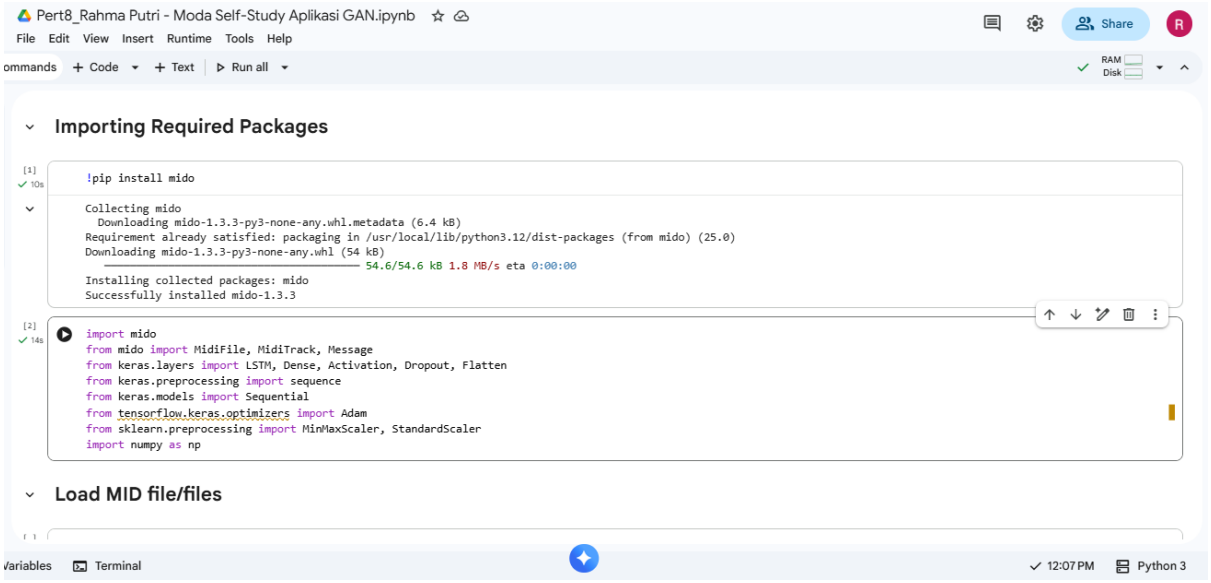

Load MID file/files

```
!unzip /content/chillhopdata.zip -d chillhop
```

```
Archive:  /content/chillhopdata.zip
  inflating: chillhop/1.mid
  inflating: chillhop/10.mid
  inflating: chillhop/11.mid
  inflating: chillhop/12.mid
  inflating: chillhop/13.mid
  inflating: chillhop/14.mid
  inflating: chillhop/15.mid
  inflating: chillhop/16.mid
  inflating: chillhop/17.mid
  inflating: chillhop/18.mid
  inflating: chillhop/19.mid
  inflating: chillhop/2.mid
  inflating: chillhop/20.mid
  inflating: chillhop/3.mid
  inflating: chillhop/4.mid
  inflating: chillhop/5.mid
  inflating: chillhop/6.mid
```

```
import os

notes = []
for song in os.listdir("/content/chillhop"):
  mid = MidiFile('/content/chillhop/' + song)
  for msg in mid:
    if not msg.is_meta and msg.channel == 0 and msg.type =='note_on':
      data = msg.bytes()
      notes.append(data[1])
```

Scale Data

Kode ini menggunakan `MinMaxScaler` dari pustaka `sklearn` untuk menormalkan nilai pitch dari daftar `notes` agar berada dalam rentang [0, 1], dengan terlebih dahulu mengubah daftar `notes` menjadi array NumPy dan mereshape-nya menjadi vektor kolom (-1, 1) sebelum proses transformasi.

```
scaler = MinMaxScaler()
notes = list(scaler.fit_transform(np.array(notes).reshape(-1, 1)))
```

Create Train Data

kode ini membagi data notasi musik (notes) menjadi urutan input-output, di mana input terdiri dari 30 notasi sebelumnya untuk memprediksi notasi berikutnya, dan memisahkan sebagian data untuk digunakan sebagai data uji.

```
notes = [list(note) for note in notes]

X = []
y = []

n_prev=30

for i in range(len(notes) - n_prev):
  X.append(notes[i:i+n_prev])
  y.append(notes[i+n_prev])

X_test = X[-300:]
X = X[:-300]
y = y[:-300]
```

Build LSTM

model ini menggunakan tiga lapisan LSTM untuk memproses urutan notasi musik, dengan dropout di setiap lapisan untuk mencegah overfitting, dan menghasilkan prediksi nilai kontinu (pitch) menggunakan lapisan Dense dengan aktivasi linear.

```
model = Sequential()
model.add(LSTM(256, input_shape=(n_prev, 1), return_sequences=True))
model.add(Dropout(0.6))
model.add(LSTM(128, input_shape=(n_prev, 1), return_sequences=True))
model.add(Dropout(0.6))
model.add(LSTM(64, input_shape=(n_prev, 1), return_sequences=False))
model.add(Dropout(0.6))
model.add(Dense(1))
model.add(Activation('linear'))
model.summary()

optimizer = Adam(learning_rate=0.001)
model.compile(loss='mse', optimizer=optimizer)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential mo
  super().__init__(**kwargs)
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
lstm_6 (LSTM)	(None, 30, 256)	264,192
dropout_6 (Dropout)	(None, 30, 256)	0
lstm_7 (LSTM)	(None, 30, 128)	197,120
dropout_7 (Dropout)	(None, 30, 128)	0
lstm_8 (LSTM)	(None, 64)	49,408
dropout_8 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 1)	65
activation_2 (Activation)	(None, 1)	0

```
Total params: 510,785 (1.95 MB)
Trainable params: 510,785 (1.95 MB)
Non-trainable params: 0 (0.00 B)
```

Training

Melatih model menggunakan data X dan y yang telah dipersiapkan, dengan ukuran batch 16 dan 10 epoch, serta menampilkan progres pelatihan selama proses berlangsung.

```
model.fit(np.array(X), np.array(y), batch_size=16, epochs=10, verbose=1)
```

```
Epoch 1/10
195/195 ━━━━━━━━━━━━━━━━━━━━ 26s 135ms/step - loss: 0.0178
Epoch 2/10
195/195 ━━━━━━━━━━━━━━━━━━━━ 41s 135ms/step - loss: 0.0165
Epoch 3/10
195/195 ━━━━━━━━━━━━━━━━━━━━ 41s 137ms/step - loss: 0.0157
Epoch 4/10
195/195 ━━━━━━━━━━━━━━━━━━━━ 41s 136ms/step - loss: 0.0150
Epoch 5/10
195/195 ━━━━━━━━━━━━━━━━━━━━ 41s 136ms/step - loss: 0.0153
Epoch 6/10
195/195 ━━━━━━━━━━━━━━━━━━━━ 27s 137ms/step - loss: 0.0161
Epoch 7/10
195/195 ━━━━━━━━━━━━━━━━━━━━ 43s 145ms/step - loss: 0.0149
Epoch 8/10
195/195 ━━━━━━━━━━━━━━━━━━━━ 39s 136ms/step - loss: 0.0146
Epoch 9/10
195/195 ━━━━━━━━━━━━━━━━━━━━ 27s 136ms/step - loss: 0.0154
Epoch 10/10
195/195 ━━━━━━━━━━━━━━━━━━━━ 40s 133ms/step - loss: 0.0147
<keras.src.callbacks.history.History at 0x7e617b9b4970>
```

Generating & Saving LSTM Music

Kode ini mengubah hasil prediksi model menjadi notasi MIDI yang dapat dimainkan, dan menyimpannya dalam file LSTM_music.mid.

```
prediction = model.predict(np.array(X_test))
prediction = np.squeeze(prediction)
prediction = np.squeeze(scaler.inverse_transform(prediction.reshape(-1,1)))
prediction = [int(i) for i in prediction]

mid = MidiFile()
track = MidiTrack()
t = 0
for note in prediction:
    # 147 means note_on
    # 67 is velosity
    note = np.asarray([147, note, 67])
    bytes = note.astype(int)
    msg = Message.from_bytes(bytes[0:3])
    t += 1
    msg.time = t
    track.append(msg)
mid.tracks.append(track)
mid.save('LSTM_music.mid')
```

```
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 95ms/step
```

1. Importing Required Packages (Konsep Teori)

Tahap ini adalah menyiapkan semua pustaka penting untuk:

- membaca file MIDI,
- memproses data musik,
- membuat model neural network tipe LSTM,
- melakukan normalisasi data,
- dan manipulasi array.

Tujuannya: agar lingkungan pemrograman siap untuk bekerja dengan data musik dan model machine learning.

2. Load File MIDI

File-file MIDI diekstrak dari arsip ZIP ke dalam folder khusus.

Teori tentang MIDI

MIDI bukan rekaman suara, tetapi instruksi musik, berisi:

- pitch (tinggi nada),
- velocity (keras-lemahnya nada),
- waktu penekanan,
- tipe pesan seperti *note_on* atau *note_off*.

Sehingga sangat cocok untuk diolah dengan machine learning karena formatnya numerik.

3. Ekstraksi Not (Pitch) dari MIDI

Semua file MIDI dibaca satu per satu.
Dari tiap file, hanya pesan yang:

- bukan metadata,
- berada pada channel 0,
- dan bertipe "note_on"

yang diambil.

Teori

Saat sebuah not ditekan, MIDI mengirim pesan *note_on* yang mengandung angka pitch (0–127).

Contoh:

- C4 = 60
- G4 = 67
- A4 = 69

Pitch inilah yang digunakan sebagai dataset utama untuk melatih model.

4. Normalisasi Data (MinMaxScaler)

Nilai pitch 0–127 harus diubah menjadi skala 0–1.

Teori Normalisasi

LSTM jauh lebih stabil jika data masuk berada dalam rentang kecil (0–1). Normalisasi juga mempercepat training dan mencegah bobot model menjadi tidak stabil.

5. Membuat Data Sequence (Input–Output)

Data nada dipecah menjadi format time-series:

- Input = 30 nada sebelumnya
- Output = nada ke-31

Tujuannya adalah melatih model agar bisa memprediksi nada selanjutnya berdasarkan pola musik sebelumnya.

Teori Sequence Learning

Ini sama seperti prediksi kata berikutnya pada kalimat, tapi di sini prediksi not musik berikutnya.

Contoh pola:

Nada: 60 → 62 → 64 → 65 →
Model belajar menghasilkan nada selanjutnya.

6. Membangun Model LSTM

Model terdiri dari:

- 3 lapisan LSTM (256 → 128 → 64 unit)
- Dropout di tiap lapisan (mencegah overfitting)
- Lapisan Dense (output satu nilai pitch)
- Aktivasi linear (karena prediksi bersifat numerik)

Teori LSTM

LSTM (Long Short-Term Memory):

- dirancang untuk memahami urutan,
- mampu mengingat pola jangka panjang,
- sangat cocok untuk musik karena musik adalah time-series.

Setiap lapisan LSTM menangkap pola semakin kompleks:

- lapisan pertama → progresi nada panjang,
- lapisan tengah → pola melodi,
- lapisan akhir → detail pergerakan nada.

7. Training

Model dilatih menggunakan:

- data input (urutan 30 nada),
- target output (nada ke-31),
- batch kecil,
- beberapa epoch.

Teori

Training berarti:

- model menyesuaikan bobotnya,
- mengurangi error (loss),
- belajar pola progresi melodi dari dataset MIDI.

Semakin kecil loss, semakin baik model mempelajari struktur musik.

8. Generating & Saving Music

Setelah training selesai:

- Model melakukan prediksi pada data uji.
- Hasil prediksi (0–1) dikembalikan ke skala pitch (0–127).
- Prediksi diubah menjadi pesan MIDI *note_on*.
- Semua pesan digabung menjadi satu file musik baru (LSTM_music.mid).

Teori

Inilah proses *music generation*:

- Model menciptakan rangkaian nada baru berdasarkan pola yang telah ia pelajari.
- Nada prediksi disimpan ke bentuk MIDI supaya dapat dimainkan.