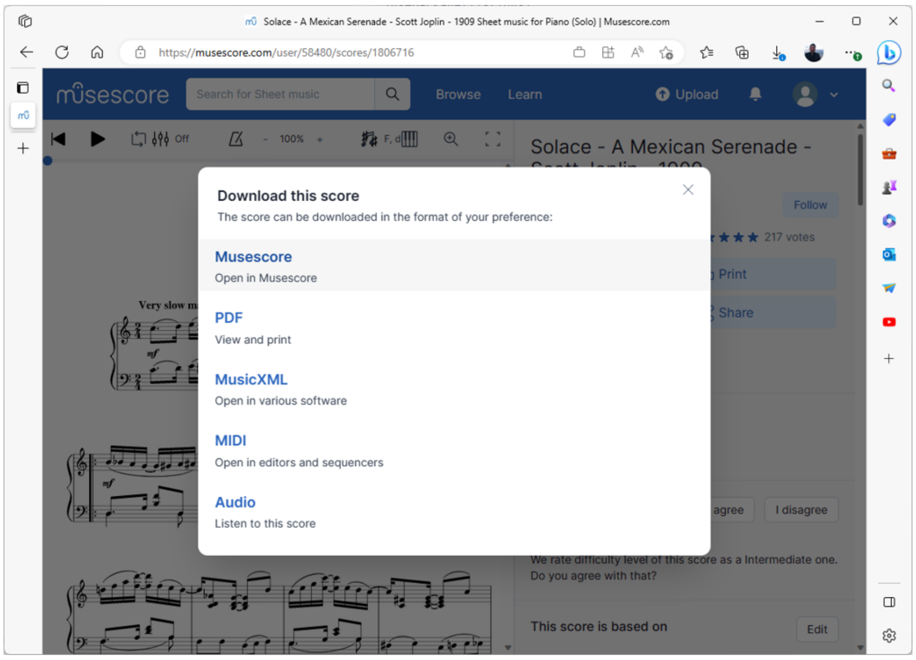
The second step is to click the "MuseScore" file download button. This action automatically saves the **MSCZ** file in the default "Downloads" folder for the Windows Operating System. This iterative process was run to get the data collection. The total number of files collected using this process was 634.

3.2 Exploratory Data Analysis

Exploring the data from music sheets presented several challenges. The journey of the exploratory data analysis of the dataset to build the corpus is described below.

3.2.1 Analysis of a Single Music Sheet

To demonstrate the concepts of music reading using Python, a convenient starting point is to learn how to read a single music sheet and explore its content. The library used to read MuseScore music sheets was **MS3**, developed by Johannes Hentschel and



Solace - A Mexican Serenade - Scott Joplin - 1909 Sheet music for Piano (Solo) | Musescore.com

https://musescore.com/user/58480/scores/1806716

Download this score

The score can be downloaded in the format of your preference:

Musescore
Open in Musescore

PDF
View and print

MusicXML
Open in various software

MIDI
Open in editors and sequencers

Audio
Listen to this score

We rate difficulty level of this score as a Intermediate one. Do you agree with that?

This score is based on

Figure 3.3: The "download" dialog frame and the "MuseScore" file download button.

publicly available at ms3. The library can open a single music sheet file using the **Score** object, which describes score metadata. The score is divided into parts (**Parts** object), each of which describes the staves (**Staves** object) associated with an instrument (**Instrument** object). This metadata allows the extraction of the set of notes and rests (**notes_and_rests**) as a `pandas.DataFrame` for the parts that describe the staves of an instrument.

The **notes_and_rests** pandas DataFrame describes a score using several columns. The columns used for the analysis are described below [4].

- **Measure Counts (mc)**. Measure count, identifier for the measure units in the XML encoding. Always starts with 1 for correspondence to MuseScore's status bar.
- **Measure Numbers (mn)**. Measure number, continuous count of complete measures as used in printed editions. Starts with 1 except for pieces beginning with a pickup measure, numbered as 0. MNs are identical for first and second endings!
- **Measure Count On Set (mc_onset)**. The value for `mc_onset` represents, expressed as fraction of a whole note, a position in a measure where **0** corresponds to the

earliest possible position (in most cases beat 1).

- **Measure Number On Set (mn_onset)**. The value for mn_onset represents, expressed as fraction of a whole note, a position in a measure where **0** corresponds to the earliest possible position of the corresponding measure number (MN).
- **Time Signatures(timesig)**. The time signature timesig of a particular measure is expressed as a string, e.g. '2/2'. The actual duration of a measure can deviate from the time signature for notational reasons: For example, a pickup bar could have an actual duration of 1/4 but still be part of a '3/8' meter, which usually has an actual duration of 3/8.
- **Staff (staff)**. In which staff an event occurs. 1 = upper staff.
- **Duration (duration)**. Duration of an event expressed in fractions of a whole note. Note that in note lists, the duration does not take into account if notes are tied together; in other words, the column expresses no durations that surpass the final bar line.
- **MIDI Piano key (midi)**. MIDI pitch with 60 = C4, 61 = C#4Db4B##3 etc.

The selected tune to be analyzed is **Danny Boy** from the **Tokyo Solo** played by **Keith Jarret** and transcribed in a MuseScore music sheet that was collected using the web scraping technique described previously. The music sheet can be found at Keith Jarrett - Danny Boy Londonderry Air Tokyo Solo 2002. The original live tune can be found at Keith Jarrett - Danny Boy (Londonderry Air).

A first hypothesis was that a piano player is a two-handed person with 5 fingers each. The piano player would be able to pulsate 10 piano keys at the same time. From now on, we will define a **time slice** as a set of music notes and rests that occur at the same time. To accept or reject the hypothesis, a grouping of midi notes by mc and mc_notes on each staff to produce column count_n_notes was generated. Figure 3.4 shows the histogram of pulsated keys in a single time slice for each staff.

A more detailed analysis adds a column to the grouping, the timesig. This grouping produces a new column called n-notes that represents the notes pulsed in a single

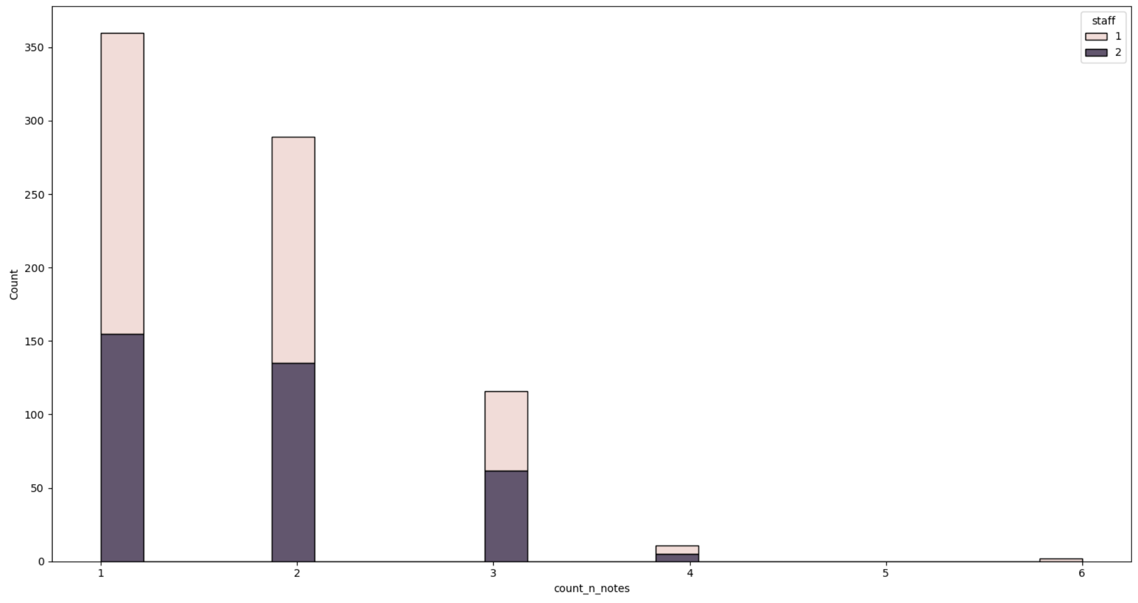


Figure 3.4: Amount of n-notes by time slice for each staff.

time slice for each time signature per staff. Figures 3.5 and 3.6 show this detailed analysis.

The three mentioned figures confirm that it is not usual to pulsate more than 5 keys by each staff in the same time slice. As each staff is associated with a single hand for the piano player, excluding any pulsated keys above the fifth won't have a higher impact on a summarization of the tune execution.

The second hypothesis is that each staff would confirm that the left hand, associated with staff 2, would play lower midi keys while the right hand, associated with staff 1, would play higher midi keys. Figure 3.7 shows that the hypothesis is true.

An interesting analysis is to identify the duration of the notes and show them in a histogram, as music is not just about notes but also about their duration. Figure 3.8 shows that some notes durations like $\frac{1}{8}$, $\frac{1}{4}$ and $\frac{1}{16}$ are the most common in the score.

A second interesting analysis is to identify the duration of the rests and show them in a histogram. Rests are also part of the music, and their duration cannot be left apart. Figure 3.9 shows that some rest durations like $\frac{1}{8}$, $\frac{1}{4}$ and $\frac{1}{2}$ are the most common in the score.

All the hypotheses and analyses performed against a single music sheet apply to all

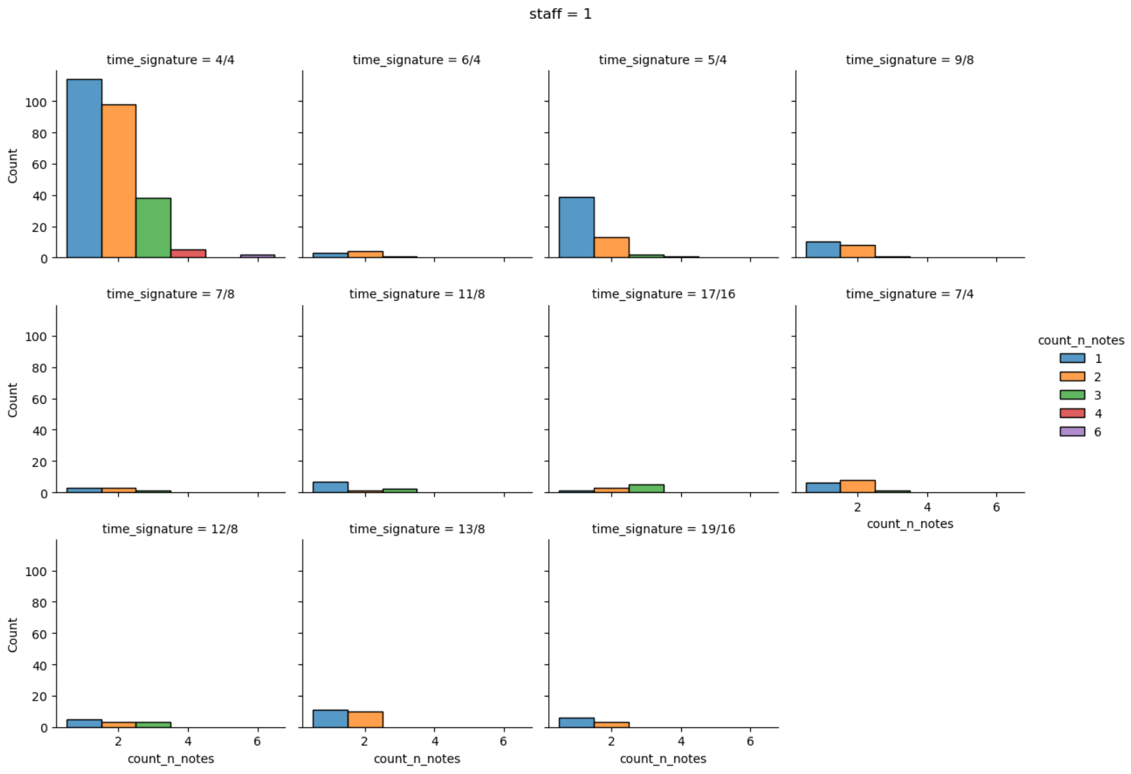


Figure 3.5: Amount of n-notes by time slice on staff 1 presented by time signature.

downloaded music sheets. These activities are going to be executed on the complete dataset in the following subsection.

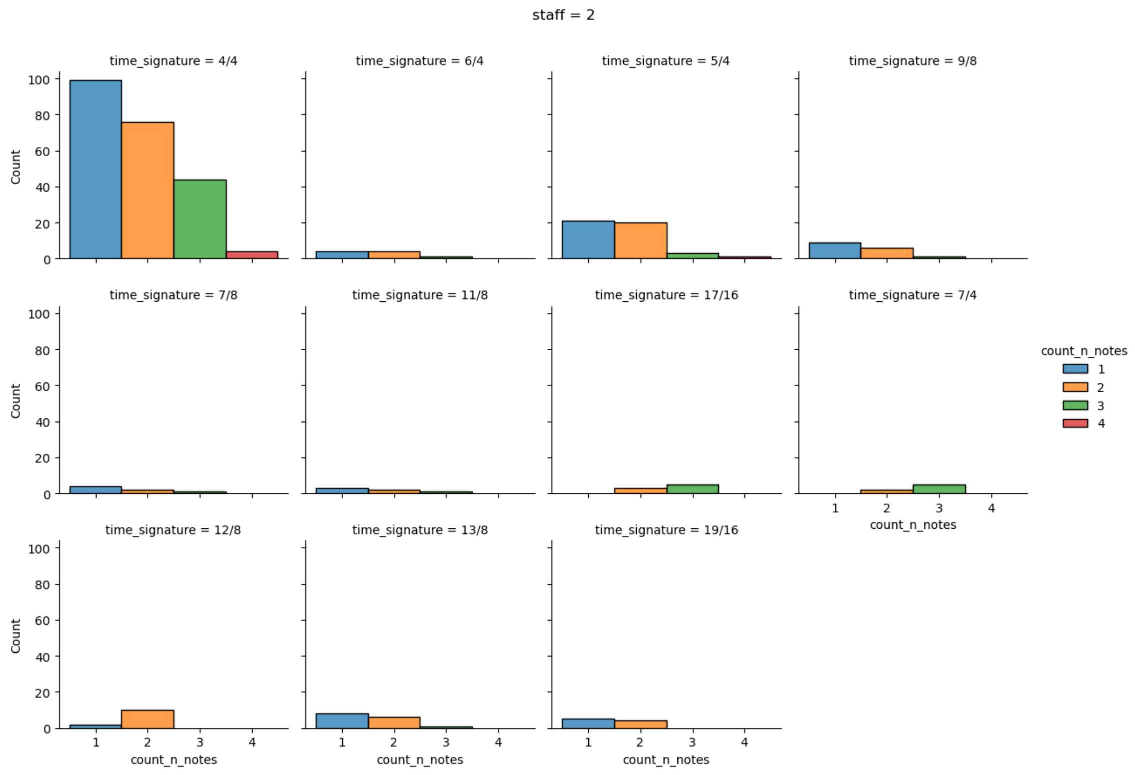


Figure 3.6: Amount of n-notes by time slice on staff 2 presented by time signature.

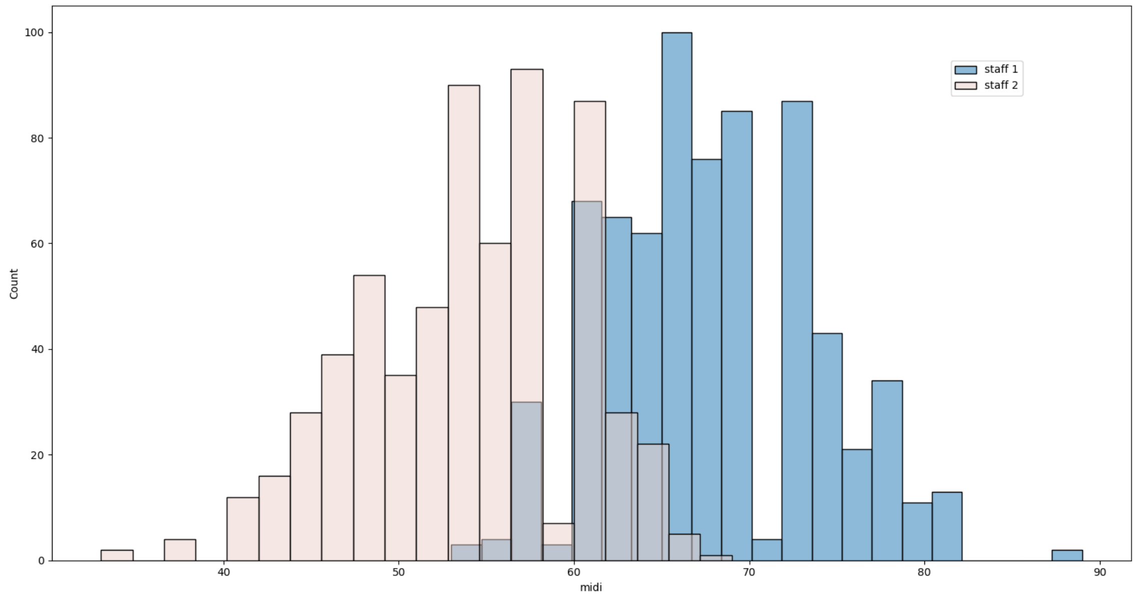


Figure 3.7: MIDI keys range for each staff.

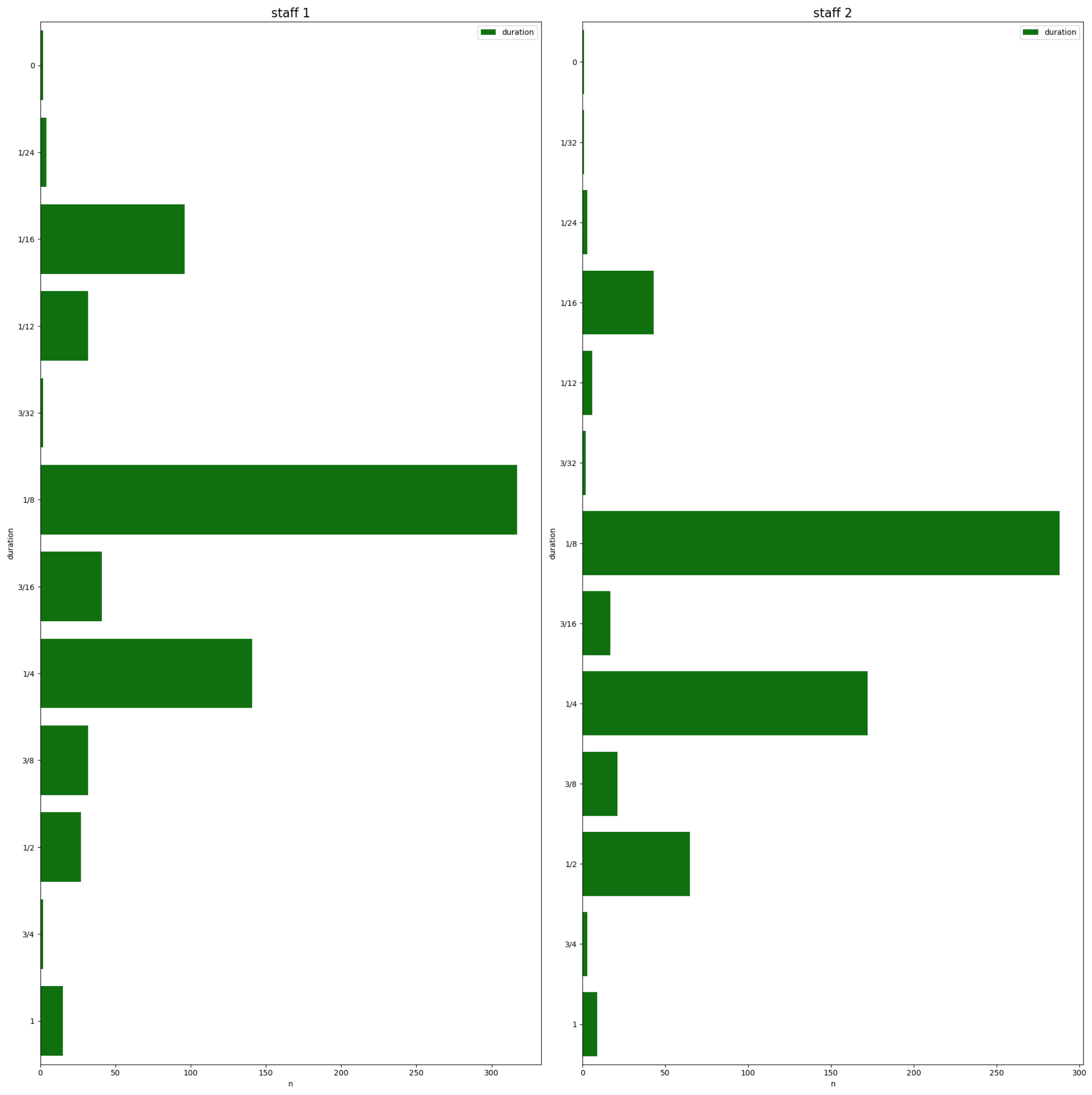


Figure 3.8: Histogram of note duration by staff.

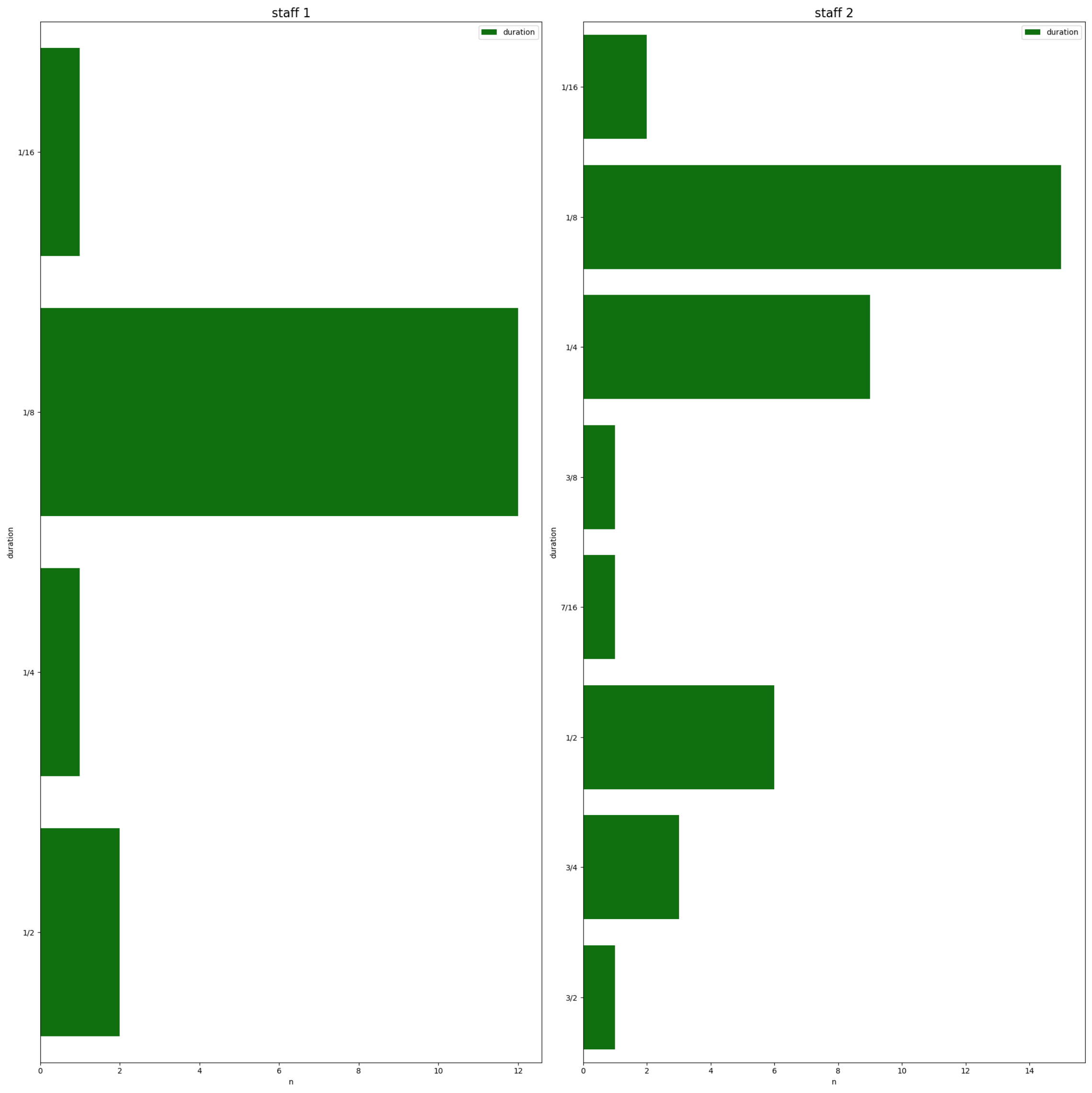


Figure 3.9: Histogram of rest duration by staff.

3.2.2 Full Data Set Analysis and Corpus Building

The set of 634 files was processed, but 172 files could not be opened due to errors found by the **ms3** library. The resulting corpus consists of 502 music sheets that can be opened and analyzed. After the initial analysis, it was found that not all music sheets have one or two staves, but some have three or four. The third and fourth staves are related to the song's theme played using the piano, but they are not the main piano playing staves. Therefore, they were excluded from the model training.

The first hypothesis is whether music sheets describe more than five keys played per staff, as we defined that a piano player would be able to play that number of piano keys at the same time slice, and if those cases are uncommon. Figures 3.10, 3.11 confirm that it is not usual to play more than five keys per staff in the same time slice. As each staff is associated with a single hand for the piano player, excluding any played keys above the fifth will not have a significant impact on summarizing the tune's execution.

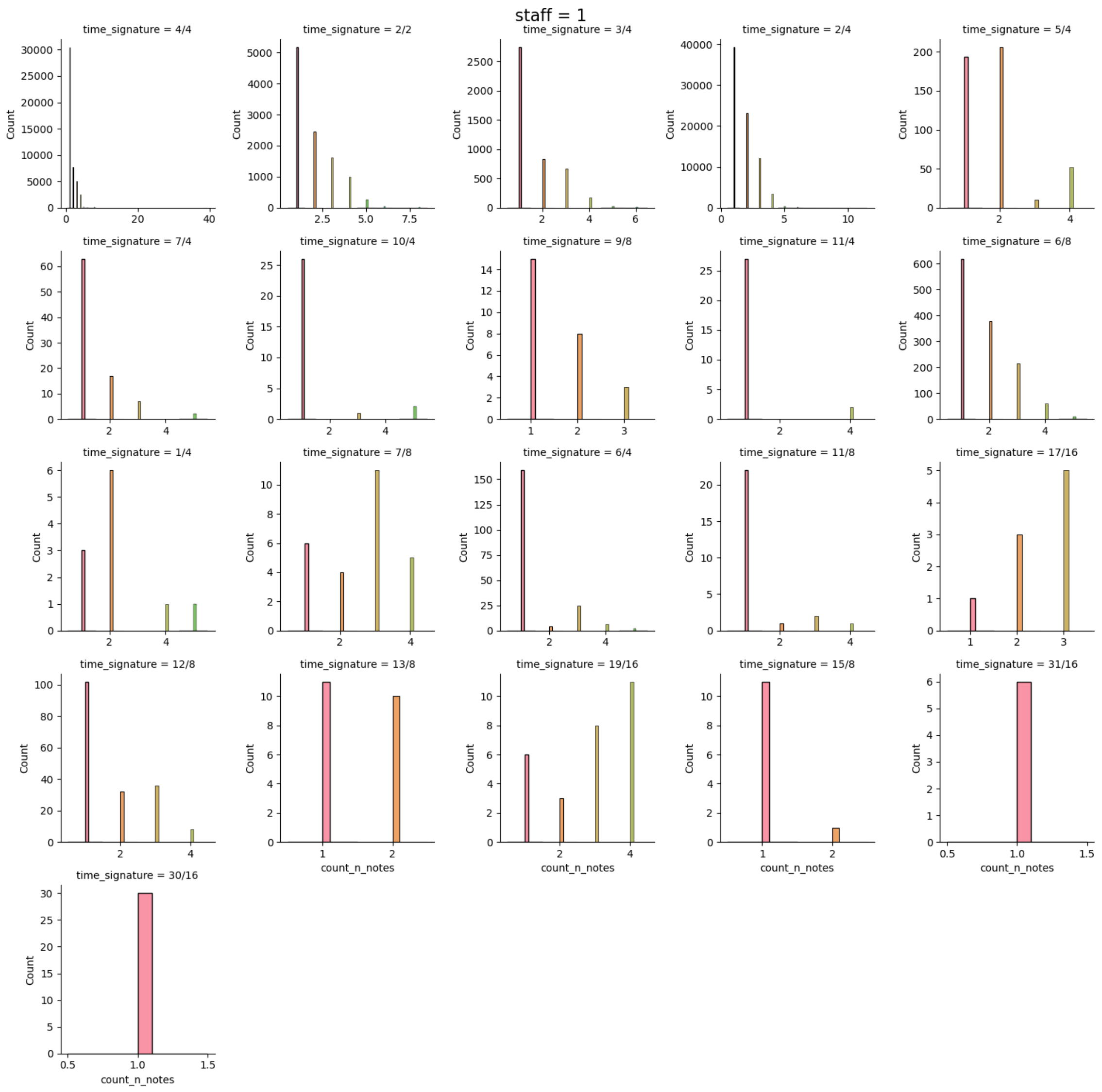


Figure 3.10: Amount of n-notes by time slice on staff 1 presented by time signature.

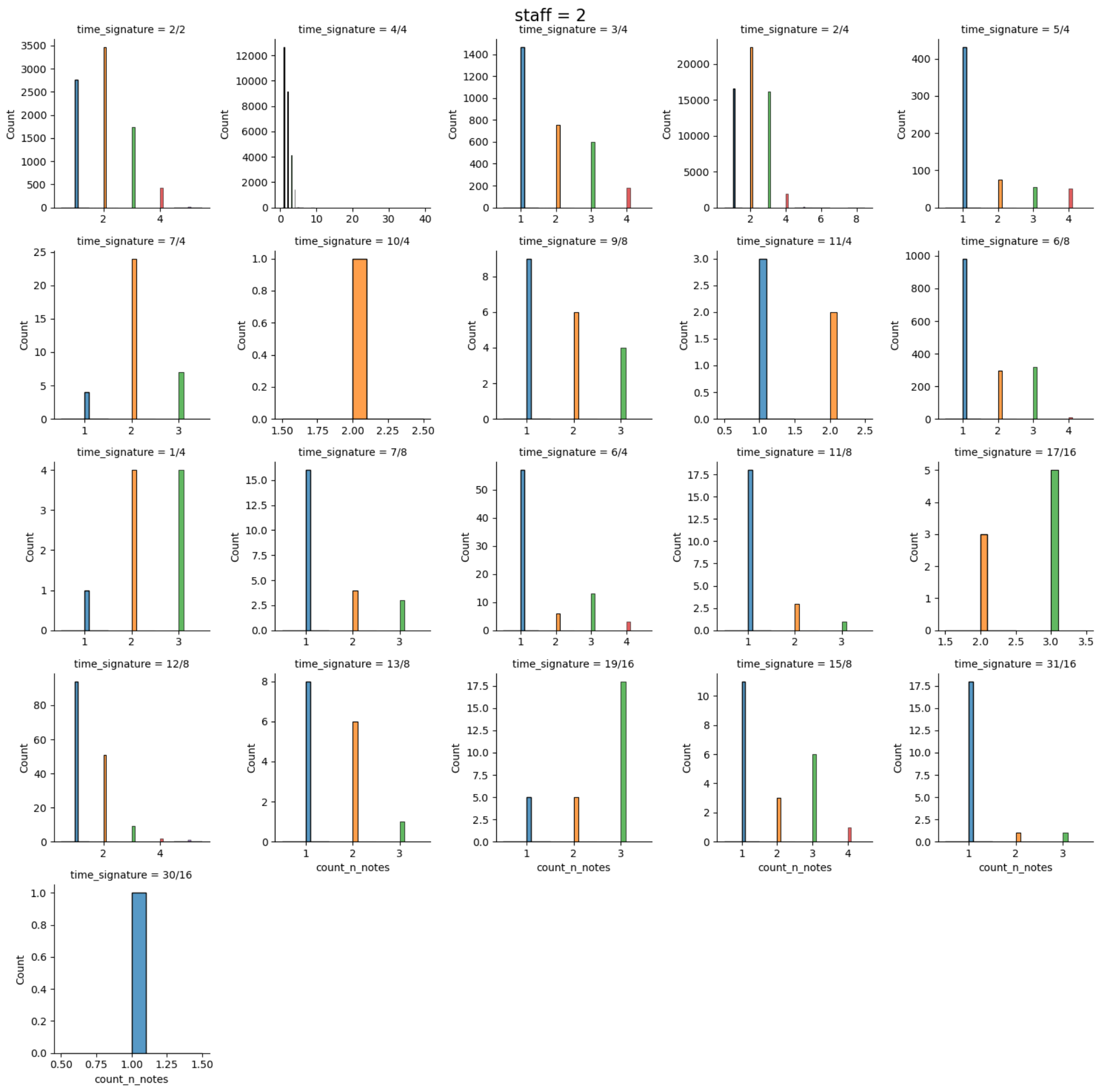


Figure 3.11: Amount of n-notes by time slice on staff 2 presented by time signature.

Notes duration and rests duration were also analyzed for staves one and two. Remember that they are the main staves for model training based on earlier findings during exploratory data analysis.

Having these notes and rests durations is the main input to create probability mass functions to obtain random values from them. This is the selected method to provide the AI with the ability to play the created tune as humans do. The second hypothesis is that each staff would confirm that the left hand, associated with staff 2, would play lower MIDI keys while the right hand, associated with staff 1, would play higher MIDI keys. Figure 3.12 shows that the hypothesis is true. Staves three and four can be seen there, but they don't add much more information to the corpus. The range values for the third and fourth staves are so minimal that a KDE plot in Figure 3.13 was needed to show the existence of a range of values for them.

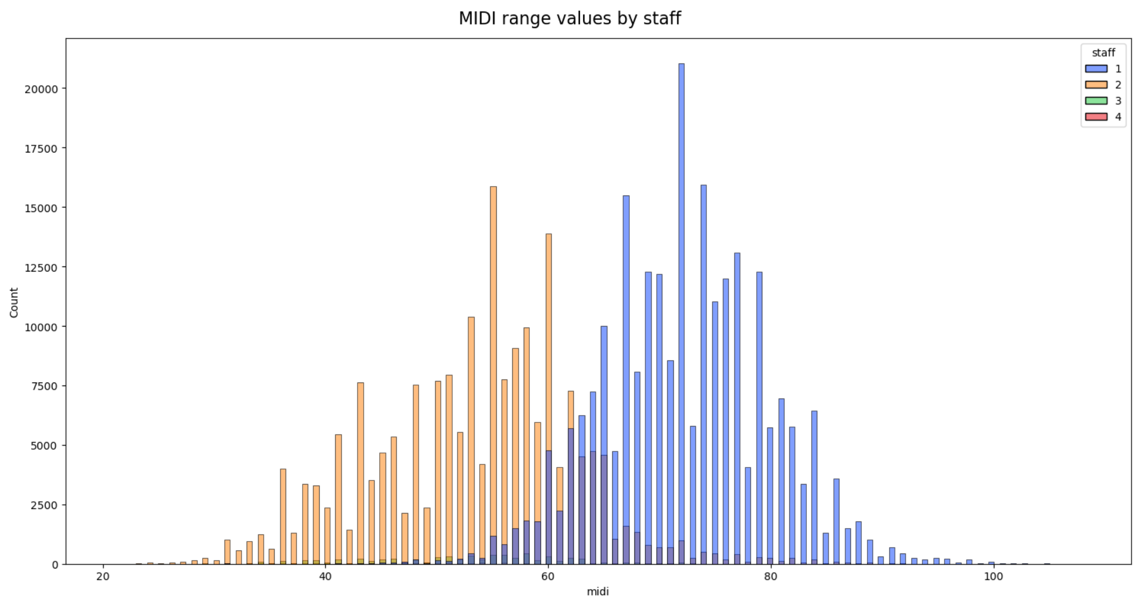


Figure 3.12: MIDI keys range for each staff.

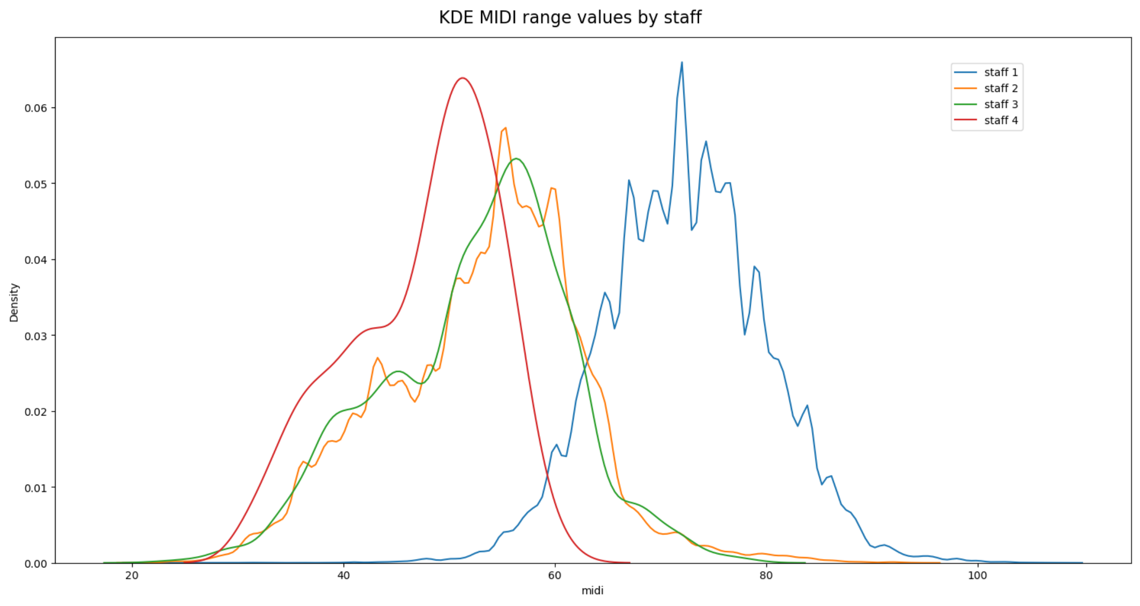


Figure 3.13: KDE MIDI keys range for each staff.

Chapter 4

Feature Engineering, Model Training, Model Consumption, and Results Comparison

4 Feature Engineering, Model Training, Model Consumption, and Results Comparison

The fourth chapter aims to explain to the reader how the feature engineering process was performed and how the neural network model was trained using the vocabulary produced from the cleaned corpus.

4.1 Feature Engineering

Once the corpus was built, creating the features to feed the neural network for model creation is a critical step in producing a solution. The following subsections describe the logic behind feature engineering.

4.1.1 The *n*-notes Slices as Encoded Vocabulary

A piano player is supposed to play the piano staves using both hands. The neural net is expected to learn how to play jazz piano tunes with the same ability. Notes pulsed at the same time, in the same measure of the piano staves are expected to sound harmonious. As described previously, the set of notes (n-notes) pulsed at the same time will be called an n-note slice. It is possible to define the 0-note slice as silence. We will split the n-notes for the left hand (bass clef) and right hand (treble clef). The **Exploratory Data Analysis** showed that *staff 1* is for the right hand and *staff 2* is for the left hand. Each note in the n-note set for staff 1 and staff 2 is described by its MIDI key value. An n-note set implies that there is no chance to have two or more notes with the same MIDI key value in the same slice. For an n-note with n higher than 5, the selected notes to build the set are those 5 with the lowest MIDI key values.

This will be the minimum piece of learning data for the neural network. Figures 4.1 and 4.2 are examples of what was analyzed in a single n-notes slice to identify note duplication as well as n-notes with n larger than 5.

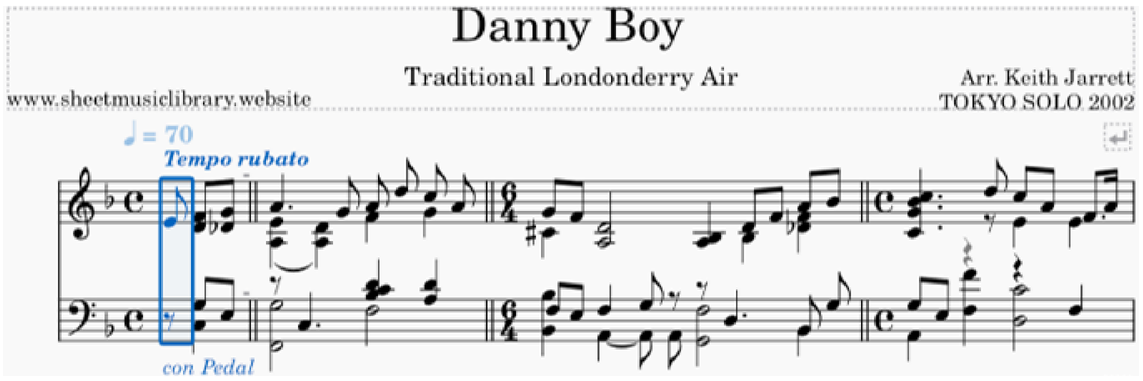


Figure 4.1: 1-note slice for the right hand and 0-note slice for the left hand

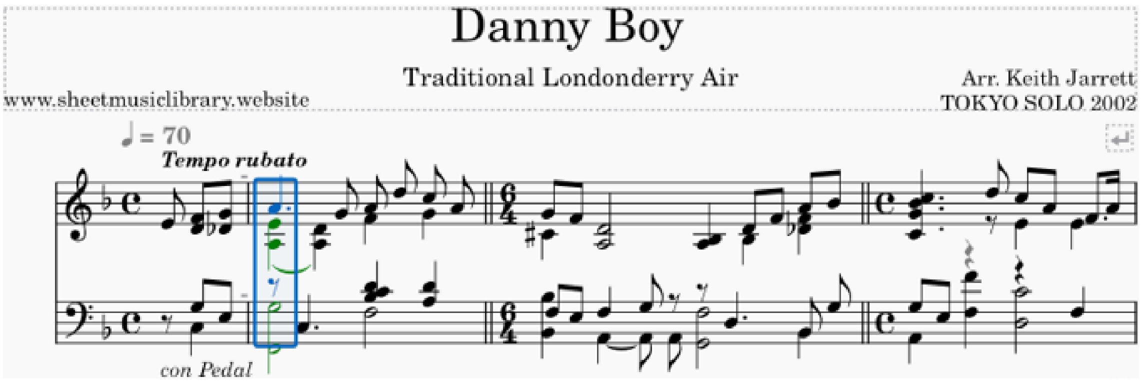
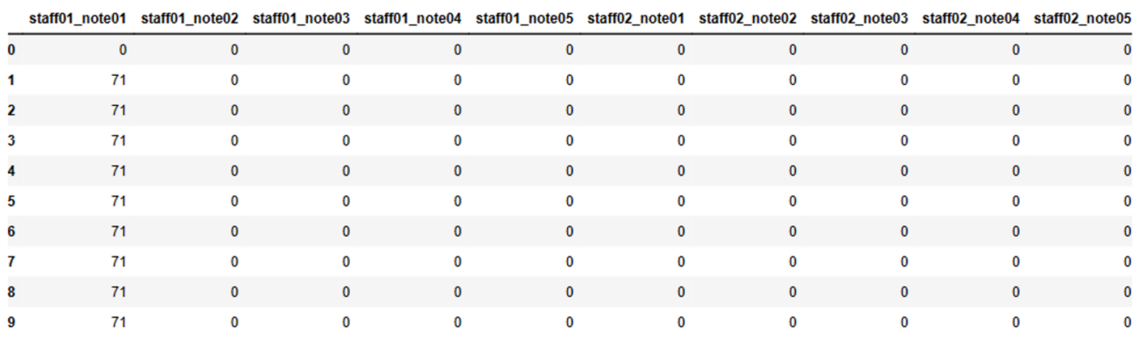


Figure 4.2: 3-note slice for the right hand and 2-note slice for the left hand

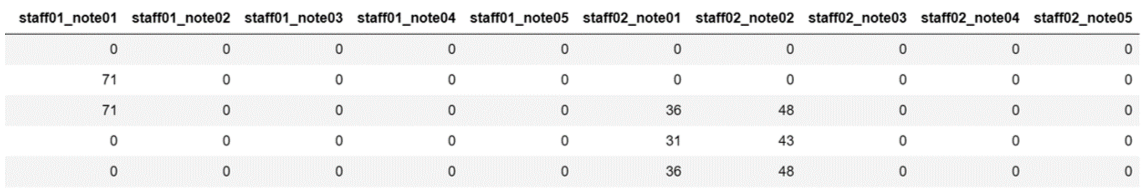
Having the sequence of staff 1 and staff 2 slices would serve as a summarized version of a tune, even if single notes in the staff 1 and staff 2 n-note slice do not preserve their duration. While note duration is not considered a critical part of the neural network training process, it is not irrelevant. Note duration and rests duration are the main inputs to create probability mass functions from which random values are obtained. This is the selected method to provide the AI with the ability to play the created tune as humans do in jazz, improvising and experimenting with note duration to create an unpredictable but pleasant feeling.

Creating a map of staff 1 n-notes and staff 2 n-notes occurring in the same time slice using a single identifier is a necessary step to reduce the space required to describe a tune. Creating a `master_n_notes` matrix by reading all the corpus to collect all the staff 1 and staff 2 n-notes slices, removing duplicates, and mapping all slices using a numerical identifier was the tokenization process implemented to create the vocabulary. The matrix is defined using 10 columns. `staff01_note01` to `staff01_note05` describes the keys pulsed using the five fingers of the right hand while `staff02_note01` to `staff02_note05` describes the keys pulsed using the five fingers of the left hand. It is necessary to clarify that `staff01_note01` doesn't represent that the thumb finger is the one that pulsates the piano key if the value is grather than 0 or that the finger didn't pulsated any key if the value is 0 but simply that **note01**, **note02**, **note03**, **note04** and **note05** for **staff01** or **staff02** where collected as pulsated keys at the same time regardless the finger pressing that key. The slices represented in the `master_n_notes` matrix contain duplicated slices, specifically 183,766. Figure 4.3 shows the first elements in the matrix with duplicates. Notice that it describes the fifth n-notes for staff 1 and the fifth n-notes for staff 2. The first row describes a slice without pulsated keys. The next nine rows describe the same slice with a single pulsated key, the key 71 as described in the `staff01_note01` column. The vocabulary must contain non-repeated elements. Figure 4.4 shows the vocabulary for model training of `master_n_notes`. The vocabulary size is 18,229.



	staff01_note01	staff01_note02	staff01_note03	staff01_note04	staff01_note05	staff02_note01	staff02_note02	staff02_note03	staff02_note04	staff02_note05
0	0	0	0	0	0	0	0	0	0	0
1	71	0	0	0	0	0	0	0	0	0
2	71	0	0	0	0	0	0	0	0	0
3	71	0	0	0	0	0	0	0	0	0
4	71	0	0	0	0	0	0	0	0	0
5	71	0	0	0	0	0	0	0	0	0
6	71	0	0	0	0	0	0	0	0	0
7	71	0	0	0	0	0	0	0	0	0
8	71	0	0	0	0	0	0	0	0	0
9	71	0	0	0	0	0	0	0	0	0

Figure 4.3: `master_n_notes` matrix without duplicates removal.



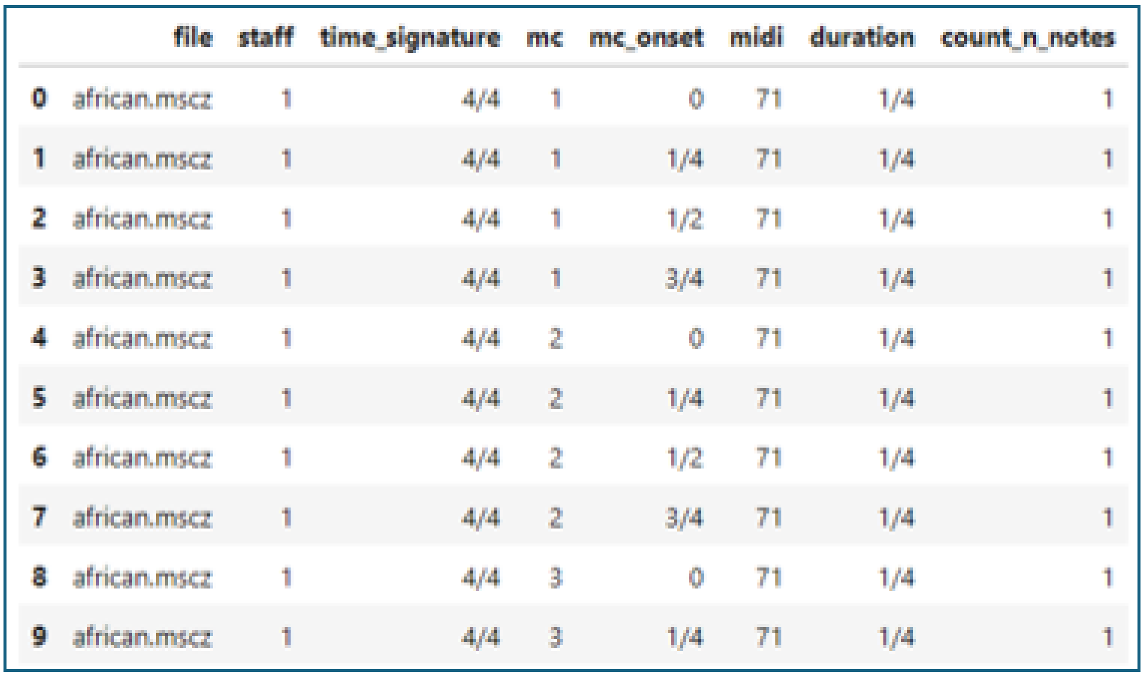
staff01_note01	staff01_note02	staff01_note03	staff01_note04	staff01_note05	staff02_note01	staff02_note02	staff02_note03	staff02_note04	staff02_note05
0	0	0	0	0	0	0	0	0	0
71	0	0	0	0	0	0	0	0	0
71	0	0	0	0	36	48	0	0	0
0	0	0	0	0	31	43	0	0	0
0	0	0	0	0	36	48	0	0	0

Figure 4.4: `master_n_notes` matrix without duplicates.

Each element in the vocabulary matrix `master_n_notes` is identified by the numerical index of the data frame. This encodes all the different set of piano keys pulsated at the same time.

4.1.2 Probability Mass Functions for Dynamic Tune Playing

During the process of building the `master_n_notes` matrix, the duration of notes found on staff 1 and staff 2 was also collected using a counting approach. The matrix produced is called `durations_distributions` and it contains a column for the score name that was analyzed (`file`), the staff analyzed (`staff`), the time signature of the measure (`time_signature`), the measure (`mc`), the position of the note inside the measure (`mc_onset`), the note midi value (`midi`), the note duration (`duration`) and the count of notes (`count_n_notes`). This last column allows us to make sure that the each element occurs once in the same positions in the score. Figure 4.5 shows the matrix content.



	file	staff	time_signature	mc	mc_onset	midi	duration	count_n_notes
0	african.mscz	1	4/4	1	0	71	1/4	1
1	african.mscz	1	4/4	1	1/4	71	1/4	1
2	african.mscz	1	4/4	1	1/2	71	1/4	1
3	african.mscz	1	4/4	1	3/4	71	1/4	1
4	african.mscz	1	4/4	2	0	71	1/4	1
5	african.mscz	1	4/4	2	1/4	71	1/4	1
6	african.mscz	1	4/4	2	1/2	71	1/4	1
7	african.mscz	1	4/4	2	3/4	71	1/4	1
8	african.mscz	1	4/4	3	0	71	1/4	1
9	african.mscz	1	4/4	3	1/4	71	1/4	1

Figure 4.5: The `durations_distributions` matrix.

This data is the foundation for creating probability mass functions that will be used to randomly select "duration" values to provide this characteristic to the notes produced from the to be trained model based on the desired time signature, the staff and the amount of pulsated notes produced by the model.

```
def durations_pfm(data, time_signature = '4/4', count_n_notes = 1,
    ↪ staff = 1):
```

```
    staff_notes_durations = data[(data.time_signature ==
        ↪ time_signature) &
                          (data.count_n_notes == count_n_notes) &
                          (data.staff == staff) &
                          (data.duration.apply(lambda x: frac(x).
                             ↪ denominator).isin([1, 2, 3, 4, 8, 16, 32,
                             ↪  3, 6, 12])) &
                          (data.duration.apply(lambda x: frac(x).
                             ↪ numerator).isin([0, 1, 2, 3]))].groupby(
                             ↪ by='duration', as_index=False)['
                             ↪ count_n_notes']
    staff_notes_durations = staff_notes_durations.count()
    staff_notes_durations['distribution'] = staff_notes_durations.
        ↪ count_n_notes / staff_notes_durations.count_n_notes.sum()

    return staff_notes_durations
```

The construction of the pmf has this definition because a piano player will use the left hand (staff 2) to play notes in different key range and duration compared with the right hand (staff 1). Notes duration will also be different for tunes in $\frac{4}{4}$ time signature compared with tunes in $\frac{5}{4}$ time signature as demonstrated in the exploratory data analysis. Lastly, the ability to play short duration notes depends on the amount of fingers used to press keys in the same time. This appear to happen as it is uncommon for piano players to play n-notes sequences with a high n value using short duration as this would require them high hand velocity and precision and the produced sounds can be confusing for people listen to them. Notice the function filters out uncommon duration values. Figure 4.6 is an example of this behavior in a $\frac{4}{4}$ time signature tune. Notice that left hand (staff 2) tents to play more $\frac{1}{4}$ notes compared with the right hand (staff 1), right hand tents to play short duration notes, this is noticeable in the proportion of notes played using $\frac{1}{16}$ duration.

The distribution is categorized as discrete since duration is not a continuous value but

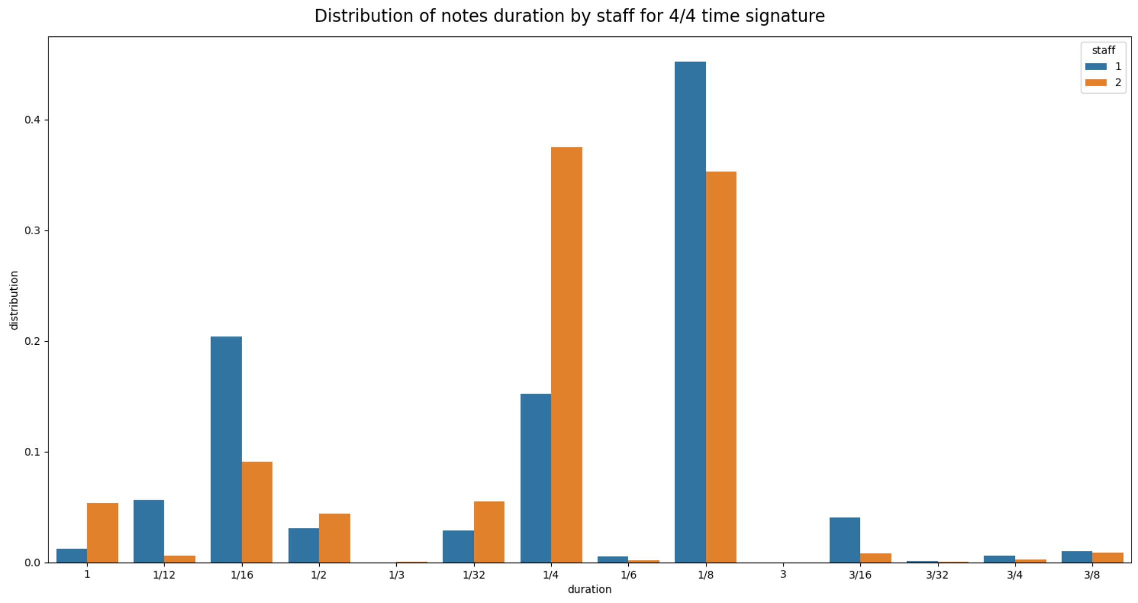


Figure 4.6: Comparison of notes duration for left hand (staff 2) and right hand (staff 1).

a set of discrete elements. Table 4.1 describes the **pmf**s created for dynamic tune playing. The application of this function will be explained later in this chapter.

PMFs per staff and time signatures					
Staves / Time sig.	$\frac{4}{4}$ time sig.	$\frac{2}{4}$ time sig.	$\frac{2}{2}$ time sig.	$\frac{3}{4}$ time sig.	$\frac{5}{4}$ time sig.
Staff 01	1-notes 2-notes 3-notes 4-notes 5-notes	1-notes 2-notes 3-notes 4-notes 5-notes	1-notes 2-notes 3-notes 4-notes 5-notes	1-notes 2-notes 3-notes 4-notes 5-notes	1-notes 2-notes 3-notes 4-notes 5-notes
Staff 02	1-notes 2-notes 3-notes 4-notes 5-notes	1-notes 2-notes 3-notes 4-notes 5-notes	1-notes 2-notes 3-notes 4-notes 5-notes	1-notes 2-notes 3-notes 4-notes 5-notes	1-notes 2-notes 3-notes 4-notes 5-notes

Table 4.1: Probability Mass Functions for Dynamic Tune Playing.

4.1.3 Encoding *n*-notes sequences

Using the `master_n_notes` matrix as identifiers to encode the n-note sequences, the corpus was read to create sequences of three n-notes as they occurred in the tunes: the PREVIOUS_N_NOTES, the ACTUAL_N_NOTES, and the NEXT_N_NOTES. This structure is useful as it is equivalent to the vector used to define a sequence of words from the vocabulary. The structure doesn't contain the n-notes but their IDs from the `master_n_notes` matrix. The features produced show two main challenges:

- Contiguous sequences with the same n-notes ID.
- Contiguous sequences of 0-notes ID.

An example of why these structures exist in the corpus is the **C-Jam Blues** tune. The music sheet contains the same note played in two consecutive slices, and the same happens for rests. Figure 4.7 shows this behavior.

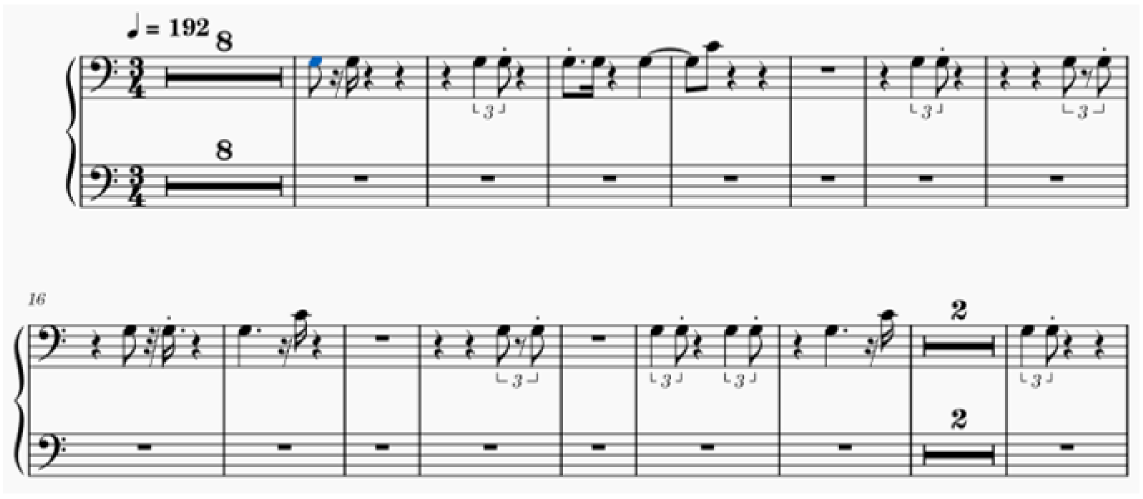
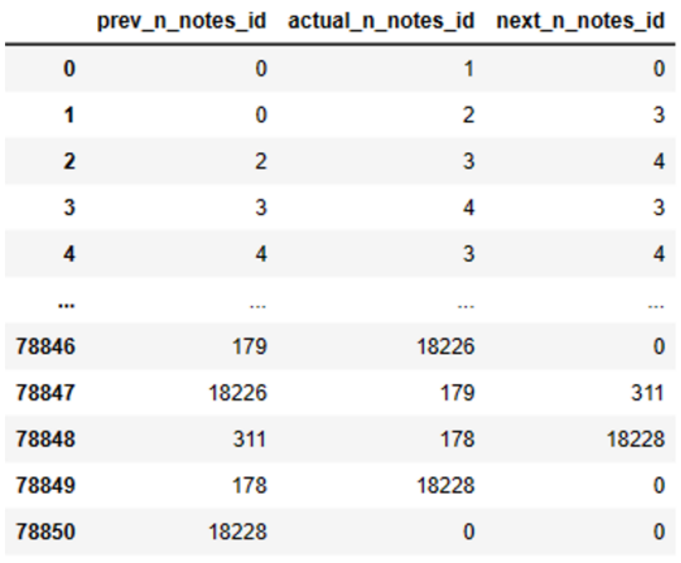


Figure 4.7: C-Jam Blues bars with consecutive slices of the same notes and consecutive rests.

This behavior was identified as the cause for the model produced to learn to go on loops of the same n-notes sequences or to go on rests. To reduce that chance, consecutive rests or consecutive n-notes IDs were identified and removed. Another possible loop is that with sequences of 2 or more n-notes slices that repeat over and over. The removal of this kind of loop is done later by evaluating model prediction outputs to avoid this behavior. The matrix containing all this information was called `unique_n_notes_sequences`. It contains unique sequences of n-notes. The matrix with duplicated sequences contained 165,563 sequences. Once duplicates were removed, the remaining sequences were 78,851. We don't want to overtrain our model with repetitive data to avoid loops.

The `unique_n_notes_sequences` matrix is the main input for the model training step.



	prev_n_notes_id	actual_n_notes_id	next_n_notes_id
0	0	1	0
1	0	2	3
2	2	3	4
3	3	4	3
4	4	3	4
...	...	...	...
78846	179	18226	0
78847	18226	179	311
78848	311	178	18228
78849	178	18228	0
78850	18228	0	0

Figure 4.8: The `unique_n_notes_sequences` matrix.

The model training process is described in the following section.

4.2 Model Training

The corpus is adequately encoded so far. The process to train a model requires a final transformation for model training. Both processes are described below.

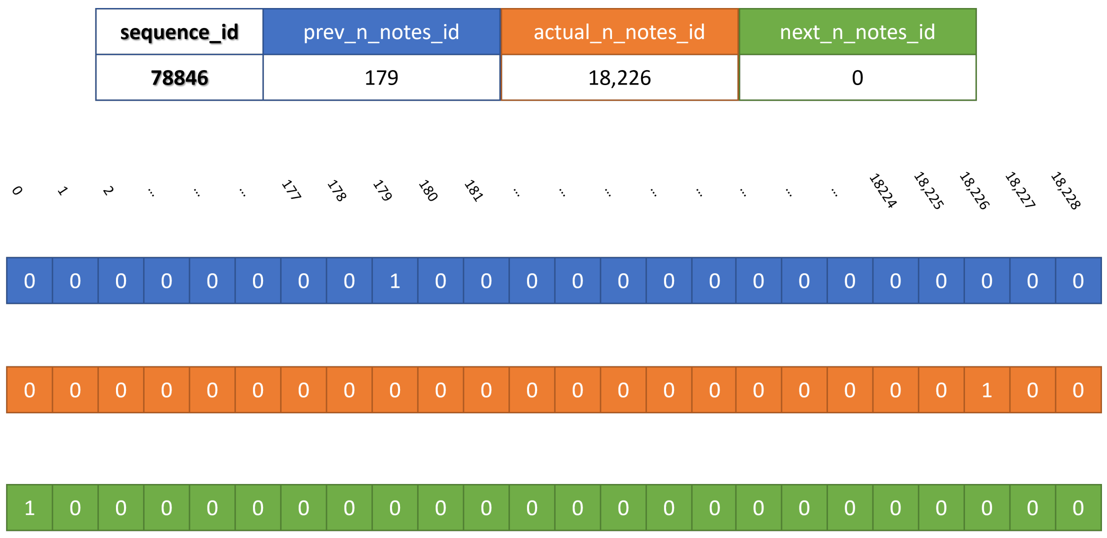
4.2.1 Input Tensors

The `unique_n_notes_sequences` matrix is the main input for model training. It describes a sequence of 3 n-note slices (3 elements from the vocabulary in a sequence) by their IDs, as shown in figure 4.8. The `unique_n_notes_sequences` matrix storing those sequences are to be translated into tensor sequences.

The `unique_n_notes_sequences` matrix describes sequences of 3 n-notes represented by their IDs. This technique is similar to the one to represent sequences of words from a vocabulary like this: "i", "trust", "you", or this one: "go", "for", "it". The representation stored in the matrix is not for plain text values but their IDs from the `master_n_notes` matrix.

Figure 4.9 describes the mapping of a single sequence with ID $78,846$ as stored in the `unique_n_notes_sequences` matrix (upper table) translated to its tensor representation (the 3 colored vectors below). This tensor consists of 3 vectors to store the one-hot encoded pulsated n-notes in a sequence of 3 elements. So, each vector is of size 18,229 as this is the length of the vocabulary. Notice that a single element of the vector will show 1 while 0 will be stored for any other element. All values in the tensor are set to 0, except for the i-th element given by the n-notes ID. The i-th element is set to 1. The blue vector represent the encoded first n-notes in the sequence, the 179-th element is set to 1 while any other element is 0. The orange vector represent the encoded following n-notes in the sequence, the $18,226$-th element is set to 1 while any other element is 0. Lastly, the green vector represent the encoded following n-notes in the sequence, the 0-th element is set to 1 while any other element is 0. This 0-th element represents the rest (no keys pressed), as mentioned previously.

This tensor structure is similar to that used for Natural Language Processing (NLP)



sequence_id	prev_n_notes_id	actual_n_notes_id	next_n_notes_id
78846	179	18,226	0

Figure 4.9: Mapping of the sequence ID 78846 in the `unique_n_notes_sequences` matrix translated to tensor representation.

problems. This sequence was used to feed recurrent neural networks as the approach to create a reliable model for automated music creation based on a single initial n-notes element. Music n-notes sequences don't need to be too long, as the previous n-notes in the sequence maintain a harmonic relationship with the actual n-notes. That is why using `prev_n_notes_id` to predict `actual_n_notes_id` is enough for model training. This is the approach used during LSTM model training and GRU model training.

4.2.2 Training a model

For model training, two algorithms were used: LSTM and GRU. The architecture of the neural network for model training is similar for both. The input layer consumes an 18,229-element input tensor, and this input is the output for the next layer. An LSTM or GRU layer gets its input from the previous layer. The LSTM or GRU layer gets a tensor of 18,229 elements, and this layer produces a 1,024-element long output. A dense layer gets input from the LSTM or GRU layer. This layer gets a 1,024-element long input and produces an 18,299 output. The last layer produces normalized probability values using the softmax function. Both architectures are described in Figure 4.10. This is a simple but good enough architecture to create the artificial intelligence for music composing assistance.

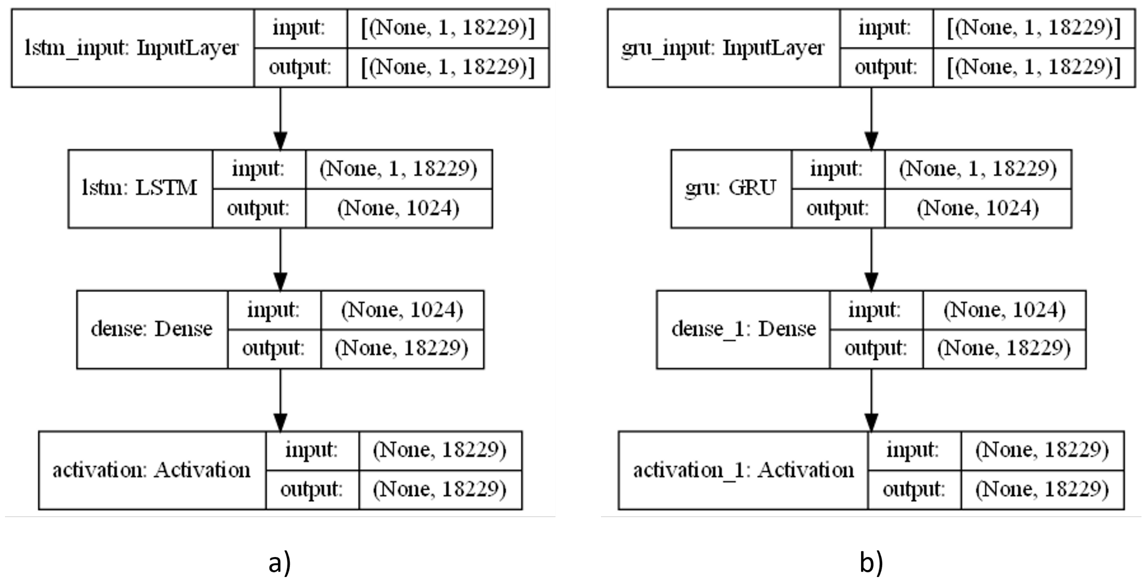


Figure 4.10: a) Architecture of a LSTM RNN. b) Architecture of a GRU RNN.

Both models had similar `accuracy` for each training round. As there is not a clear winning model, the model to be chosen is the one with the best `accuracy` for that specific experimentation round. Notice that produced music is hard to categorize as **right** or **wrong**. As long as a two n-note sequence sounds harmonic, many of them having the same `prev_n_notes_id` will be just right. That is why there is no final decision to select a model other than the criteria given by the `accuracy` of both. This approach to model selection is the same that enterprises use for model retraining. The model to be published for application consumption is the one with the best performance.

This training approach was used for remote training, training, and publishing of the model using Azure Machine Learning.

4.3 Model Consumption

Consuming the model created and deployed as an endpoint using Azure Machine Learning requires the definition of a scoring script. The script consumes a vector that describes an initial set of n-notes. It will also take care of two problems found during model testing. The first problem is having outputs of n-notes going into repeated sequences. The second problem was the need to create a sequence of n-notes, setting duration for them, but in a dynamic execution as jazz piano players would do during the tune creation phase. The proposed solution to the first problem is described next.

The endpoint receives an initial vector of n-notes. This vector goes into a transformation to get its ID based on the `master_n_notes` set. This ID is then used for one-hot encoding to create a tensor representation of this input. This transformed input is used as the first call for predicting a tensor, from which the most probable element is then mapped back to an n-notes ID. This is then translated back to an n-notes set. Each n-notes set has ten values, the first five are for the n-notes for the treble clef (C clef), and the latest five are for the bass clef (F clef). The produced n-notes set is then appended to a Python DataFrame. This new output is used in a 64-bars loop as input to produce the next n-notes set in the sequence.

4.3.1 Avoiding To Theme Loops

As the sequence is being built, a test for repetition using the last produced n-note set is performed against groups of previous n-notes in the sequence. If a loop is identified, then getting another value from the top 5 most probable recent n-note predictions, excluding the top 1 using random selection, is the approach to avoid loops. This worked fine during model testing.

4.3.2 Dynamic Execution Using Probability Mass Functions

The second issue to be addressed was providing dynamic duration to the produced sequence of n-notes. The approach to achieve this was to use probability mass functions (pmfs) as described previously. Every single iteration used to produce new n-notes to be appended in the sequence was used as input to call the `random.choices` function. This function accepts the population of notes duration, the weights as the frequency of each possible note duration and the amount of values to be returned randomly. The time signature parameter will be one of the following:

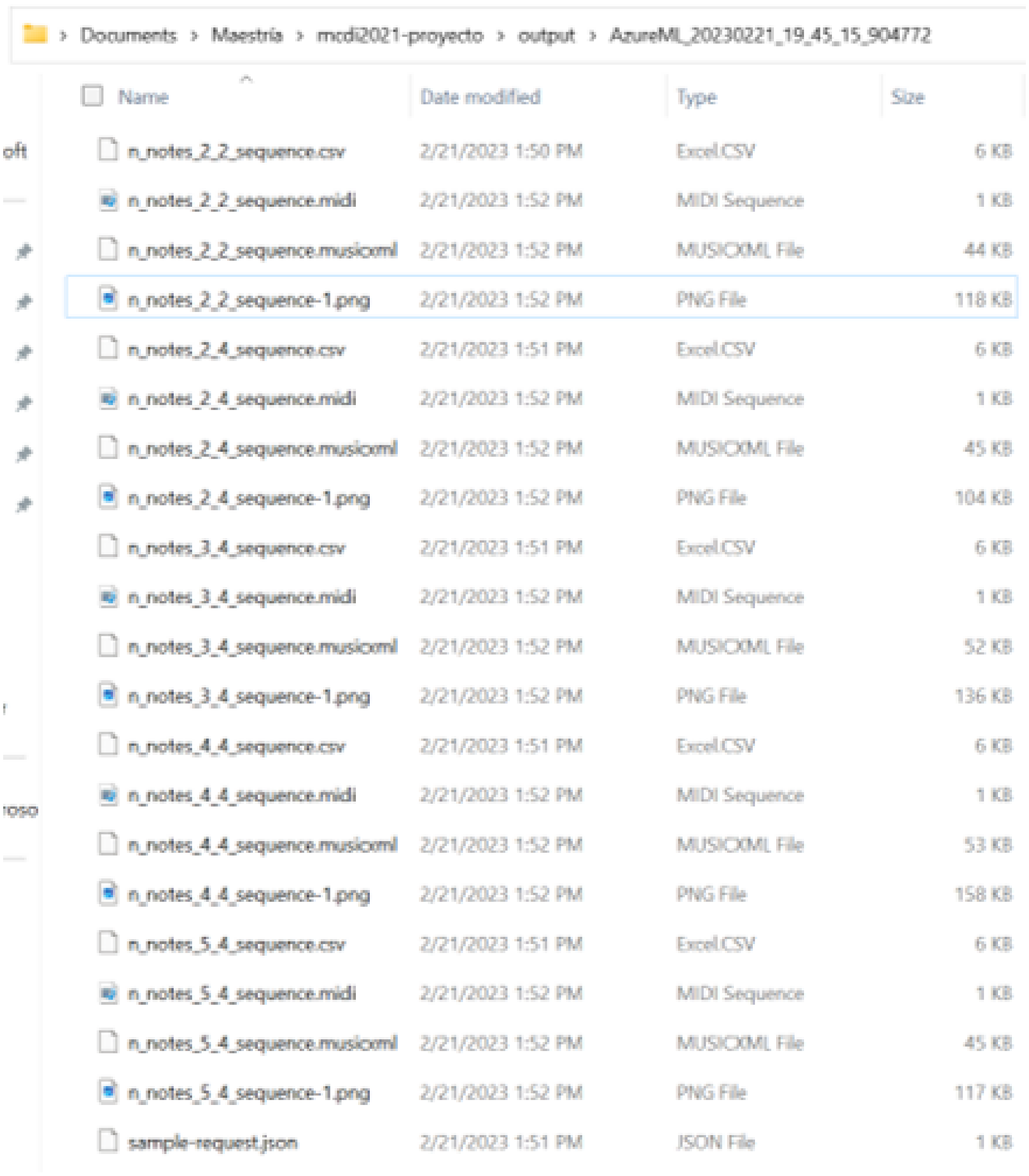
- $\frac{2}{2}$ for staff 1 and staff 2.
- $\frac{2}{4}$ for staff 1 and staff 2.

- $\frac{3}{4}$ for staff 1 and staff 2.
- $\frac{4}{4}$ for staff 1 and staff 2.
- $\frac{5}{4}$ for staff 1 and staff 2.

Each time signature and staff is used to produce an output DataFrame, all them containing the same n-notes sequence. The notes will last based on the random duration from the pmf. The n-notes set for each time signature and the notes' duration is a new tune. This is the approach to produce new music as ideas for music composition using neural nets.

4.3.3 Producing New Music Sequences

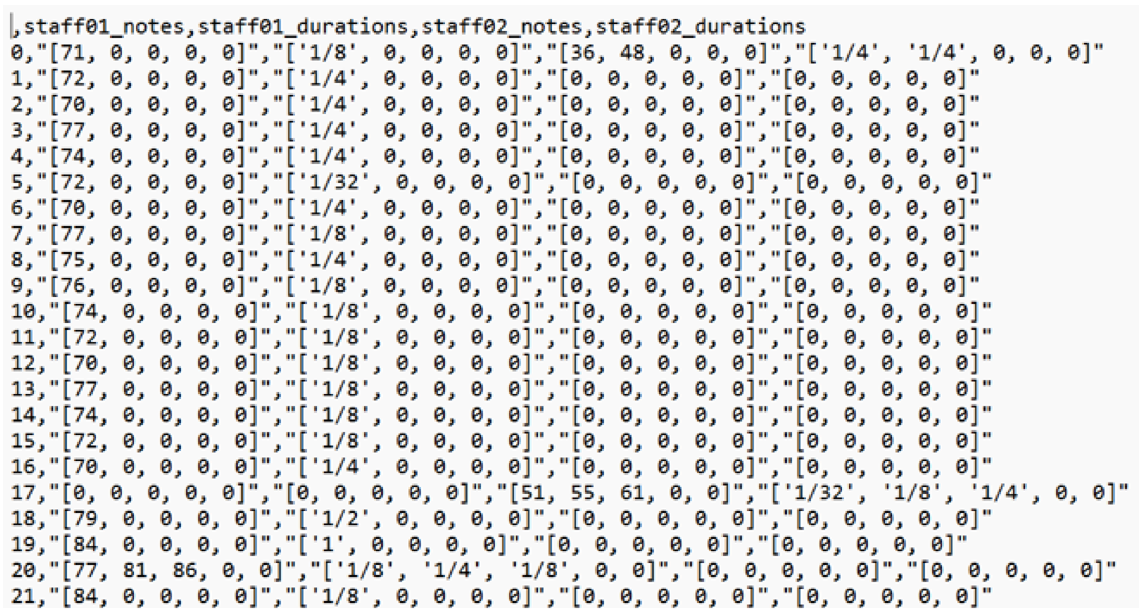
When calling the deployed model using the REST API and executing the code described for scoring, a pandas DataFrame is produced, which is then converted to a JSON document as the last part of the scoring code execution. This JSON document contains the created sequence of n-notes and their duration. The data is then presented locally using the `music21` Python library. This library is used to produce XML scores, MIDI files, PNG files, and the CSV representation for every n-notes sequence for each time signature. An example of the output is shown in figures 4.11, 4.12, 4.13, and 4.14. This same code can be implemented on mobile devices to consume the output from the REST API endpoint.



Documents › Maestría › mcdi2021-proyecto › output › AzureML_20230221_19_45_15_904772

Name	Date modified	Type	Size
n_notes_2_2_sequence.csv	2/21/2023 1:50 PM	Excel.CSV	6 KB
n_notes_2_2_sequence.midi	2/21/2023 1:52 PM	MIDI Sequence	1 KB
n_notes_2_2_sequence.musicxml	2/21/2023 1:52 PM	MUSICXML File	44 KB
n_notes_2_2_sequence-1.png	2/21/2023 1:52 PM	PNG File	118 KB
n_notes_2_4_sequence.csv	2/21/2023 1:51 PM	Excel.CSV	6 KB
n_notes_2_4_sequence.midi	2/21/2023 1:52 PM	MIDI Sequence	1 KB
n_notes_2_4_sequence.musicxml	2/21/2023 1:52 PM	MUSICXML File	45 KB
n_notes_2_4_sequence-1.png	2/21/2023 1:52 PM	PNG File	104 KB
n_notes_3_4_sequence.csv	2/21/2023 1:51 PM	Excel.CSV	6 KB
n_notes_3_4_sequence.midi	2/21/2023 1:52 PM	MIDI Sequence	1 KB
n_notes_3_4_sequence.musicxml	2/21/2023 1:52 PM	MUSICXML File	52 KB
n_notes_3_4_sequence-1.png	2/21/2023 1:52 PM	PNG File	136 KB
n_notes_4_4_sequence.csv	2/21/2023 1:51 PM	Excel.CSV	6 KB
n_notes_4_4_sequence.midi	2/21/2023 1:52 PM	MIDI Sequence	1 KB
n_notes_4_4_sequence.musicxml	2/21/2023 1:52 PM	MUSICXML File	53 KB
n_notes_4_4_sequence-1.png	2/21/2023 1:52 PM	PNG File	158 KB
n_notes_5_4_sequence.csv	2/21/2023 1:51 PM	Excel.CSV	6 KB
n_notes_5_4_sequence.midi	2/21/2023 1:52 PM	MIDI Sequence	1 KB
n_notes_5_4_sequence.musicxml	2/21/2023 1:52 PM	MUSICXML File	45 KB
n_notes_5_4_sequence-1.png	2/21/2023 1:52 PM	PNG File	117 KB
sample-request.json	2/21/2023 1:51 PM	JSON File	1 KB

Figure 4.11: Output files created using the JSON response from the Azure ML REST API.



```
,staff01_notes,staff01_durations,staff02_notes,staff02_durations
0,"[71, 0, 0, 0, 0]","['1/8', 0, 0, 0, 0]","[36, 48, 0, 0, 0]","['1/4', '1/4', 0, 0, 0]"
1,"[72, 0, 0, 0, 0]","['1/4', 0, 0, 0, 0]","[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]"
2,"[70, 0, 0, 0, 0]","['1/4', 0, 0, 0, 0]","[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]"
3,"[77, 0, 0, 0, 0]","['1/4', 0, 0, 0, 0]","[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]"
4,"[74, 0, 0, 0, 0]","['1/4', 0, 0, 0, 0]","[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]"
5,"[72, 0, 0, 0, 0]","['1/32', 0, 0, 0, 0]","[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]"
6,"[70, 0, 0, 0, 0]","['1/4', 0, 0, 0, 0]","[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]"
7,"[77, 0, 0, 0, 0]","['1/8', 0, 0, 0, 0]","[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]"
8,"[75, 0, 0, 0, 0]","['1/4', 0, 0, 0, 0]","[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]"
9,"[76, 0, 0, 0, 0]","['1/8', 0, 0, 0, 0]","[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]"
10,"[74, 0, 0, 0, 0]","['1/8', 0, 0, 0, 0]","[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]"
11,"[72, 0, 0, 0, 0]","['1/8', 0, 0, 0, 0]","[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]"
12,"[70, 0, 0, 0, 0]","['1/8', 0, 0, 0, 0]","[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]"
13,"[77, 0, 0, 0, 0]","['1/8', 0, 0, 0, 0]","[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]"
14,"[74, 0, 0, 0, 0]","['1/8', 0, 0, 0, 0]","[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]"
15,"[72, 0, 0, 0, 0]","['1/8', 0, 0, 0, 0]","[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]"
16,"[70, 0, 0, 0, 0]","['1/4', 0, 0, 0, 0]","[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]"
17,"[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]","[51, 55, 61, 0, 0]","['1/32', '1/8', '1/4', 0, 0]"
18,"[79, 0, 0, 0, 0]","['1/2', 0, 0, 0, 0]","[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]"
19,"[84, 0, 0, 0, 0]","['1', 0, 0, 0, 0]","[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]"
20,"[77, 81, 86, 0, 0]","['1/8', '1/4', '1/8', 0, 0]","[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]"
21,"[84, 0, 0, 0, 0]","['1/8', 0, 0, 0, 0]","[0, 0, 0, 0, 0]","[0, 0, 0, 0, 0]"
```

Figure 4.12: CSV file created using the JSON response from the Azure ML REST API.

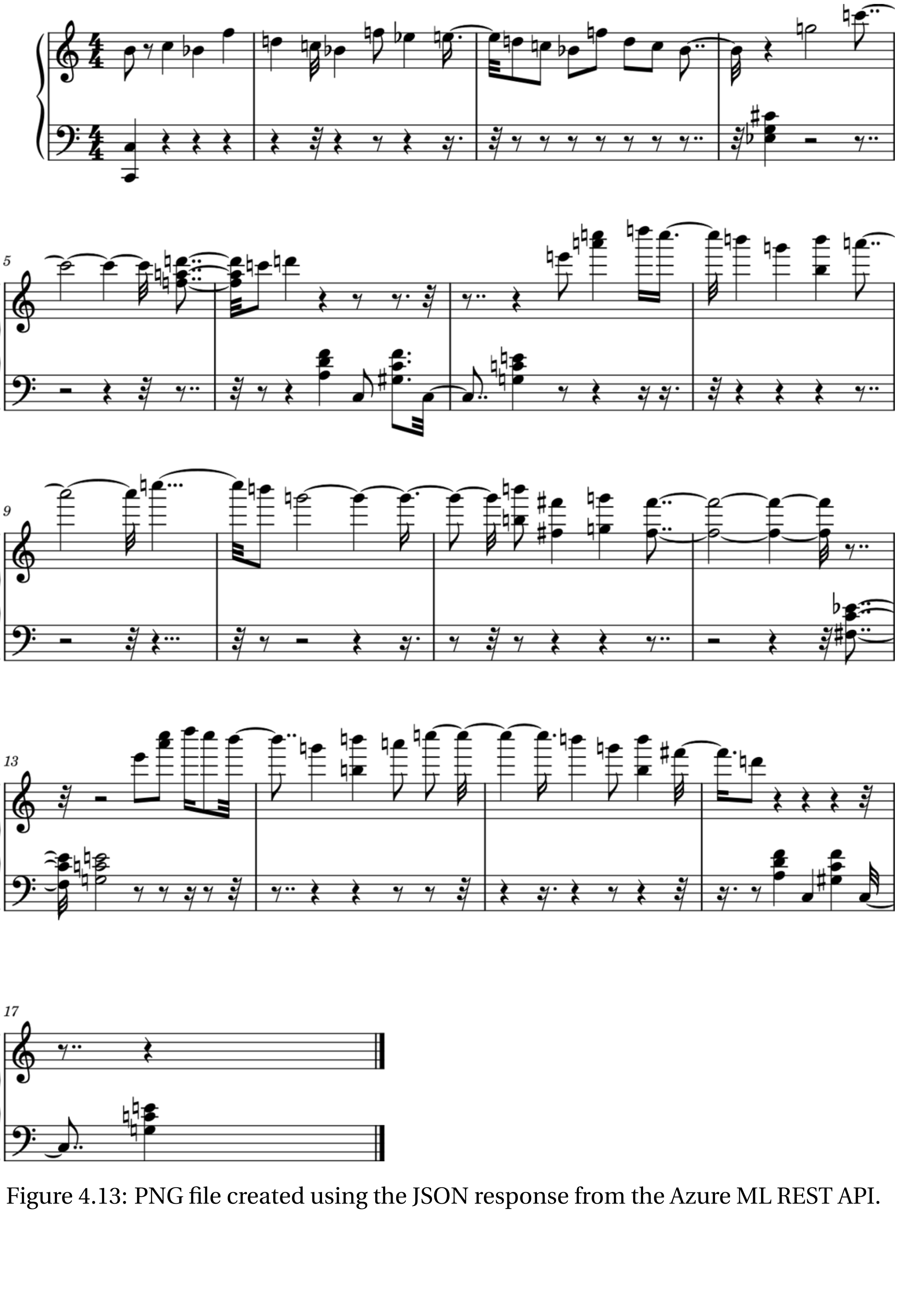


Figure 4.13: PNG file created using the JSON response from the Azure ML REST API.

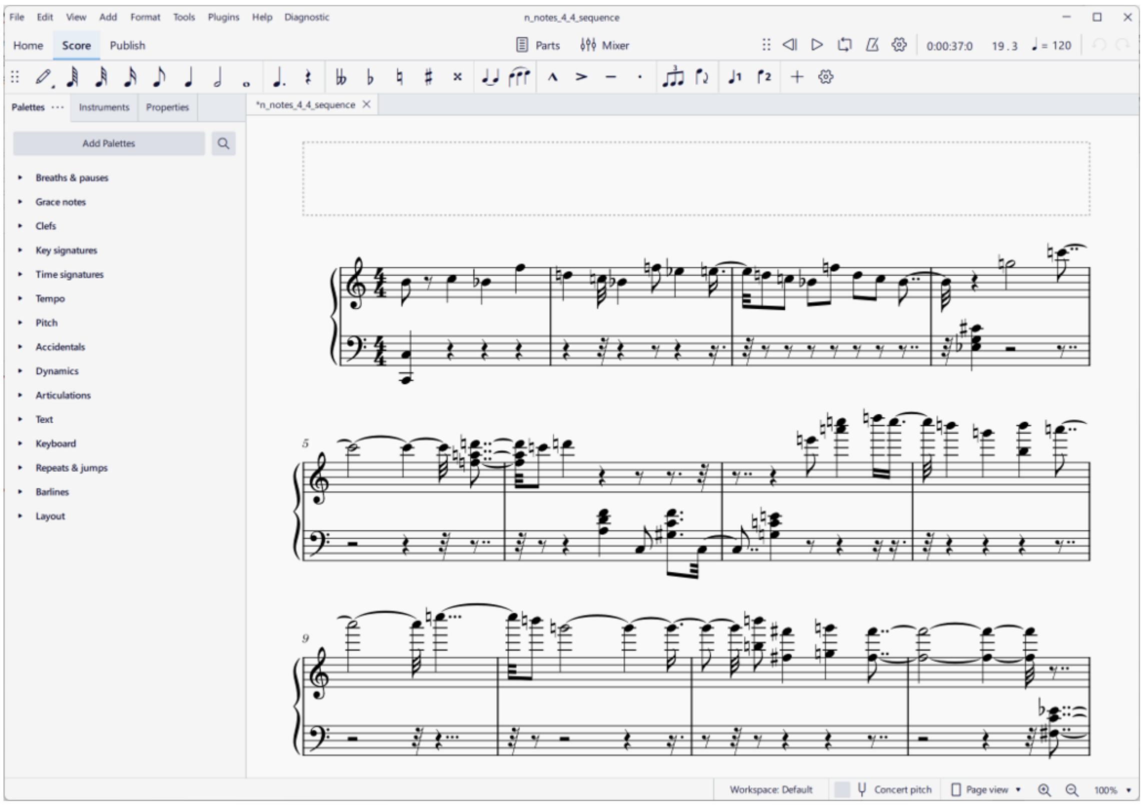


Figure 4.14: MusicXML file created using the JSON response from the Azure ML REST API.

4.4 Results Comparison

Using the model, a set of 100 new scores was created. These scores were used to build a new dataset to be compared with the original corpus. Those datasets were compared by their statistical distribution of n-notes, note duration, rest duration, and MIDI keys.

The figures 4.15 and 4.16 show the distribution of n-notes per staff from the newly created dataset.

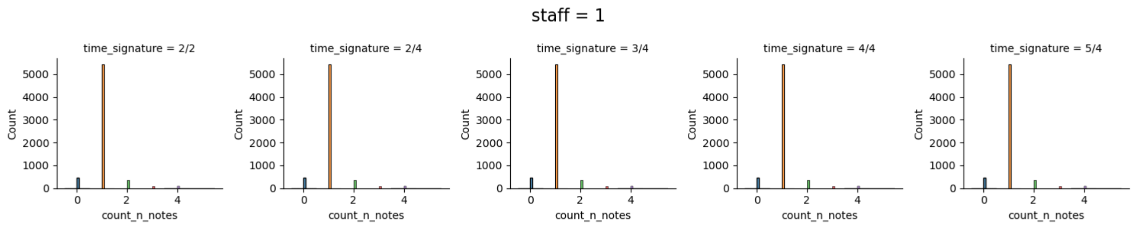


Figure 4.15: Amount of n-notes in the produced dataset by time slice on staff 1 presented by time signature.

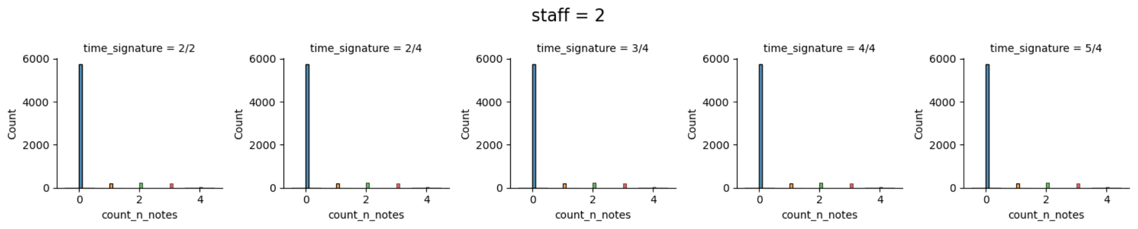


Figure 4.16: Amount of n-notes in the produced dataset by time slice on staff 2 presented by time signature.

Notes duration and rests duration were also analyzed for staves one and two. The figures 4.17, 4.18, 4.19, and 4.20 show the duration of notes and rests from the new dataset.

The figure 4.21 shows the range of MIDI keys on the new dataset. The distribution of those values can be verified in figure 4.22.

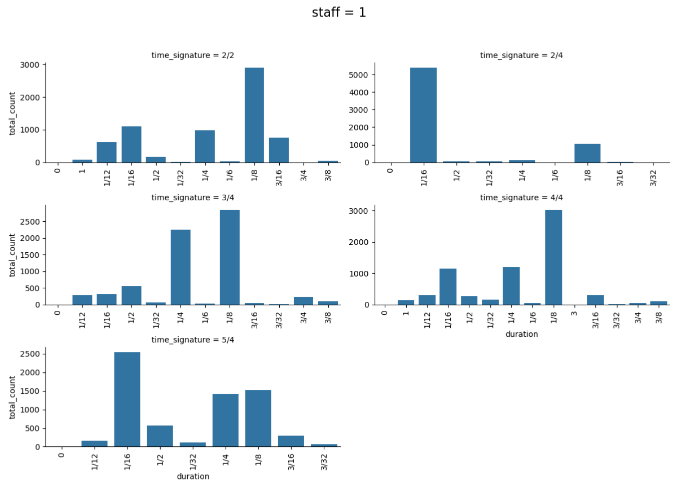


Figure 4.17: Notes duration in staff 1 from the produced dataset presented by time signature.

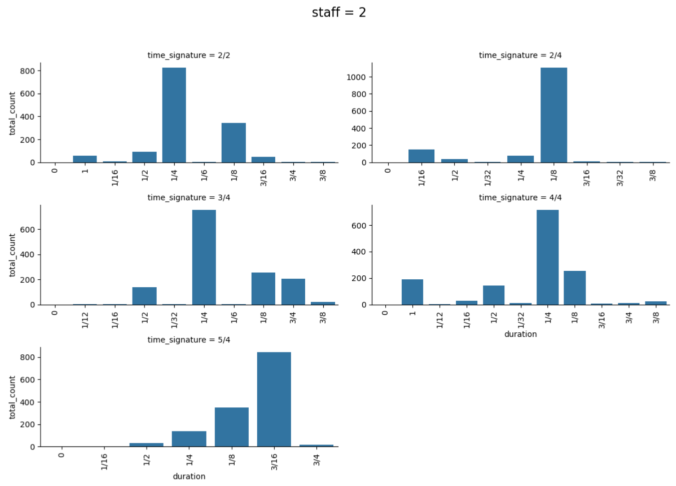


Figure 4.18: Notes duration in staff 2 from the produced dataset presented by time signature.

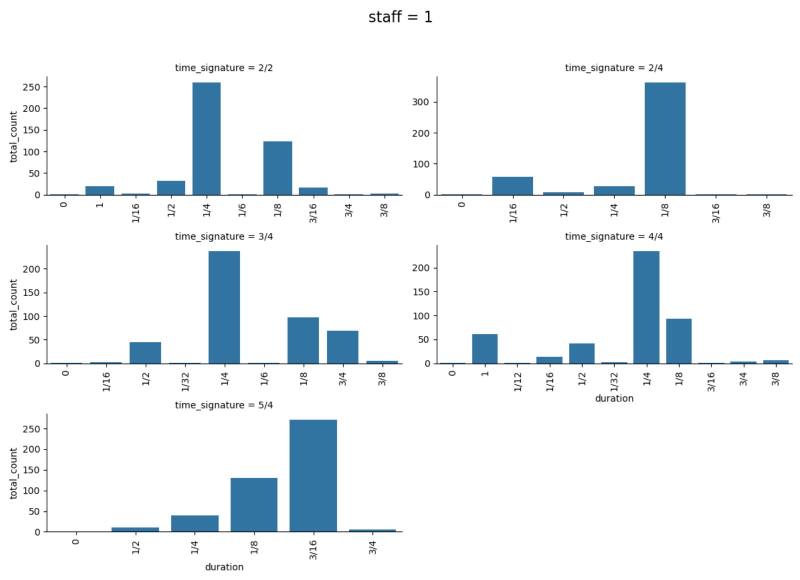


Figure 4.19: Rests duration in staff 1 from the produced dataset presented by time signature.

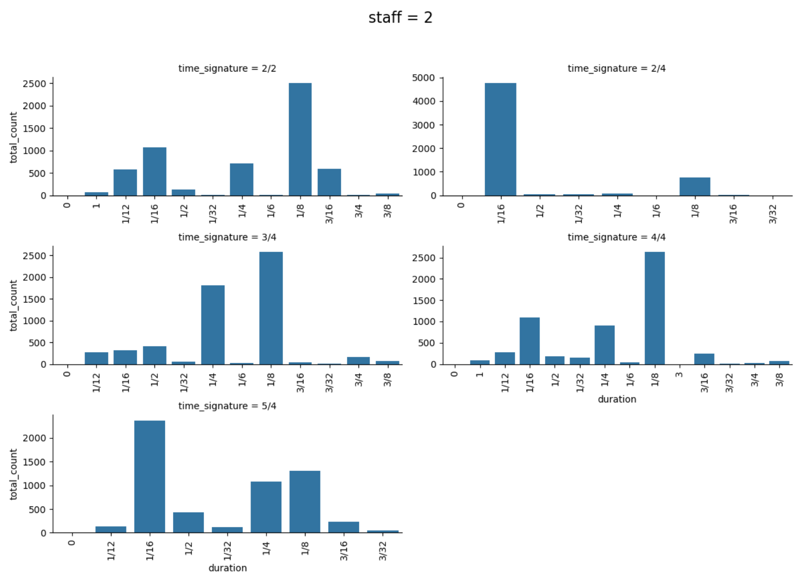


Figure 4.20: Rests duration in staff 2 from the produced dataset presented by time signature.

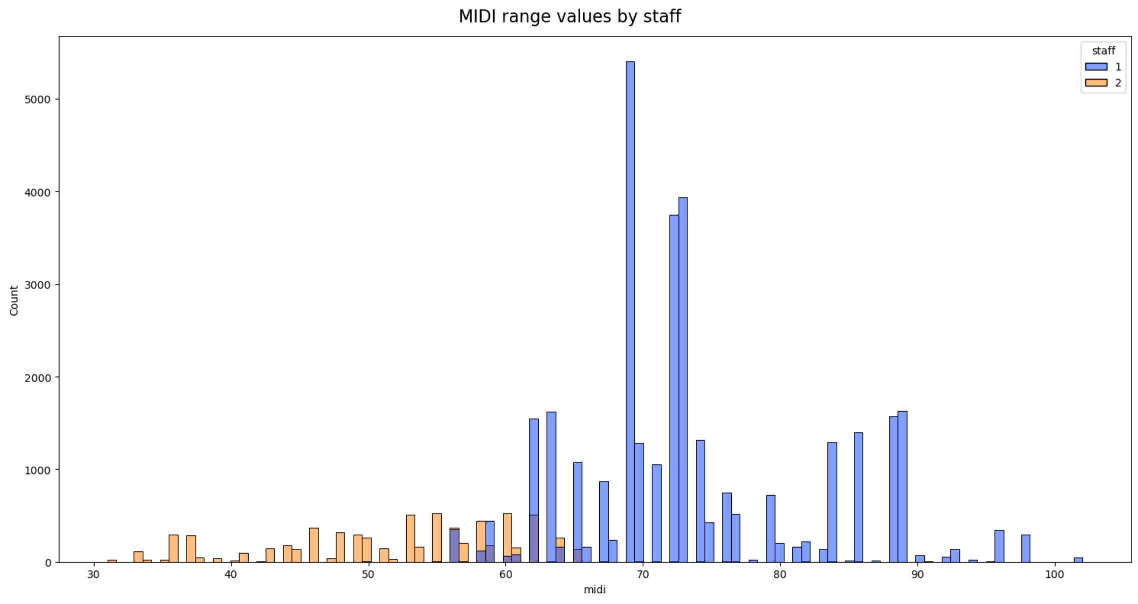


Figure 4.21: MIDI keys range for each staff in the produced dataset.

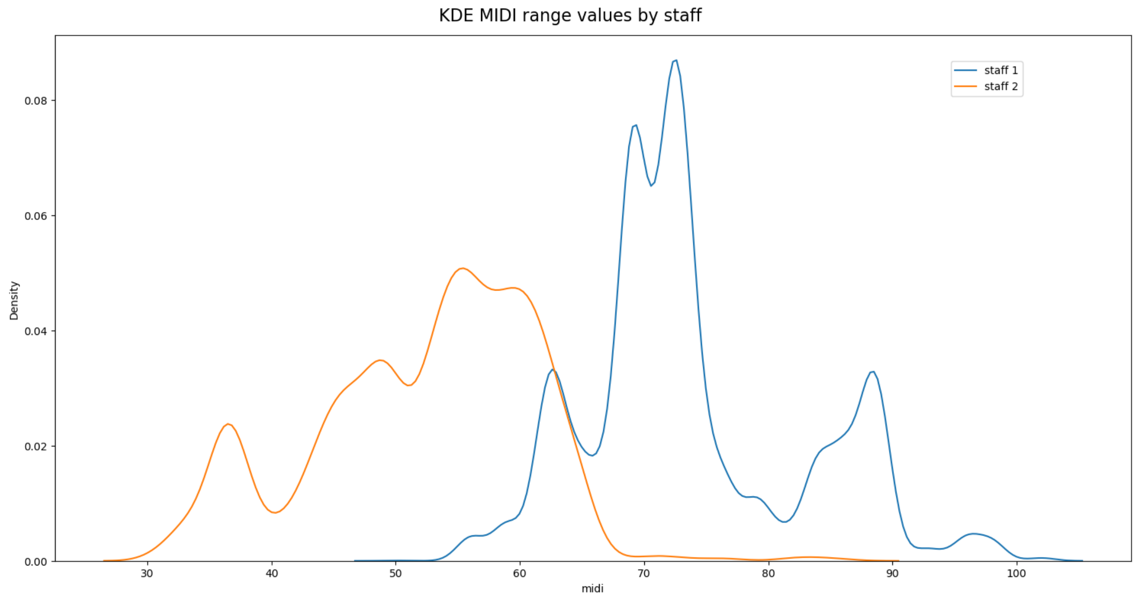


Figure 4.22: KDE MIDI keys range for each staff in the produced dataset.

The new data produced was compared with the original corpus using the Kolmogorov-Smirnov two-sided test. This test is intended to identify how different the produced tunes are compared with the training data. Table 4.2 shows test results by time signature and staff using the count of n-notes. Table 4.3 shows test results by notes duration. Table 4.4 shows test results by rests duration. Table 4.5 shows test results by MIDI keys.

The produced tunes are statistically different compared to the training corpus. This can be interpreted as new ideas for piano jazz tunes composing.

Time signature	Staff 1	Staff 2

2/2	p-value = 0.000
	statistic = 0.4269
		p-value = 0.000
		statistic = 0.8958

2/4	p-value = 0.000
	statistic = 0.4164
		p-value = 0.000
		statistic = 0.8958

4/4	p-value = 0.000
	statistic = 0.2564
		p-value = 0.000
		statistic = 0.8958

3/4	p-value = 0.000
	statistic = 0.3010
		p-value = 0.000
		statistic = 0.8958

5/4	p-value = 0.000
	statistic = 0.4988
		p-value = 0.000
		statistic = 0.8958

Table 4.2: n-notes distributions differences.

Time signature	Staff 1	Staff 2

2/2	p-value = 0.000
	statistic = 0.9162
		p-value = 0.000
		statistic = 0.8958

2/4	p-value = 0.000
	statistic = 0.9359
		p-value = 0.000
		statistic = 0.8958

4/4	p-value = 0.000
	statistic = 0.8658
		p-value = 0.000
		statistic = 0.8958

3/4	p-value = 0.000
	statistic = 0.8705
		p-value = 0.000
		statistic = 0.8958

5/4	p-value = 0.000
	statistic = 0.8771
		p-value = 0.000
		statistic = 0.8958

Table 4.3: Notes duration differences.

Time signature	Staff 1	Staff 2

2/2	p-value = 0.000
	statistic = 0.8421
		p-value = 0.000
		statistic = 0.8281

2/4	p-value = 0.000
	statistic = 0.8225
		p-value = 0.000
		statistic = 0.7859

4/4	p-value = 0.000
	statistic = 0.8009
		p-value = 0.000
		statistic = 0.7591

3/4	p-value = 0.000
	statistic = 0.8857
		p-value = 0.000
		statistic = 0.6286

5/4	p-value = 0.000
	statistic = 1.0000
		p-value = 0.000
		statistic = 0.3750

Table 4.4: Rests duration differences.

Staff 1	Staff 2

p-value = 0.000
statistic = 0.1299
	p-value = 0.000
	statistic = 0.0607

Table 4.5: MIDI keys differences.

Conclusions

Conclusions

Assisted composing using neural networks for jazz piano is achievable by defining a generative AI based on the same approach used to create text as a sequence of words. Jazz piano music sheets can be translated into the right format to train recurrent neural networks. These models can be implemented using cloud technologies such as Azure Machine Learning. Providing music with dynamic execution based on note duration from probability distributions and avoiding loops by implementing logic to identify them and select other probable n-notes in the sequence is a suitable approach to implementing assisted composing using neural networks for jazz piano. The produced tunes are different from the training corpus, which means that the produced ideas for piano jazz tune composing are innovative.

Bibliography

[1] AREVALO, E. *MuseScore-Web-Crawler.* https://github.com/EvaArevalo/Musescore-Web-Crawler, (accessed: 2023-03-16).

[2] DEVOTO, M. *sequence.* https://www.britannica.com/art/sequence-musical-composition, (accessed: 2023-03-02).

[3] GOLDBERG, Y. *Recurrent Neural Networks: Modeling Sequences and Stacks.* Springer International Publishing, Cham, 2017, pp. 164–166.

[4] HENTSCHEL, J. *Manual.* https://johentsch.github.io/ms3/build/html/manual/index.html, (accessed: 2023-03-17).

[5] IBM. *What is natural language processing?* https://www.ibm.com/topics/natural-language-processing, (accessed: 2023-03-03).

[6] JURAFSKY, D., AND MARTIN, J. H. Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition. third edition draft (october 16th, 2019). (accessed: 2023-02-28), 2019.

[7] MICROSOFT. *Regular Expression Language - Quick Reference.* https://learn.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference, (accessed: 2023-03-16).

[8] MICROSOFT. *RESTful web API design.* https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design, (accessed: 2023-03-02).

[9] MICROSOFT. *What is artificial intelligence?* https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-artificial-intelligence, (accessed: 2023-02-28).

[10] MORÁN MARTÍNEZ, M. C. Psicología y música: Inteligencia musical y desarrollo estético. *Revista Digital Universitaria [en línea] 10*, 11 (November 2009).

[11] OF AMERICAN HISTORY, N. M. *What is Jazz?* https://americanhistory.si.edu/smithsonian-jazz/education/what-jazz, (accessed: 2023-02-28).

[12] OPENAI. *MuseNet*. https://openai.com/research/musenet, (accessed: 2023-03-13).

[13] RESEARCH, G. *Magenta*. https://research.google/teams/brain/magenta/, (accessed: 2023-02-28).

[14] SKANSI, S. *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*. Undergraduate Topics in Computer Science. Springer International Publishing, 2018.

[15] WASKOM, M. *seaborn.kdeplot*. https://seaborn.pydata.org/generated/seaborn.kdeplot.html, (accessed: 2023-03-20).

[16] WIKIPEDIA. *Analog signal*. https://en.wikipedia.org/wiki/Analog_signal, (accessed: 2023-03-16).

[17] WIKIPEDIA. *MIDI*. https://en.wikipedia.org/wiki/MIDI, (accessed: 2023-03-16).

[18] ZHU, Q., AND LUO, J. Generative pre-trained transformer for design concept generation: An exploration. *Proceedings of the Design Society 2* (2022), 1825–1834.

APPENDIX A

Project Repository

Full code used for this project can be found in this GitHub repository.