MUSIFY- Music Composition using A.I.

Music Composition

Music Composition is a process of creating a new piece of music

Composition means "putting together". Thus, music composition is something where music notes are put together in such a way that it gives pleasant sensation to our ears

Parameters such as pitch interval, notes, chords, tempo etc. are used for composing short piece of music

About the Project

- The Project mainly focusses on music from **Piano** instrument
- Uses **Long Short Term Memory** (LSTM) , a type of Recurrent Neural Network (RNN)
- Platform: **Google Colab**
- Language: **Python 3.8**
- Libraries Used: **Tensorflow, Music21, Keras, NumPy, Sklearn, tqdm**
- Dataset: Classical Music MIDI | Kaggle

Terminologies

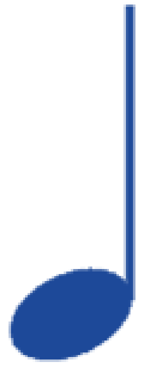
- **Note**: This is a sound produced by a single key
- **Chords**: The combination of 2 or more notes is called a chord
- **Octave**: The distance between two notes is stated as an octave in a piano
  It is specifically the gap between the two notes that share the same letter name
- **Recurrent Neural Networks (RNN)**
  A recurrent neural network is a class of artificial neural networks that make use of sequential information. They are called recurrent because they perform the same function for every single element of a sequence, with the result being dependent on previous computations
- **Long Short Term Memory (LSTM)**
  - LSTMs are a type of Recurrent Neural Network that can efficiently learn via gradient descent
  - Using a gating mechanism, they are able to recognize and encode long-term patterns
  - Useful to solve problems where the network has to remember information for a long period of time
  - Applications: Music and text generation etc.
  - Limitation: It requires lots of resources and time to get trained for real world applications

Libraries

- **Music21**
  - Music21 is a Python toolkit used for computer-aided musicology
  - It allows us to teach the fundamentals of music theory, generate music examples and study music
  - The toolkit provides a simple interface to acquire the musical notation of MIDI files
  - Additionally, it allows us to create Note and Chord objects so that we can make our own MIDI files easily
- **Keras**
  - Keras is a high-level neural networks API that simplifies interactions with TensorFlow
  - It was developed with a focus on enabling fast experimentation
- **TensorFlow**
  - TensorFlow is a free and open-source software library for machine learning and artificial intelligence
  - It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks
- **NumPy**
  - NumPy is a Python library used for working with arrays
  - It also has functions for working in domain of linear algebra, Fourier transform, and matrices
- **tqdm**
  - *tqdm* is a library in *Python* which is used for creating Progress Meters or Progress Bars

Project Structure

- **All Midi Files/** : This is the dataset folder containing various midi files of different composers
- **code.ipynb** : In this file, we will build, train and test our model
- **MOD/** : This directory contains optimizer, metrics, and weights of our trained model
- **AI_composed_music.mid**: This is a music file of predicted notes



STEPS

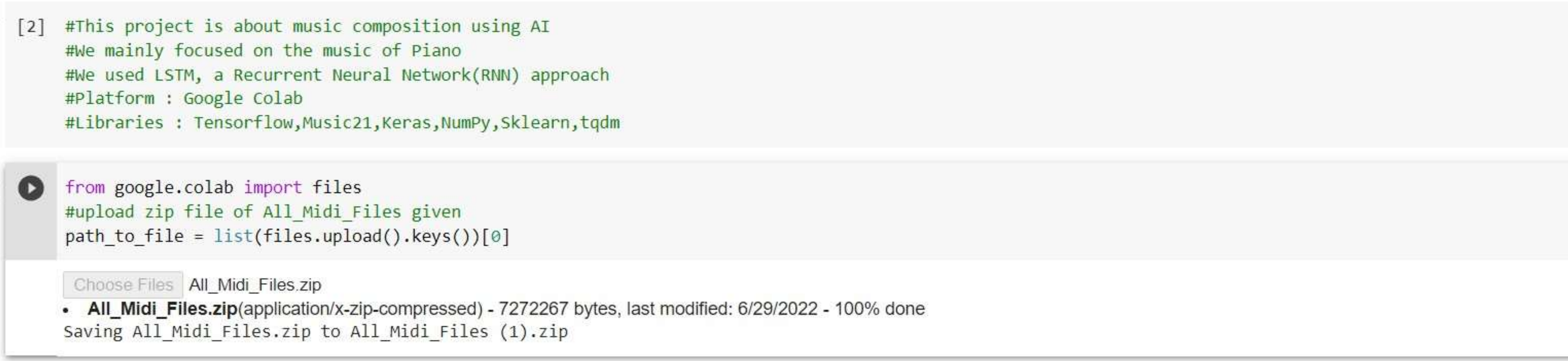
(For developing Code from Scratch)

STEP 1

Choosing zip file

- To load midi files in the code, I need to load the data first to the google colab session
- I need to select the zip file of all music files

CODE SNIPPET



```
[2] #This project is about music composition using AI
    #We mainly focused on the music of Piano
    #We used LSTM, a Recurrent Neural Network(RNN) approach
    #Platform : Google Colab
    #Libraries : Tensorflow,Music21,Keras,NumPy,Sklearn,tqdm
```

```
from google.colab import files
#upload zip file of All_Midi_Files given
path_to_file = list(files.upload().keys())[0]
```

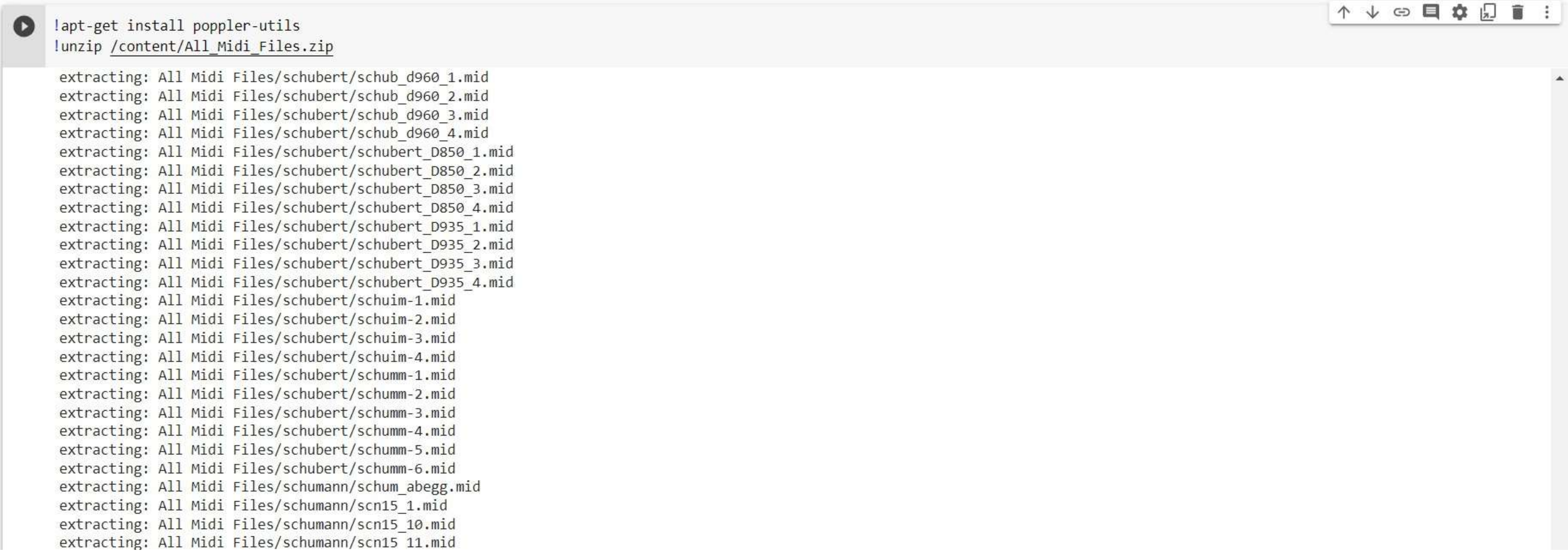
```
Choose Files  All_Midi_Files.zip
• All_Midi_Files.zip(application/x-zip-compressed) - 7272267 bytes, last modified: 6/29/2022 - 100% done
Saving All_Midi_Files.zip to All_Midi_Files (1).zip
```

STEP 2

Extracting zip file

- After uploading the zip file, I need to extract the music files in the session itself.

CODE SNIPPET



```
!apt-get install poppler-utils
!unzip /content/All_Midi_Files.zip
```

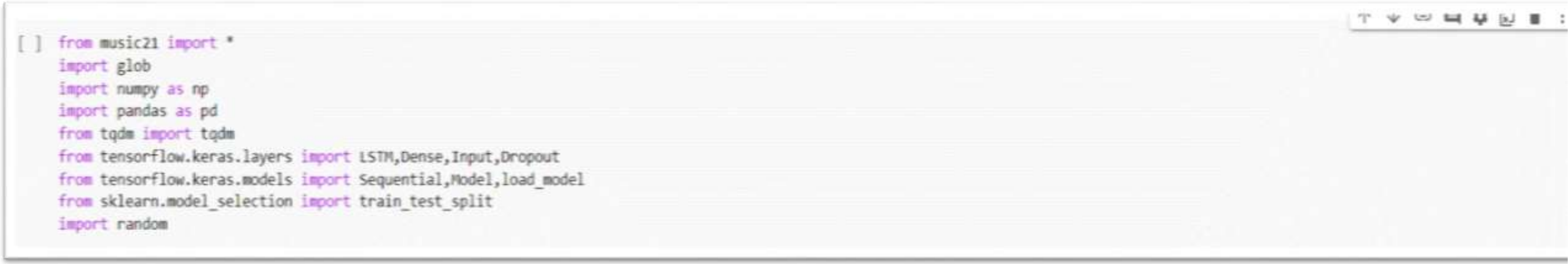
```
extracting: All Midi Files/schubert/schub_d960_1.mid
extracting: All Midi Files/schubert/schub_d960_2.mid
extracting: All Midi Files/schubert/schub_d960_3.mid
extracting: All Midi Files/schubert/schub_d960_4.mid
extracting: All Midi Files/schubert/schubert_D850_1.mid
extracting: All Midi Files/schubert/schubert_D850_2.mid
extracting: All Midi Files/schubert/schubert_D850_3.mid
extracting: All Midi Files/schubert/schubert_D850_4.mid
extracting: All Midi Files/schubert/schubert_D935_1.mid
extracting: All Midi Files/schubert/schubert_D935_2.mid
extracting: All Midi Files/schubert/schubert_D935_3.mid
extracting: All Midi Files/schubert/schubert_D935_4.mid
extracting: All Midi Files/schubert/schuim-1.mid
extracting: All Midi Files/schubert/schuim-2.mid
extracting: All Midi Files/schubert/schuim-3.mid
extracting: All Midi Files/schubert/schuim-4.mid
extracting: All Midi Files/schubert/schumm-1.mid
extracting: All Midi Files/schubert/schumm-2.mid
extracting: All Midi Files/schubert/schumm-3.mid
extracting: All Midi Files/schubert/schumm-4.mid
extracting: All Midi Files/schubert/schumm-5.mid
extracting: All Midi Files/schubert/schumm-6.mid
extracting: All Midi Files/schumann/schum_abegg.mid
extracting: All Midi Files/schumann/scn15_1.mid
extracting: All Midi Files/schumann/scn15_10.mid
extracting: All Midi Files/schumann/scn15_11.mid
```

STEP 3

Import Libraries

- Import all the important Libraries

CODE SNIPPET

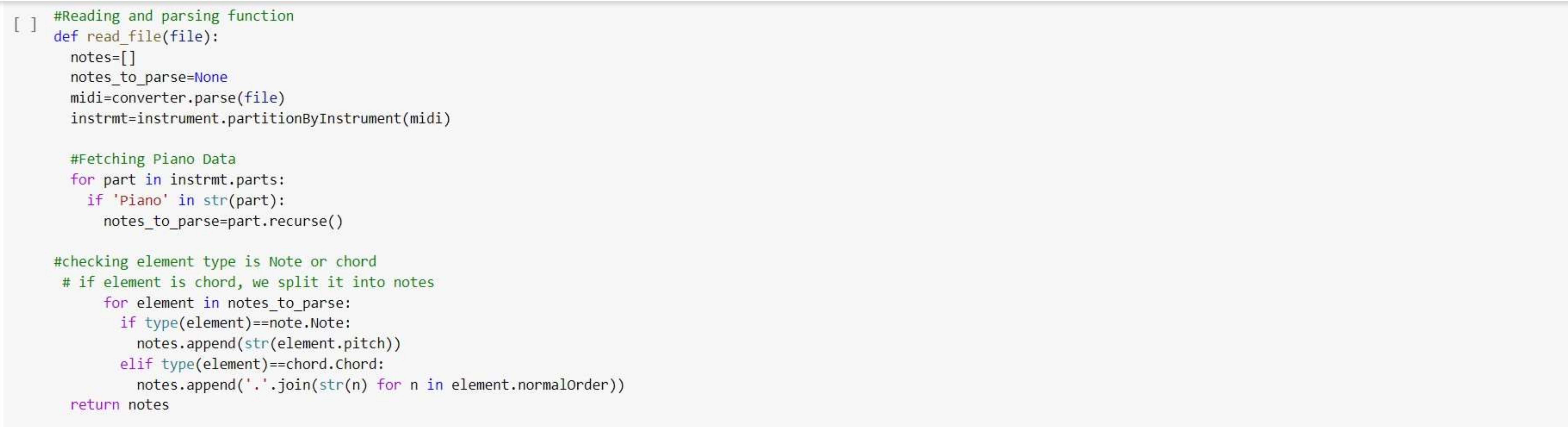


```
[ ] from music21 import *
    import glob
    import numpy as np
    import pandas as pd
    from tqdm import tqdm
    from tensorflow.keras.layers import LSTM,Dense,Input,Dropout
    from tensorflow.keras.models import Sequential,Model,load_model
    from sklearn.model_selection import train_test_split
    import random
```

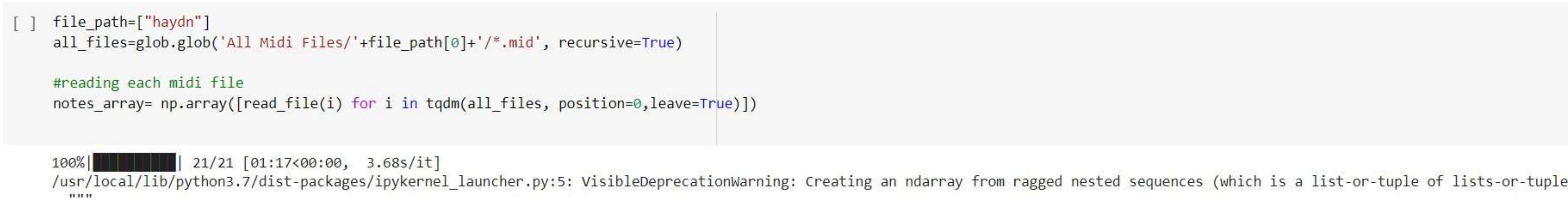
STEP 4 Reading and Parsing the Midi File

- The midi file dataset has to be read using Music21 library
- “**Haydn**” composed files has been used. (You can use more or less depending on your system)
- For this project, the files that contain sequential streams of **Piano** data has only been worked on
- All files are separated by their instruments and **Piano** is used only
- Piano stream from the midi file contains many data like **Keys**, **Time Signature**, **Chord**, **Note** etc.
- We require only **Notes** and **Chords** to generate music
- Lastly, the arrays of notes and chords has to be returned

CODE SNIPPET



```
[ ] #Reading and parsing function
    def read_file(file):
      notes=[]
      notes_to_parse=None
      midi=converter.parse(file)
      instrmt=instrument.partitionByInstrument(midi)

      #Fetching Piano Data
      for part in instrmt.parts:
        if 'Piano' in str(part):
          notes_to_parse=part.recurse()

    #checking element type is Note or chord
     # if element is chord, we split it into notes
          for element in notes_to_parse:
            if type(element)==note.Note:
              notes.append(str(element.pitch))
            elif type(element)==chord.Chord:
              notes.append('.'.join(str(n) for n in element.normalOrder))
      return notes
```



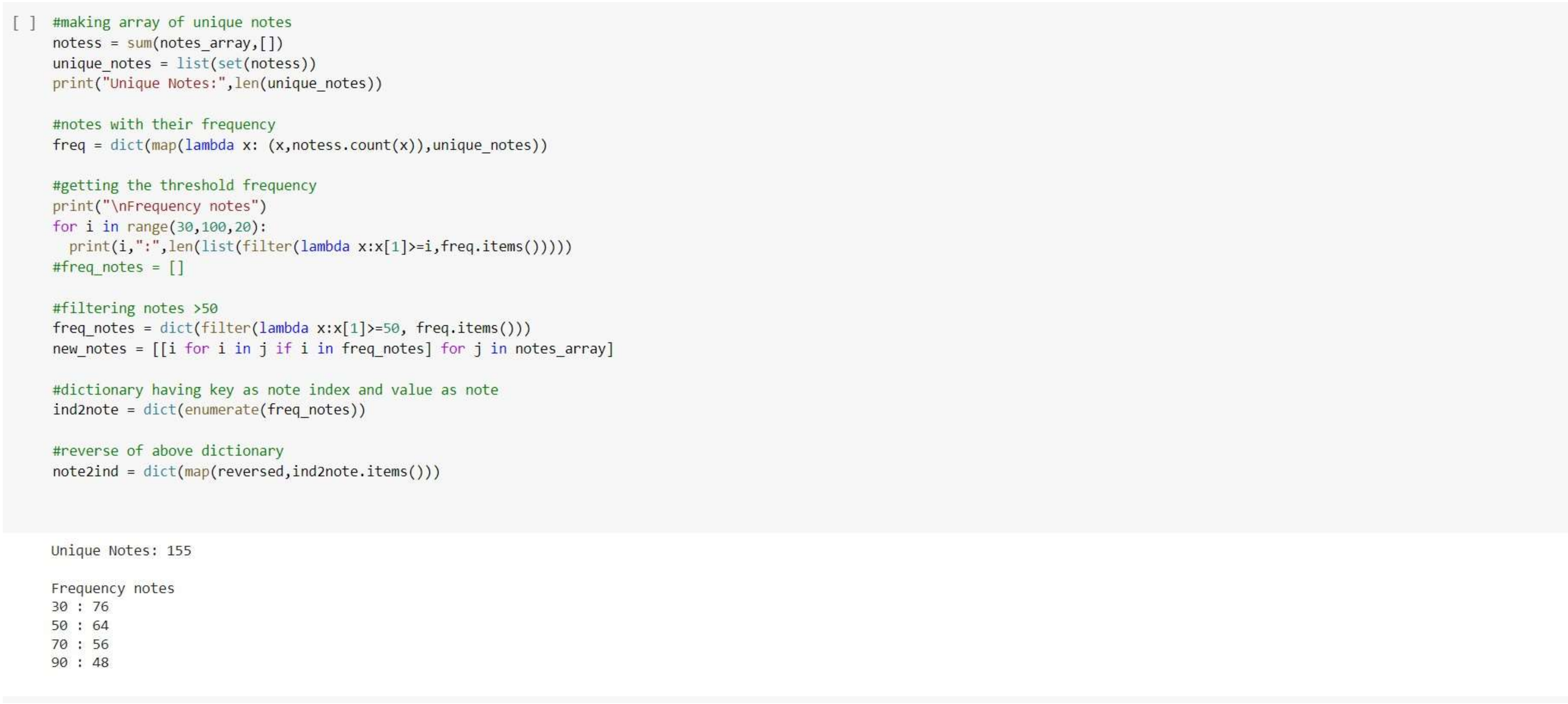
```
[ ] file_path=["haydn"]
    all_files=glob.glob('All Midi Files/'+file_path[0]+'/*.mid', recursive=True)

    #reading each midi file
    notes_array= np.array([read_file(i) for i in tqdm(all_files, position=0,leave=True)])

    100%|██████████| 21/21 [01:17<00:00,  3.68s/it]
    /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuple
      """
```

STEP 5

Exploring the Dataset

- This is done to check the number of **unique notes** and their **distribution**
- **50** is used as a threshold frequency
- Only those notes which have frequencies more than **50** have been considered
- **Two** dictionaries are created where one will have **notes index as a key and notes as value** and other will be the reverse of the first i.e. **key will be notes and value will be its respective index**
- These dictionaries will be used in the next steps

CODE SNIPPET



```
[ ] #making array of unique notes
    notess = sum(notes_array,[])
    unique_notes = list(set(notess))
    print("Unique Notes:",len(unique_notes))

    #notes with their frequency
    freq = dict(map(lambda x: (x,notess.count(x)),unique_notes))

    #getting the threshold frequency
    print("\nFrequency notes")
    for i in range(30,100,20):
      print(i,":",len(list(filter(lambda x:x[1]>=i,freq.items()))))
    #freq_notes = []

    #filtering notes >50
    freq_notes = dict(filter(lambda x:x[1]>=50, freq.items()))
    new_notes = [[i for i in j if i in freq_notes] for j in notes_array]

    #dictionary having key as note index and value as note
    ind2note = dict(enumerate(freq_notes))

    #reverse of above dictionary
    note2ind = dict(map(reversed,ind2note.items()))
```

```
Unique Notes: 155

Frequency notes
30 : 76
50 : 64
70 : 56
90 : 48
```

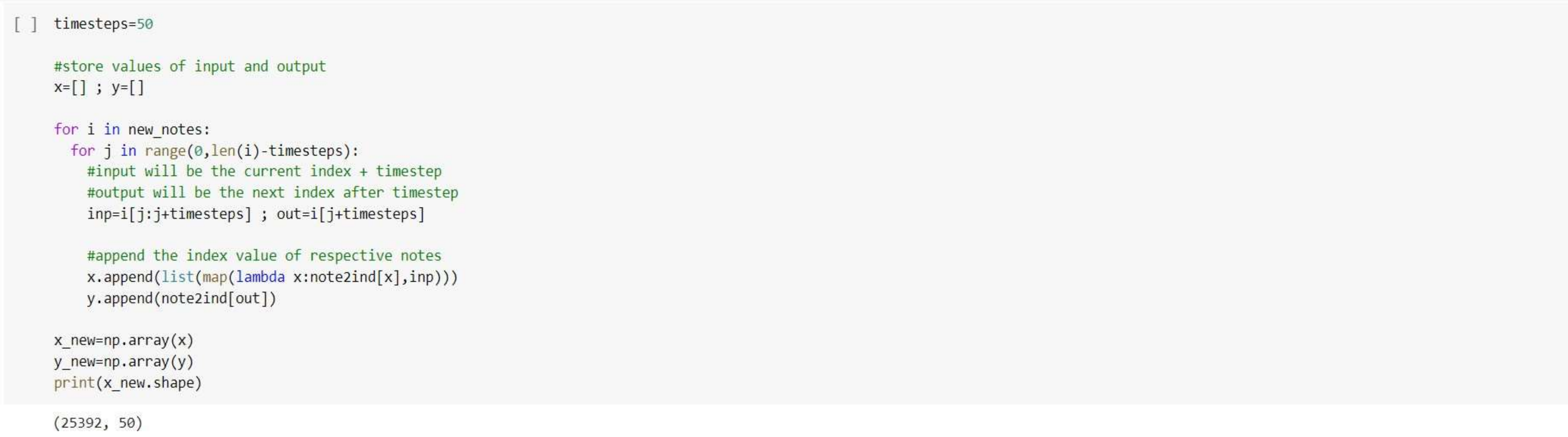
STEP 4 Input and Output Sequence for model

- **Input** and **output** sequences for our model are created
- A **timestep** of **50** has been used. So, if we traverse 50 notes of our input sequence then the **51st** note will be the output for that sequence
- **Example**:
  - While using ‘SOC stands for Seasons of Code’ sentence with a timestep of 2, we will have to provide 2 words at every input to get the output

(x)	**(y)**
SOC stands	for
Stands for	Seasons
for Seasons	of
Seasons of	Code

- As our model requires numeric data, all notes are converted to its respective index value using the “**note2ind**” (note to index) dictionary which has been created earlier

CODE SNIPPET

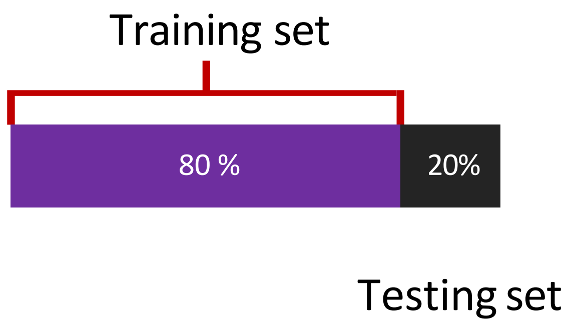


```
[ ]  timesteps=50

     #store values of input and output
     x=[] ; y=[]

     for i in new_notes:
       for j in range(0,len(i)-timesteps):
         #input will be the current index + timestep
         #output will be the next index after timestep
         inp=i[j:j+timesteps] ; out=i[j+timesteps]

         #append the index value of respective notes
         x.append(list(map(lambda x:note2ind[x],inp)))
         y.append(note2ind[out])

     x_new=np.array(x)
     y_new=np.array(y)
     print(x_new.shape)
```

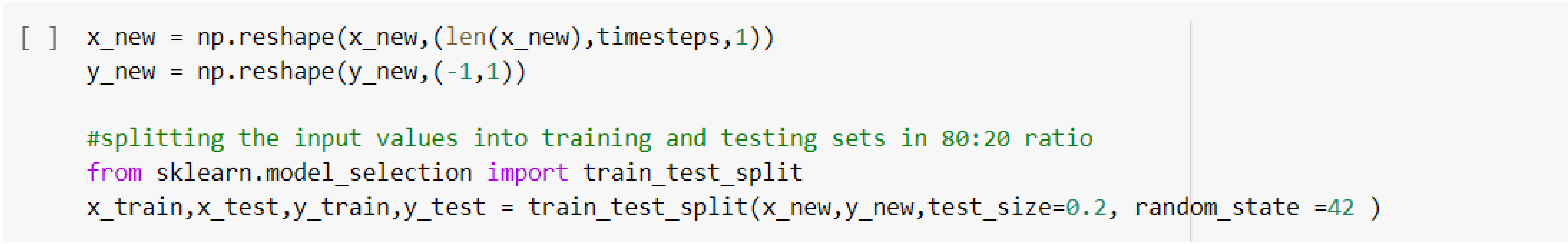
```
(25392, 50)
```

STEP 5 Training and Testing sets

- Array for our model is re-shaped and the data is split into 80:20 ratio.



CODE SNIPPET



```
[ ]  x_new = np.reshape(x_new,(len(x_new),timesteps,1))
     y_new = np.reshape(y_new,(-1,1))

     #splitting the input values into training and testing sets in 80:20 ratio
     from sklearn.model_selection import train_test_split
     x_train,x_test,y_train,y_test = train_test_split(x_new,y_new,test_size=0.2, random_state =42 )
```

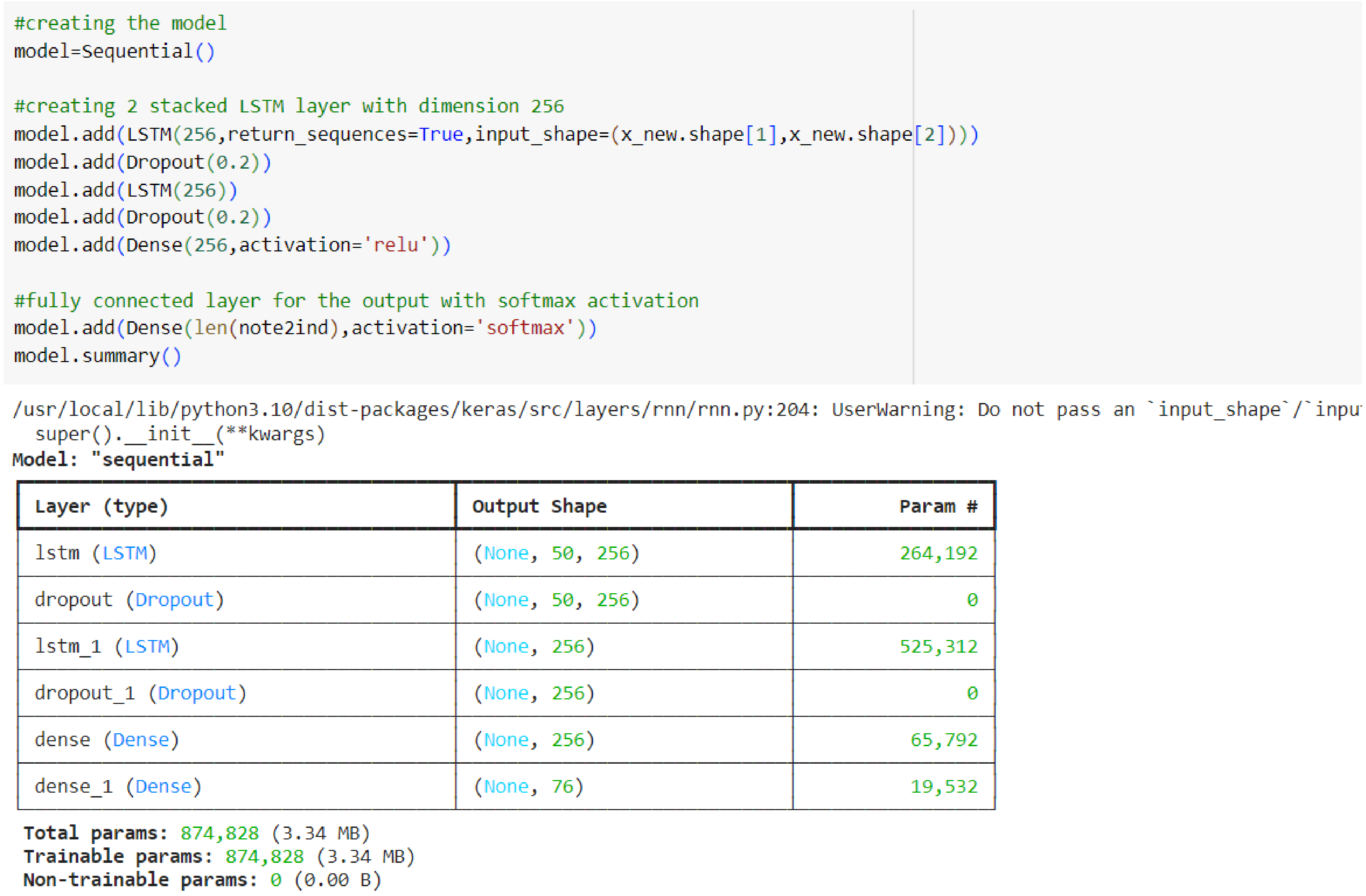
STEP 6 Building the Model

- 2 stacked **LSTM** layers with a dropout rate of **0.2** are used
- A fully connected **Dense** layer has been used for output
- Output dimension of the Dense Layer is taken same as the length of our unique notes along with the '**softmax**' activation function (Used for multi-class classification problems)

Dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random. It basically prevents overfitting while training the model, while it does not affect the inference model.

CODE SNIPPET

Building the Model using LSTMs

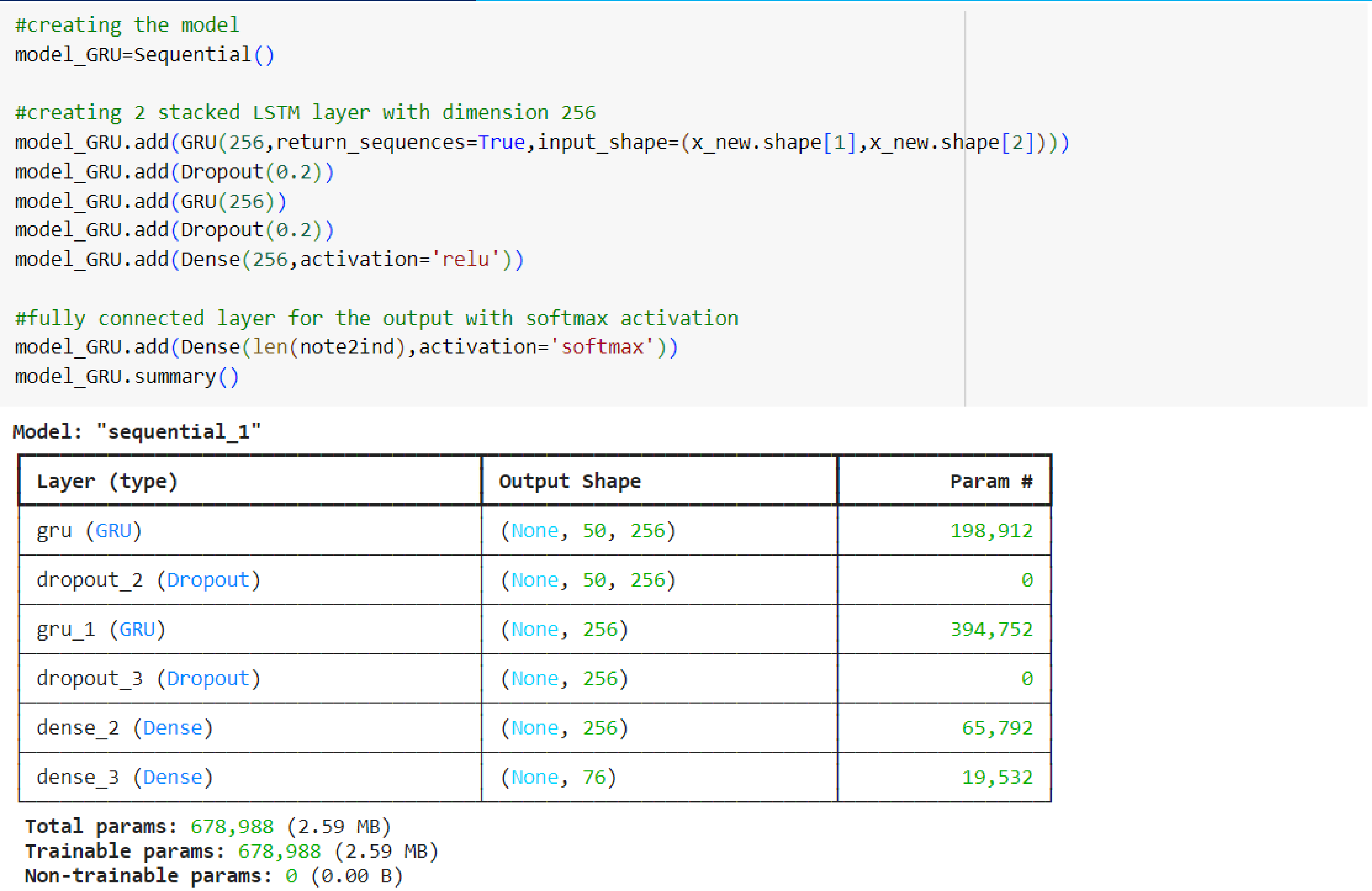


```python
#creating the model
model=Sequential()

#creating 2 stacked LSTM layer with dimension 256
model.add(LSTM(256,return_sequences=True,input_shape=(x_new.shape[1],x_new.shape[2])))
model.add(Dropout(0.2))
model.add(LSTM(256))
model.add(Dropout(0.2))
model.add(Dense(256,activation='relu'))

#fully connected layer for the output with softmax activation
model.add(Dense(len(note2ind),activation='softmax'))
model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(**kwargs)
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ lstm (LSTM)                          │ (None, 50, 256)             │         264,192 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout (Dropout)                    │ (None, 50, 256)             │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ lstm_1 (LSTM)                        │ (None, 256)                 │         525,312 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_1 (Dropout)                  │ (None, 256)                 │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense (Dense)                        │ (None, 256)                 │          65,792 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_1 (Dense)                      │ (None, 76)                  │          19,532 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 874,828 (3.34 MB)
 Trainable params: 874,828 (3.34 MB)
 Non-trainable params: 0 (0.00 B)
```

CODE SNIPPET

Building the Model using GRUs



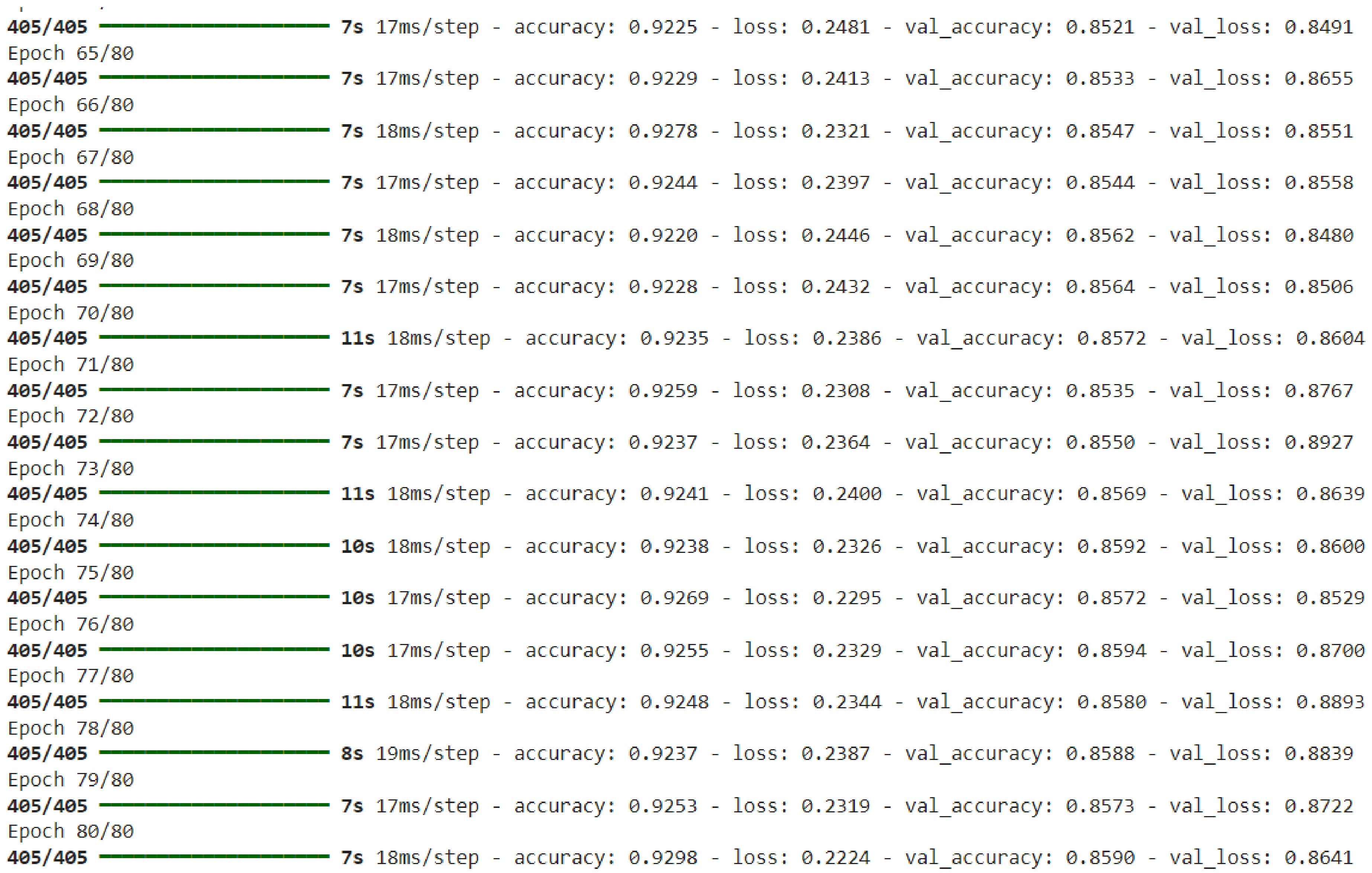
```
#creating the model
model_GRU=Sequential()

#creating 2 stacked LSTM layer with dimension 256
model_GRU.add(GRU(256,return_sequences=True,input_shape=(x_new.shape[1],x_new.shape[2])))
model_GRU.add(Dropout(0.2))
model_GRU.add(GRU(256))
model_GRU.add(Dropout(0.2))
model_GRU.add(Dense(256,activation='relu'))

#fully connected layer for the output with softmax activation
model_GRU.add(Dense(len(note2ind),activation='softmax'))
model_GRU.summary()
```

```
Model: "sequential_1"
┌───────────────────────────────┬───────────────────────────┬─────────────────┐
│ Layer (type)                  │ Output Shape              │         Param # │
├───────────────────────────────┼───────────────────────────┼─────────────────┤
│ gru (GRU)                     │ (None, 50, 256)           │         198,912 │
├───────────────────────────────┼───────────────────────────┼─────────────────┤
│ dropout_2 (Dropout)           │ (None, 50, 256)           │               0 │
├───────────────────────────────┼───────────────────────────┼─────────────────┤
│ gru_1 (GRU)                   │ (None, 256)               │         394,752 │
├───────────────────────────────┼───────────────────────────┼─────────────────┤
│ dropout_3 (Dropout)           │ (None, 256)               │               0 │
├───────────────────────────────┼───────────────────────────┼─────────────────┤
│ dense_2 (Dense)               │ (None, 256)               │          65,792 │
├───────────────────────────────┼───────────────────────────┼─────────────────┤
│ dense_3 (Dense)               │ (None, 76)                │          19,532 │
└───────────────────────────────┴───────────────────────────┴─────────────────┘
 Total params: 678,988 (2.59 MB)
 Trainable params: 678,988 (2.59 MB)
 Non-trainable params: 0 (0.00 B)
```

STEP 7 Training the Model

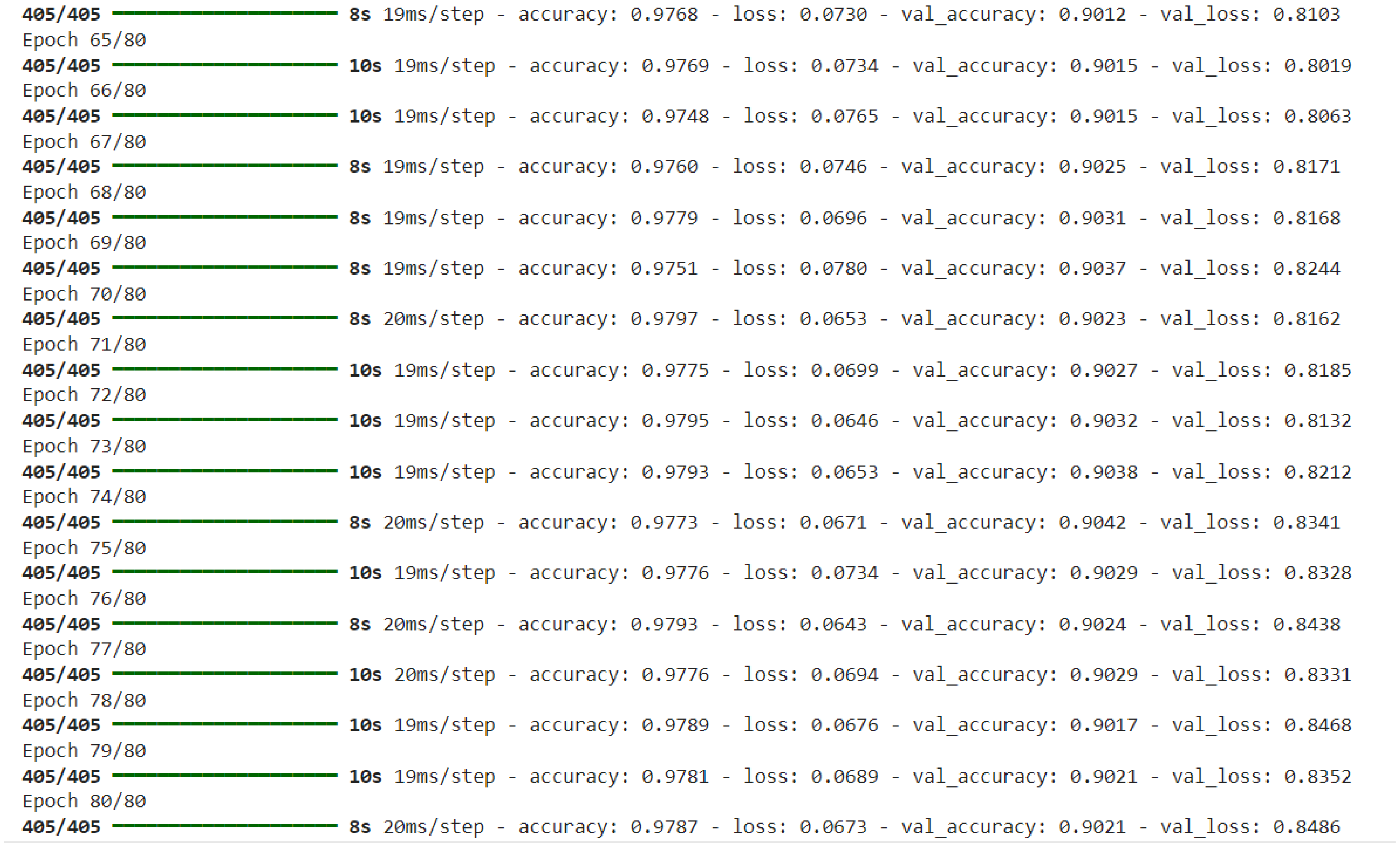
- After building the model, it is trained on the input and output data
- For this, ‘**Adam**’ optimizer is used on **batch size** of **128** and for total **80 epochs**
- After Training, model is **saved** for prediction

RESULTS FOR GRUs



```
405/405 ━━━━━━━━━━━━━━━━━━━━ 7s 17ms/step - accuracy: 0.9225 - loss: 0.2481 - val_accuracy: 0.8521 - val_loss: 0.8491
Epoch 65/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 7s 17ms/step - accuracy: 0.9229 - loss: 0.2413 - val_accuracy: 0.8533 - val_loss: 0.8655
Epoch 66/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 7s 18ms/step - accuracy: 0.9278 - loss: 0.2321 - val_accuracy: 0.8547 - val_loss: 0.8551
Epoch 67/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 7s 17ms/step - accuracy: 0.9244 - loss: 0.2397 - val_accuracy: 0.8544 - val_loss: 0.8558
Epoch 68/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 7s 18ms/step - accuracy: 0.9220 - loss: 0.2446 - val_accuracy: 0.8562 - val_loss: 0.8480
Epoch 69/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 7s 17ms/step - accuracy: 0.9228 - loss: 0.2432 - val_accuracy: 0.8564 - val_loss: 0.8506
Epoch 70/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - accuracy: 0.9235 - loss: 0.2386 - val_accuracy: 0.8572 - val_loss: 0.8604
Epoch 71/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 7s 17ms/step - accuracy: 0.9259 - loss: 0.2308 - val_accuracy: 0.8535 - val_loss: 0.8767
Epoch 72/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 7s 17ms/step - accuracy: 0.9237 - loss: 0.2364 - val_accuracy: 0.8550 - val_loss: 0.8927
Epoch 73/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - accuracy: 0.9241 - loss: 0.2400 - val_accuracy: 0.8569 - val_loss: 0.8639
Epoch 74/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.9238 - loss: 0.2326 - val_accuracy: 0.8592 - val_loss: 0.8600
Epoch 75/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 10s 17ms/step - accuracy: 0.9269 - loss: 0.2295 - val_accuracy: 0.8572 - val_loss: 0.8529
Epoch 76/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 10s 17ms/step - accuracy: 0.9255 - loss: 0.2329 - val_accuracy: 0.8594 - val_loss: 0.8700
Epoch 77/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - accuracy: 0.9248 - loss: 0.2344 - val_accuracy: 0.8580 - val_loss: 0.8893
Epoch 78/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 8s 19ms/step - accuracy: 0.9237 - loss: 0.2387 - val_accuracy: 0.8588 - val_loss: 0.8839
Epoch 79/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 7s 17ms/step - accuracy: 0.9253 - loss: 0.2319 - val_accuracy: 0.8573 - val_loss: 0.8722
Epoch 80/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 7s 18ms/step - accuracy: 0.9298 - loss: 0.2224 - val_accuracy: 0.8590 - val_loss: 0.8641
```

RESULTS FOR LSTMs



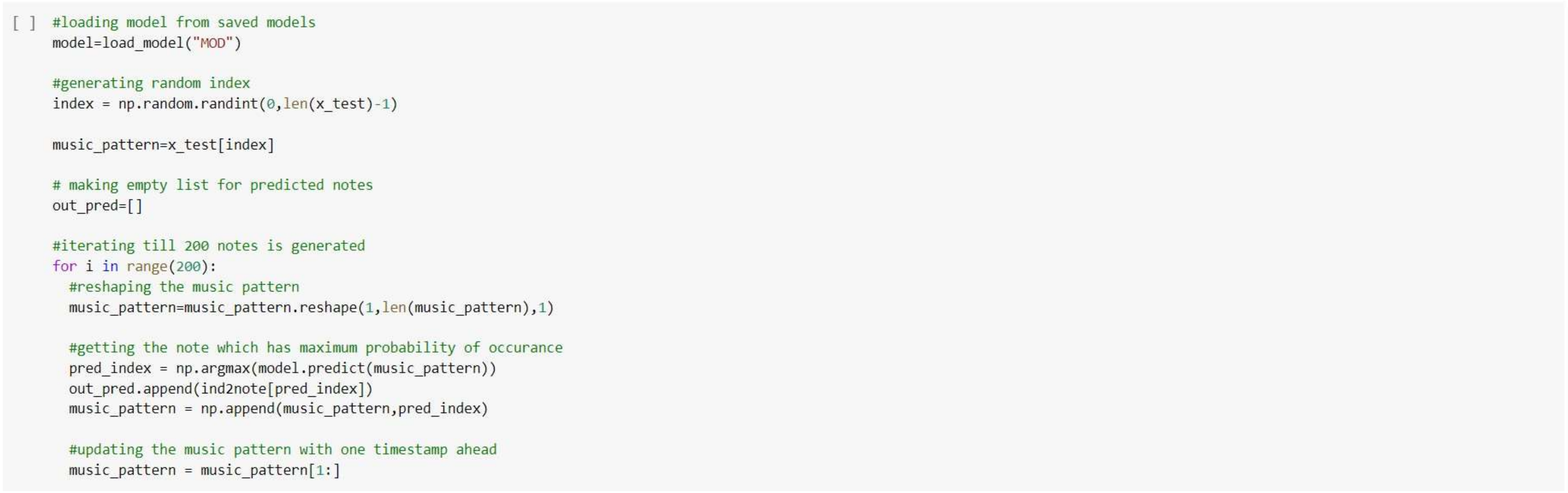
```
405/405 ━━━━━━━━━━━━━━━━━━━━ 8s 19ms/step - accuracy: 0.9768 - loss: 0.0730 - val_accuracy: 0.9012 - val_loss: 0.8103
Epoch 65/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - accuracy: 0.9769 - loss: 0.0734 - val_accuracy: 0.9015 - val_loss: 0.8019
Epoch 66/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - accuracy: 0.9748 - loss: 0.0765 - val_accuracy: 0.9015 - val_loss: 0.8063
Epoch 67/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 8s 19ms/step - accuracy: 0.9760 - loss: 0.0746 - val_accuracy: 0.9025 - val_loss: 0.8171
Epoch 68/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 8s 19ms/step - accuracy: 0.9779 - loss: 0.0696 - val_accuracy: 0.9031 - val_loss: 0.8168
Epoch 69/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 8s 19ms/step - accuracy: 0.9751 - loss: 0.0780 - val_accuracy: 0.9037 - val_loss: 0.8244
Epoch 70/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 8s 20ms/step - accuracy: 0.9797 - loss: 0.0653 - val_accuracy: 0.9023 - val_loss: 0.8162
Epoch 71/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - accuracy: 0.9775 - loss: 0.0699 - val_accuracy: 0.9027 - val_loss: 0.8185
Epoch 72/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - accuracy: 0.9795 - loss: 0.0646 - val_accuracy: 0.9032 - val_loss: 0.8132
Epoch 73/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - accuracy: 0.9793 - loss: 0.0653 - val_accuracy: 0.9038 - val_loss: 0.8212
Epoch 74/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 8s 20ms/step - accuracy: 0.9773 - loss: 0.0671 - val_accuracy: 0.9042 - val_loss: 0.8341
Epoch 75/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - accuracy: 0.9776 - loss: 0.0734 - val_accuracy: 0.9029 - val_loss: 0.8328
Epoch 76/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 8s 20ms/step - accuracy: 0.9793 - loss: 0.0643 - val_accuracy: 0.9024 - val_loss: 0.8438
Epoch 77/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 10s 20ms/step - accuracy: 0.9776 - loss: 0.0694 - val_accuracy: 0.9029 - val_loss: 0.8331
Epoch 78/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - accuracy: 0.9789 - loss: 0.0676 - val_accuracy: 0.9017 - val_loss: 0.8468
Epoch 79/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - accuracy: 0.9781 - loss: 0.0689 - val_accuracy: 0.9021 - val_loss: 0.8352
Epoch 80/80
405/405 ━━━━━━━━━━━━━━━━━━━━ 8s 20ms/step - accuracy: 0.9787 - loss: 0.0673 - val_accuracy: 0.9021 - val_loss: 0.8486
```

STEP 8 Inference Phase

- Using the trained model, the notes will be predicted
- A random integer(**index**) is generated for our testing input array which will be our testing input pattern
- Our array is then re-shaped and the output is predicted
- Using the ‘**np.argmax()**’ function, we get the data of the maximum probability value
- This predicted index is converted to notes using ‘**ind2note**’(index to note) dictionary
- Our next music pattern is one step ahead of the previous pattern
- This process is repeated till we generate **200** notes
- This parameter can be **changed** as per your requirements

CODE SNIPPET

Inference Phase



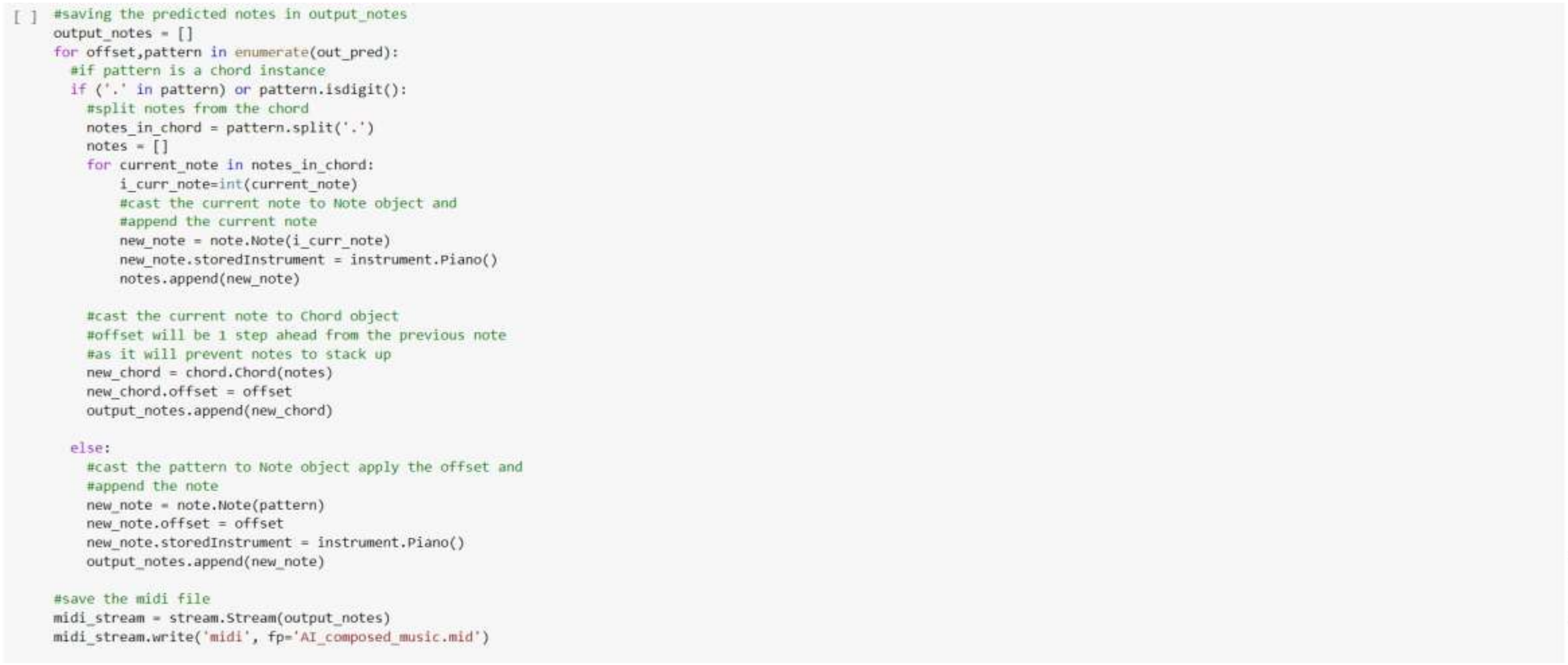
```
[ ] #loading model from saved models
    model=load_model("MOD")

    #generating random index
    index = np.random.randint(0,len(x_test)-1)

    music_pattern=x_test[index]

    # making empty list for predicted notes
    out_pred=[]

    #iterating till 200 notes is generated
    for i in range(200):
      #reshaping the music pattern
      music_pattern=music_pattern.reshape(1,len(music_pattern),1)

      #getting the note which has maximum probability of occurance
      pred_index = np.argmax(model.predict(music_pattern))
      out_pred.append(ind2note[pred_index])
      music_pattern = np.append(music_pattern,pred_index)

      #updating the music pattern with one timestamp ahead
      music_pattern = music_pattern[1:]
```

STEP 9

Saving the File

- The predicted output notes are saved into a MIDI File

CODE SNIPPET



```
[ ] #saving the predicted notes in output_notes
    output_notes = []
    for offset,pattern in enumerate(out_pred):
      #if pattern is a chord instance
      if ('.' in pattern) or pattern.isdigit():
        #split notes from the chord
        notes_in_chord = pattern.split('.')
        notes = []
        for current_note in notes_in_chord:
            i_curr_note=int(current_note)
            #cast the current note to Note object and
            #append the current note
            new_note = note.Note(i_curr_note)
            new_note.storedInstrument = instrument.Piano()
            notes.append(new_note)

        #cast the current note to Chord object
        #offset will be 1 step ahead from the previous note
        #as it will prevent notes to stack up
        new_chord = chord.Chord(notes)
        new_chord.offset = offset
        output_notes.append(new_chord)

      else:
        #cast the pattern to Note object apply the offset and
        #append the note
        new_note = note.Note(pattern)
        new_note.offset = offset
        new_note.storedInstrument = instrument.Piano()
        output_notes.append(new_note)

    #save the midi file
    midi_stream = stream.Stream(output_notes)
    midi_stream.write('midi', fp='AI_composed_music.mid')
```

OUTPUT AUDIO FILE LINK:

https://drive.google.com/file/d/1A0se9zmoqUaVUjwIIiUAGMXGWdEm2uah/view?usp=

sharing

REFERENCE ARTICLE LINK:

https://www.analyticsvidhya.com/blog/2021/12/step-by-step-guide-to-build-image-caption-generator-using- deep-learning/

THANK YOU