**MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE GWALIOR**

(A Govt. Aided UGC Autonomous Institute Affiliated to RGPV, Bhopal)

*NAAC Accredited with A++ Grade*

**Project Report**

**on**

# "Random Melody Generator using long Short-term memory network"


**Submitted By:**

**Amey Kelkar**

**(0901AD211005)**

**Prakash Kurram**

**(0901AD211037)**

**Faculty Mentor:**

**Dr. Vibha Tiwari**

**Assistant Professor, Centre for Artificial Intelligence**



**CENTRE FOR ARTIFICIAL INTELLIGENCE**

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE

GWALIOR - 474005 (MP) est. 1957


JULY-DEC. 2023

**MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE GWALIOR**

(A Govt. Aided UGC Autonomous Institute Affiliated to RGPV, Bhopal)

***NAAC Accredited with A++ Grade***

# CERTIFICATE

This is certified that **Amey Kelkar** (0901AD211005) and **Prakash Kurram** (0901AD211037) has submitted the project report titled **Random melody generator using long short-term memory network (LSTM)** under the mentorship of **Dr. Vibha Tiwari**, in partial fulfilment of the requirement for the award of degree of Bachelor of Technology in **Artificial Intelligence and Data Science** from Madhav Institute of Technology and Science, Gwalior.

**Dr. Vibha Tiwari**
Faculty Mentor
Assistant Professor
Centre for Artificial Intelligence

**Dr. R. R. Singh**
Coordinator
Centre for Artificial Intelligence

**MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE GWALIOR**

(A Govt. Aided UGC Autonomous Institute Affiliated to RGPV, Bhopal)

***NAAC Accredited with A++ Grade***

# DECLARATION

I hereby declare that the work being presented in this project report, for the partial fulfilment of requirement for the award of the degree of Bachelor of Technology in **Artificial Intelligence and Data Science** at Madhav Institute of Technology & Science, Gwalior is an authenticated and original record of my work under the mentorship of **Dr. Vibha Tiwari**, **Assistant Professor**, Centre for Artificial Intelligence.

I declare that I have not submitted the matter embodied in this report for the award of any degree or diploma anywhere else.


Amey Kelkar
0901AD211005
III Year,
Centre for Artificial Intelligence

Prakash Kurram
0901AD211037
III Year,
Centre for Artificial Intelligence

**MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE GWALIOR**

(A Govt. Aided UGC Autonomous Institute Affiliated to RGPV, Bhopal)

***NAAC Accredited with A++ Grade***

# ACKNOWLEDGEMENT

The full semester project has proved to be pivotal to my career. I am thankful to my institute, **Madhav Institute of Technology and Science** to allow me to continue my disciplinary/interdisciplinary project as a curriculum requirement, under the provisions of the Flexible Curriculum Scheme (based on the AICTE Model Curriculum 2018), approved by the Academic Council of the institute. I extend my gratitude to the Director of the institute, **Dr. R. K. Pandit** and Dean Academics, **Dr. Manjaree Pandit** for this.

I would sincerely like to thank my department, **Centre for Artificial Intelligence,** for allowing me to explore this project. I humbly thank **Dr. R. R. Singh**, Coordinator, Centre for Artificial Intelligence, for his continued support during the course of this engagement, which eased the process and formalities involved.

I am sincerely thankful to my faculty mentor. I am grateful to the guidance of **Dr. Vibha Tiwari**, Assistant Professor, Centre for Artificial Intelligence, for her continued support and guidance throughout the project. I am also very thankful to the faculty and staff of the department.

Amey Kelkar
0901AD211005
III Year,
Centre for Artificial Intelligence

Prakash Kurram
0901AD211037
III Year,
Centre for Artificial Intelligence

# ABSTRACT

This project focuses on the development of a **Random Melody Generator** using long Short-term memory network, a type of recurrent neural network (RNN) known for its ability to capture long-range dependencies in sequential data. In order to produce fresh and varied melodies, the suggested RMG makes use of LSTM networks, giving musicians, composers, and music lovers a tool to experiment with new sound possibilities. The architecture of the RMG involves training the LSTM network on a dataset of existing melodies to learn the underlying patterns and structures. The trained model is then capable of generating original melodies by predicting the next notes in a sequence, taking into account the contextual information from the preceding notes. The randomness is introduced through the inherent nature of the LSTM's ability to capture intricate musical patterns while allowing for creative deviations.

**Keyword:** Random Melody Generator, Long Short-Term Memory Network, Artificial Intelligence, Music Composition, Recurrent Neural Network, Creative AI.

## सार:

यह परियोजना एक यादृच्छिक स्वर जेनरेटर के विकास पर केंद्रित है जिसमें लॉन्ग शॉर्ट-टर्म मेमोरी नेटवर्क का उपयोग किया जाता है, जो सीक्वेंशियल डेटा में लम्बी दूरी की ज़रूरतों को पकड़ने की क्षमता के लिए जाना जाता है, एक प्रकार के रिकरंट न्यूरल नेटवर्क (RNN) का हिस्सा है। ताजगी और विविधा वाले स्वरों को उत्पन्न करने के लिए, सुझाए गए आरएमजी ने एलएसटीएम नेटवर्क का उपयोग किया है, जो संगीतकारों, संगीत रचनाकारों, और संगीत प्रेमियों को नए ध्वनि संभावनाओं के साथ प्रयोग करने का एक उपकरण प्रदान करता है। आरएमजी की संरचना में एलएसटीएम नेटवर्क को मौजूदा स्वरों के डेटासेट पर प्रशिक्षित करने का शामिल है ताकि यह मौजूदा पैटर्न और संरचनाओं को सीख सके। प्रशिक्षित मॉडल फिर क्रम में आगे के स्वरों की पूर्वानुमान करके मौलिक स्वर सृजन करने की क्षमता रखता है, पिछले स्वरों से सांविदानिक जानकारी को ध्यान में रखते हुए। यह यादृच्छिकता एलएसटीएम की प्राकृतिक क्षमता के माध्यम से प्रस्तुत होती है जो सुरीले संगीत पैटर्न को पकड़ने के साथ-साथ रचनात्मक विचलनों की अनुमति देती है।

# TABLE OF CONTENTS

# LIST OF FIGURES

# Chapter 1: INTRODUCTION

In Today's world, music production is a field where any sound can be converted into music and the skills required to do so can be in any person. People who start their career in music sometimes are enable to learn the concepts of music or randomly produce any melody that does not follows the trends in the music industry. So, the authors have created a model which will give the user the ability to produce music without the actual knowledge of the notes and music production.

## 1.1 Project Aim:

The aim of this project is to explore the capabilities of Long Short-Term Memory (LSTM) neural networks in the context of music generation. The project seeks to develop a algorithm that leverages LSTM's ability to capture sequential dependencies in data to generate diverse and coherent musical melodies. By training the LSTM on a dataset of existing melodies, the goal is to enable the model to learn patterns and structures inherent in musical compositions, ultimately empowering it to produce original and aesthetically pleasing melodies autonomously. This project aims to contribute to the intersection of artificial intelligence and music composition, providing a creative tool for musicians and sparking insights into the potential of recurrent neural networks in the realm of generative art.

## 1.2 Objective:

The primary objective of this project is to design and implement an innovative algorithmic system that employs Long Short-Term Memory (LSTM) neural networks. The core objective is to harness the unique capabilities of LSTM networks for understanding and generating musical melodies. Through training on a diverse dataset of existing melodies, the model aims to learn intricate patterns and dependencies present in music, enabling it to autonomously compose novel and harmonious melodies. This project aspires to contribute to the field of artificial intelligence in music composition, providing a valuable tool for musicians to explore new creative possibilities and fostering a deeper understanding of the potential applications of recurrent neural networks in generative art.

# CHAPTER 2: LITERATURE SURVEY

Deep Learning is a field of Machine Learning which is inspired by a neural structure. These networks extract the features automatically from the dataset and are capable of learning any non-linear function. That's why Neural Networks are called as Universal Functional Approximators.

Hence, Deep Learning models are the state of the art in various fields like Natural Language Processing (NLP), Computer Vision, Speech Synthesis and so on.
There are many approaches to generate music AI.
Some of them are as follows
Approach-1 music generation using WaveNet (WaveNet is a Deep Learning-based generative model for raw audio developed by Google DeepMind).
Approach-2 music generation using long short-term memory network

Long Short-Term Memory Model, popularly known as LSTM, is a variant of Recurrent Neural Networks (RNNs) that is capable of capturing the long-term dependencies in the input sequence. LSTM has a wide range of applications in Sequence-to-Sequence modelling tasks like Speech Recognition, Text Summarization, Video Classification, and so on.

In this project, I used the concept of LSTM to create a model that can generate a whole melody starting from a certain set of notes and amplifying it again and again. The data preprocessing is the key factor in this project as we have to extract the notes and chords from the songs so that our model can be trained on it and by amplifying those notes and chords randomly, hence generate a whole melody.

# CHAPTER 3: PRELIMINARY DESIGN

## 3.1 LIBERARIES USED

Random melody generator using LSTM uses a wide range of python libraries to perform many import functions in this project. These are –

1. **music21 –** music21 is a python library toolkit that is used for computer – aided musicology. It contains various functions and packages that are used to handle the musical data and is a powerful and flexible toolkit for researchers, educators, and musicians interested in working with symbolic musical data in Python. It's particularly useful for tasks such as music analysis, manipulation, and basic composition.
2. **IPython -** IPython, short for "Interactive Python," is an enhanced interactive interface for the Python programming language. It provides an interactive shell with features that go beyond the standard Python interpreter, making it a powerful tool for interactive computing, data exploration, and scientific computing.
3. **numpy -** NumPy, short for Numerical Python, is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. NumPy serves as a foundational library for numerical and scientific computing in Python and is widely used in various fields, including data science, machine learning, physics, engineering, and more.
4. **Tensorflow -** TensorFlow is an open-source machine learning library developed by Google. Widely used for building and training deep learning models. It contains wide variety of machine learning tools and language models that are used in predictive analysis.
5. **Matplotlib** - Matplotlib is a popular Python data visualization library that enables the creation of high-quality charts, plots, and figures. With a user-friendly interface, it offers extensive customization options for creating diverse visualizations. Matplotlib is widely used in scientific computing, data analysis, and machine learning for presenting insights in a visually compelling manner.
6. **Pandas** - Pandas is a powerful data manipulation and analysis library for Python. It provides easy-to-use data structures, such as DataFrames, for efficient handling and exploration of structured data. With built-in functions for data cleaning, filtering, and aggregation, Pandas simplifies the process of working with tabular data, making it essential for data scientists and analysts.
7. **OS** - The `os' library is used for interacting with the operating system, facilitating tasks such as file operations and system-level interactions.

8. **Collections** - Collections offers specialized, high-performance alternatives to built-in data types. Featuring containers like Counter for counting elements, defaultdict for default values, and namedtuple for creating named tuples, it enhances data handling. This module is a valuable resource for streamlined data manipulation, providing versatile tools for efficient programming.
9. **Sklearn** - Scikit-learn (sklearn) is a comprehensive machine learning library for Python. With a simple and consistent interface, it offers a wide array of tools for classification, regression, clustering, and more. Sklearn supports model training, evaluation, and hyperparameter tuning, making it a go-to library for building and deploying machine learning model.

## 3.2 Data collection and preprocessing

In this project I have used the data that was available on the Kaggle.com as classical music MIDI. In this dataset there were samples based only on classical piano.

### 3.2.1 About the dataset:

This dataset consists of classical piano MIDI files of famous composers and in this folder, I have used the folder “Albeniz” to train my model on. The folder has 14 midi files that comprise of different melody samples created on piano.

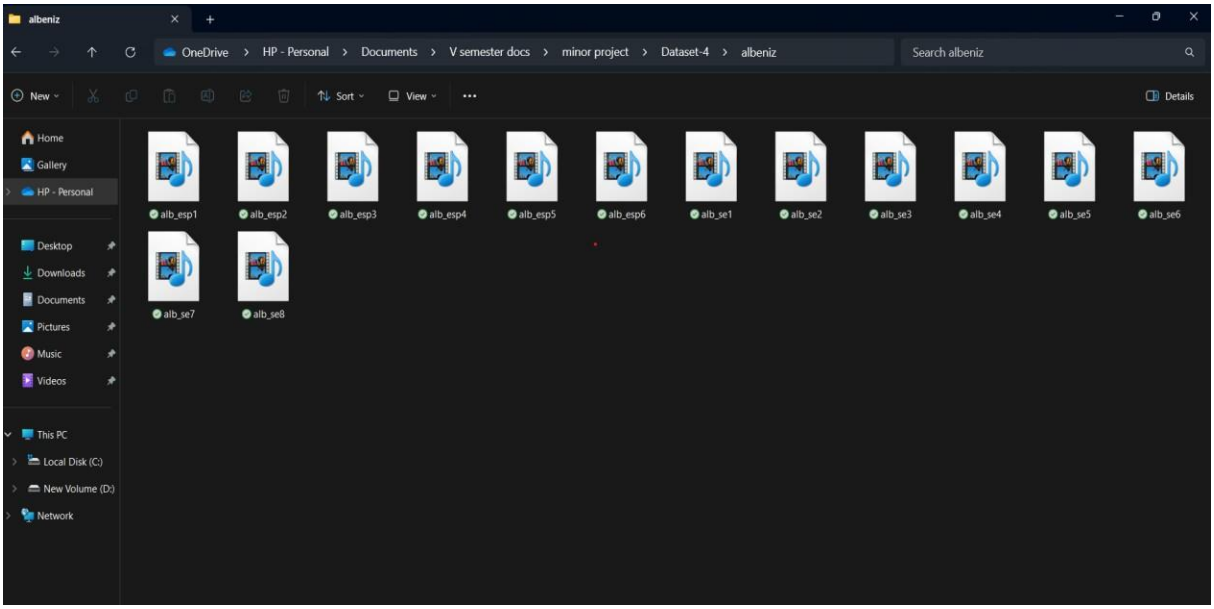


Figure 3.2.1

### 3.2.2 Loading data:

First, I have set a midi path where all the files of the dataset are stored and I have appended the address to all the files of that folder. Then I have appended the resultant string to a songs list.

```
midi_dataset_path = r"Dataset-4/albeniz"

songs = []
for path, subdirs, files in os.walk(midi_dataset_path):
    print(f"path : {path}")
    print(f"subdirs : {subdirs}")
    print(f"files : {files}")

    for file in files:
        if file[-3:] == "mid":
            song = converter.parse(os.path.join(path, file))
            songs.append(song)

print(songs)
```

Figure 3.2.2

### 3.2.3 extracting notes and chords from the songs

Now I define a function called `extract_notes()` that takes a list of music files (files) as input and extracts musical notes and chords from each file using the music21 library. It iterates through the parts of each instrument in the music, recursively extracts notes and chords, and appends them to the notes list. The resulting corpus is a list of string representations of pitches. The code then prints the total number of notes in the Bach dataset.

```
def extract_notes(files):
    notes = []
    pick = None
    for j in files:
        song = instrument.partitionByInstrument(j)
        for part in song.parts:
            pick = part.recurse()
            for element in pick:
                if isinstance(element, note.Note):
                    notes.append(str(element.pitch))
                elif isinstance(element, chord.Chord):
                    notes.append(".".join(str(n) for n in element.normalOrder))
    return notes

corpus = extract_notes(songs)
print(f"total notes in the bach dataset : {len(corpus)}")
```

Figure 3.2.3

The provided Python code defines a function named `chords_n_notes()` that takes a list of string representations of pitches (sample) as input and converts them into a musical stream using the music21 library. It iterates through the input, distinguishing between chords (indicated by the presence of a period or digits) and individual notes. For chords, it splits the string, creates note.Note instances for each note in the chord, assembles them into a chord.Chord, and appends the chord to the melody. For individual notes, it directly creates a note.Note and appends it to the melody. The resulting musical stream is then returned.

The code also demonstrates the function's usage by applying it to the first 50 elements of the corpus and storing the result in the variable x.

```
def chords_n_notes(sample):
    melody = []
    offset = 0

    for i in sample:
        if ("." in i) or (i.isdigit()):
            chord_notes = i.split(".")
            print(chord_notes)
            notes = []

            for j in chord_notes:
                inst_note = int(j)
                note_sam = note.Note(inst_note)
                notes.append(note_sam)
                chord_sam = chord.Chord(notes)
                chord_sam.offset = offset
                melody.append(chord_sam)
        else:
            note_sam = note.Note(i)
            note_sam.offset = offset
            melody.append(note_sam)

        offset += 1
    melody_stream = stream.Stream(melody)
    return melody_stream



x = chords_n_notes(corpus[:50])
```

Figure 3.2.4

It first counts the occurrences of each unique musical note in the corpus. The code then prints information about the corpus, including the total number of unique notes, the average recurrence of a note, the most frequent note and its count, and the least frequent note and its count.

```
count_num = Counter(corpus)
print(f"total unique notes in the corpus : {len(count_num)}")
notes = list(count_num.keys())
recurrence = list(count_num.values())

def avg(lst):
    return sum(lst)/len(lst)

print("Average recurrenc for a note in Corpus:", avg(recurrence))
print("Most frequent note in Corpus appeared:", max(recurrence), "times")
print("Least frequent note in Corpus appeared:", min(recurrence), "time")
```

Figure 3.2.5

The provided Python code utilizes the matplotlib library to create a histogram visualizing the frequency distribution of notes in the given corpus. It sets up a figure with a specific size and facecolor, defines bins for the histogram using np.arange, and then creates the histogram using plt.hist. Additionally, a vertical line is added at the x-value of 100 using plt.axvline. The title, xlabel, and ylabel are set to label the plot appropriately, and finally, the plot is displayed using plt.show(). The color scheme involves shades of blue for the bars and a pinkish color for the vertical line.

```
plt.figure(figsize=(18,3),facecolor="#97BACB")
bins = np.arange(0,(max(recurrence)), 50)
plt.hist(recurrence, bins=bins, color="#97BACB")
plt.axvline(x=100,color="#DBACC1")
plt.title("Frequency Distribution Of Notes In The Corpus")
plt.xlabel("Frequency Of Chords in Corpus")
plt.ylabel("Number Of Chords")
plt.show()
```

Figure 3.2.6

The provided Python code identifies and collects musical notes that occurred less than 50 times in the given corpus. It iterates through the items in the count_num dictionary, where each item represents a unique note and its corresponding frequency in the corpus. If the frequency (value) is less than 50, the note (key) is considered rare, and it is appended to the rare_notes list. The code then prints the total number of notes that occurred less than 50 times in the corpus.

```
rare_notes = []

for index, (key, value) in enumerate(count_num.items()):
    if value < 50:
        m = key
        rare_notes.append(m)

print(f"total number of notes that occured less than 50 times : {len(rare_notes)}")
```

Figure 3.2.7

The code below removes musical elements from the corpus that are identified as rare notes based on the rare_notes list. It iterates through each element in the corpus and removes elements that are found in the rare_notes list. After this filtering process, the code prints the length of the updated corpus. Finally, it generates a sorted list of unique symbols (symb) present in the modified corpus.

```
for element in corpus:
    if element in rare_notes:
        corpus.remove(element)

print(f"length of corpus after eliminating rare notes : {len(corpus)}")


symb = sorted(list(set(corpus)))
print(symb)
```

Figure 3.2.8

The code below creates sequences of length 40 from the corpus for the purpose of generating training data for a machine learning model. It iterates through the range of indices in the corpus, with each iteration forming a sequence of length 40 (features) and its corresponding target element (target). The sequences are converted into numerical representations using a mapping dictionary. The resulting features and targets are stored in the list's features and targets, respectively. The code then prints the total number of sequences generated from the corpus.

```
length = 40

features = []
targets = []
for i in range(0 , l_corpus - length, 1):
    feature = corpus[i : i + length]
    target = corpus[i + length]
    features.append([mapping[j] for j in feature])
    targets.append(mapping[target])


l_data_points = len(targets)
print(f"total number of sequences in the corpus : {l_data_points}")
```

Figure 3.2.9

The provided Python code prepares the features (X) and targets (y) for training a neural network using the TensorFlow library. It uses NumPy to reshape the features list into a three-dimensional array with dimensions (l_data_points, length, 1), where length represents the sequence length, and 1 signifies a single feature per element. The features are then normalized by dividing by the total number of unique symbols (l_symb). The targets are converted into categorical format using tensorflow.keras.utils.to_categorical.

```
X = (np.reshape(features, (l_data_points, length, 1)))/ float(l_symb)
y = tensorflow.keras.utils.to_categorical(targets)
```

# Chapter 4: FINAL ANALYSIS AND DESIGN

## 4.1 splitting the dataset for training and loss correction

The provided Python code uses the train_test_split function from scikit-learn to split the dataset into training and testing sets. It assigns the features (X) and targets (y) to training and testing sets, denoted as x_train, x_test, y_train, and y_test. The test_size parameter is set to 0.2, indicating that 20% of the data will be used for testing, and the random_state parameter ensures reproducibility by fixing the random seed.

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

## 4.2 Implementing LSTM model

The provided Python code defines a sequential model using Keras (assumed to be from TensorFlow). The model architecture consists of two LSTM layers with 512 and 256 units, respectively. The first LSTM layer is configured to return sequences, and a dropout layer with a dropout rate of 0.1 is added after it. The second LSTM layer is followed by a dense layer with 256 units and another dropout layer. The final dense layer has units equal to the number of classes in the target (y.shape[1]) and uses the softmax activation function.

The model is compiled using the Adamax optimizer with a learning rate of 0.01 and categorical cross-entropy as the loss function.

```
model = Sequential()

model.add(LSTM(512, input_shape = (X.shape[1], X.shape[2]), return_sequences = True))
model.add(Dropout(0.1))
model.add(LSTM(256))
model.add(Dense(256))
model.add(Dropout(0.1))
model.add(Dense(y.shape[1], activation = "softmax"))

model.compile(
    optimizer = Adamax(learning_rate = 0.01),
    loss = 'categorical_crossentropy',

)
```

Figure 4.2.1

The provided Python code trains the defined Keras model using the fit method. It uses the training data x_train and y_train with a batch size of 256 and trains the model for 200 epochs.

```
model.fit(x_train, y_train, batch_size = 256, epochs = 200)
```

The above LSTM model train on 200 epochs till the loss is minimized.

```
Epoch 1/200
49/49 [==============================] - 114s 2s/step - loss: 4.5861
Epoch 2/200
49/49 [==============================] - 134s 3s/step - loss: 4.3608
Epoch 3/200
49/49 [==============================] - 134s 3s/step - loss: 4.3517
Epoch 4/200
49/49 [==============================] - 138s 3s/step - loss: 4.3470
Epoch 5/200
49/49 [==============================] - 147s 3s/step - loss: 4.3472
Epoch 6/200
49/49 [==============================] - 152s 3s/step - loss: 4.3455
Epoch 7/200
49/49 [==============================] - 154s 3s/step - loss: 4.3407
Epoch 8/200
49/49 [==============================] - 155s 3s/step - loss: 4.3417
Epoch 9/200
49/49 [==============================] - 155s 3s/step - loss: 4.3392
Epoch 10/200
49/49 [==============================] - 150s 3s/step - loss: 4.3429
Epoch 11/200
49/49 [==============================] - 150s 3s/step - loss: 4.3415
Epoch 12/200
49/49 [==============================] - 127s 3s/step - loss: 4.3385
Epoch 13/200
49/49 [==============================] - 71s 1s/step - loss: 4.3378
Epoch 14/200
49/49 [==============================] - 74s 2s/step - loss: 4.3398
Epoch 15/200
49/49 [==============================] - 73s 1s/step - loss: 4.3378
Epoch 16/200
49/49 [==============================] - 73s 1s/step - loss: 4.3382
```

Figure 4.2.2

The provided Python code calls the Melody_Generator function with a specified Note_Count of 75. It initializes the generation process with a random seed from the test set (x_test), predicts the next note in the sequence iteratively, and diversifies the predictions using a temperature parameter. The generated notes are then converted back to their original representations using a reverse mapping.

The resulting values are stored in the variable's music_notes and melody. music_notes is a list of the generated notes, and melody is a musical stream created from these notes.

```
def Malody_Generator(Note_Count):
    seed = x_test[np.random.randint(0,len(x_test)-1)]
    Music = ""
    Notes_Generated=[]
    for i in range(Note_Count):
        seed = seed.reshape(1,length,1)
        prediction = model.predict(seed, verbose=0)[0]
        prediction = np.log(prediction) / 1.0 #diversity
        exp_preds = np.exp(prediction)
        prediction = exp_preds / np.sum(exp_preds)
        index = np.argmax(prediction)
        index_N = index/ float(l_symb)
        Notes_Generated.append(index)
        Music = [reversed_mapping[char] for char in Notes_Generated]
        seed = np.insert(seed[0],len(seed[0]),index_N)
        seed = seed[1:]

    Melody = chords_n_notes(Music)
    Melody_midi = stream.Stream(Melody)
    return Music,Melody_midi

music_notes, melody = Malody_Generator(75)
```

Figure 4.2.3

## 4.3 Storing the random melody generated

```
melody.write('midi', 'melody_generated_using_lstm-3.mid')
```

The above code writes the generated melody to a MIDI file named 'melody_generated_using_lstm-3.mid'. The write method is applied to the melody object, specifying the output format as MIDI and providing the desired filename for the generated MIDI file.

The below image shows the melody generated in the folder whose path is given

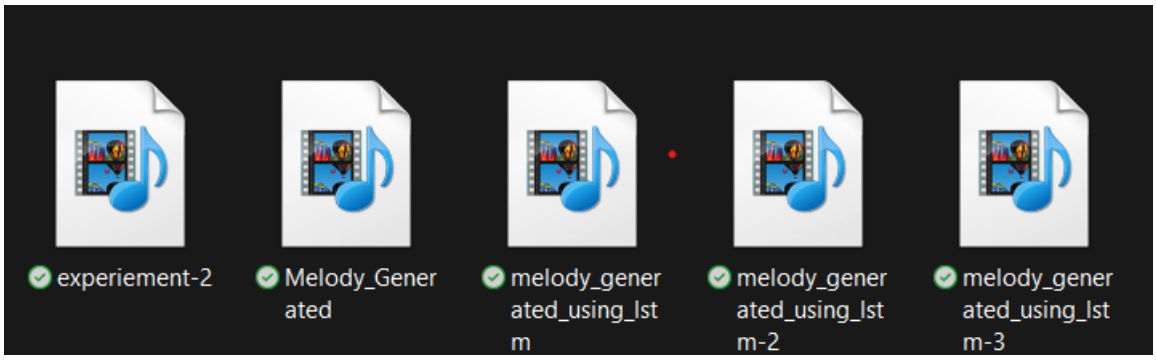


Figure 4.2.4

## 4.4 Application:

The application of the provided code lies in generating melodies using a Long Short-Term Memory (LSTM) neural network trained on a music corpus. This has potential applications in music composition, where AI can assist or inspire composers by generating new musical sequences based on learned patterns from existing compositions.

## 4.5 Problems Faced:

1. **Data Preparation:** The code assumes the availability of a well-prepared music dataset (**corpus**). Creating such datasets with high-quality and diverse musical content can be challenging.
2. **Model Training:** The performance of the generated melodies heavily depends on the quality and quantity of the training data, and tuning hyperparameters is crucial for optimal results.
3. **Sequence Generation:** Balancing creativity and structure in generated sequences can be challenging. Adjusting parameters like temperature during sequence generation can impact the diversity of the generated melodies.

## 4.6 Limitations:

1. **Musical Quality:** While the model can learn patterns, generating musically coherent and aesthetically pleasing melodies is a complex task. The generated sequences may lack the nuanced creativity of a human composer.
2. **Dependency on Training Data:** The model's effectiveness is highly dependent on the quality and representativeness of the training dataset. Biases and limitations in the dataset may be reflected in the generated melodies.
3. **Overfitting:** If the model is trained on a limited dataset, it may memorize specific sequences rather than learning general musical patterns, leading to overfitting.

## 4.7 Conclusion:

The code demonstrates the use of LSTM networks for music generation, showcasing the potential of AI in creative tasks. Despite facing challenges related to data quality, training, and sequence generation, the approach provides a foundation for exploring AI's role in music composition. Continued research and refinement of models, coupled with diverse and comprehensive datasets, can contribute to further advancements in AI-generated music.

# REFERENCE


[1] https://www.kaggle.com/code/karnikakapoor/music-generation-lstm

[2] https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5

[3] https://david-exiga.medium.com/music-generation-using-lstm-neural-networks-44f6780a4c5

[4] https://www.tensorflow.org/

[5] https://numpy.org/