

## LAB Assignment No 7

### Recurrent Neural Network (RNN)

#### LAB Task 1:

#### Next Word Prediction using RNN

**Objective:** Learn how RNNs can predict the next word in a sentence.

**Dataset:** Any small text corpus — e.g., *Shakespeare.txt* or *Wikipedia sample*.

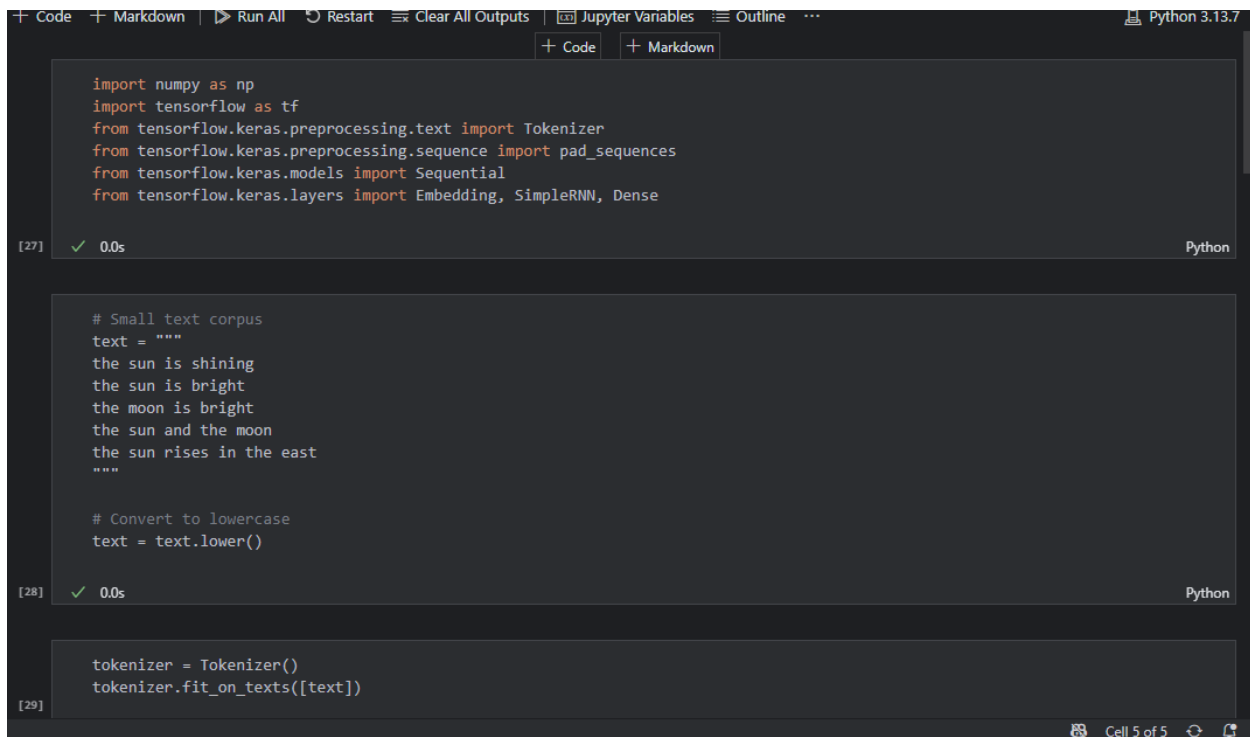
#### Tasks:

1. Load and clean the text data.
2. Tokenize and convert text into sequences.
3. Build a simple **RNN model** using `keras.layers.SimpleRNN`.
4. Train it to predict the next word given previous 3–5 words.
5. Test by entering a custom text prompt and predict the next word.

#### Output:

Model predicts probable next word, e.g.,

**Input:** “The sun is” → **Output:** “shining”



```
+ Code + Markdown | ▶ Run All ⏮ Restart ⏭ Clear All Outputs | 📄 Jupyter Variables 📄 Outline ... Python 3.13.7
+ Code + Markdown

import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense

[27] ✓ 0.0s Python

# Small text corpus
text = """
the sun is shining
the sun is bright
the moon is bright
the sun and the moon
the sun rises in the east
"""

# Convert to lowercase
text = text.lower()

[28] ✓ 0.0s Python

tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])

[29]
```

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])

word_index = tokenizer.word_index
total_words = len(word_index) + 1

# Convert text to sequences
sequences = []
for line in text.split('\n'):
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(3, len(token_list)):
        sequences.append(token_list[i-3:i+1])

sequences = np.array(sequences)

# Split input and output
X = sequences[:, :-1]
y = sequences[:, -1]

# Vocabulary size
print("Total words:", total_words)
```

✓ 0.0s Python

Total words: 11

```
Code + Markdown | ▶ Run All | ⌂ Restart | 🗑 Clear All Outputs | 📄 Jupyter Variables | 📖 Outline | ⋮
model = Sequential([
    Embedding(total_words, 10, input_length=3),
    SimpleRNN(50),
    Dense(total_words, activation='softmax')
])

model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)

model.summary()
```

✓ 0.0s Python

Model: "sequential\_4"

Layer (type)	Output Shape	Param #
embedding_4 (Embedding)	?	0 (unbuilt)
simple_rnn_4 (SimpleRNN)	?	0 (unbuilt)
dense_4 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)

Trainable params: 0 (0.00 B)

Non-trainable params: 0 (0.00 B)

Cell 5 of 5

```
def predict_next_word(model, tokenizer, text_input):
    sequence = tokenizer.texts_to_sequences([text_input])[0]
    sequence = pad_sequences([sequence], maxlen=3, padding='pre')
    predicted = np.argmax(model.predict(sequence), axis=-1)

    for word, index in tokenizer.word_index.items():
        if index == predicted:
            return word

# Test input
input_text = "the sun is"
output_word = predict_next_word(model, tokenizer, input_text)

print("Input:", input_text)
print("Predicted Next Word:", output_word)
```

✓ 1.6s Python

1/1 ————— 1s 1s/step

Input: the sun is

Predicted Next Word: moon

## LAB Task 2:

### Stock Price Prediction using RNN

**Objective:** Predict future stock prices using time series data.

**Dataset:** Use *Google Stock Price* dataset (from Kaggle or Yahoo Finance).

#### Tasks:

1. Import dataset and normalize values.
2. Prepare time-step sequences (e.g., 60 previous days → next day price).
3. Build and train an **RNN model** using SimpleRNN layers.
4. Evaluate predictions vs actual prices (plot graph).

#### Output:

Line graph showing predicted vs real stock price trend.

```
time_step = 5 # 60 ki jagah 5 (small dataset ke liye)
```

```
X = []  
y = []
```

```
for i in range(time_step, len(scaled_data)):  
    X.append(scaled_data[i-time_step:i, 0])  
    y.append(scaled_data[i, 0])
```

```
X = np.array(X)  
y = np.array(y)
```

```
X = np.reshape(X, (X.shape[0], X.shape[1], 1))
```

```
print(X.shape, y.shape)
```

✓ 0.0s

Python

(55, 5, 1) (55,)

```
model = Sequential()  
model.add(SimpleRNN(50, return_sequences=True, input_shape=(X.shape[1], 1)))  
model.add(SimpleRNN(50))  
model.add(Dense(1))  
  
model.compile(optimizer='adam', loss='mean_squared_error')
```

✓ 0.1s

Python

c:\Users\h\AppData\Local\Programs\Python\Python313\lib\site-packages\keras\src\layers\rnn\rnn.py:199: UserWarning: Do not pass an "input\_shape"/"input\_dim" argument to super().\_\_init\_\_(\*\*kwargs)

Code + Markdown | Run All | Restart | Clear All Outputs | Jupyter Variables | Outline | Python 3.13.7

```
super().__init__(**kwargs)
```

```
model.fit(X, y)
```

[71] ✓ 5.0s

Python

2/2 5s 69ms/step - loss: 0.5932

<keras.src.callbacks.history.History at 0x28f9a955a70>

```
predicted_prices = model.predict(X)  
  
predicted_prices = scaler.inverse_transform(predicted_prices)  
real_prices = scaler.inverse_transform(y.reshape(-1, 1))
```

[72] ✓ 1.0s

Python

WARNING:tensorflow:6 out of the last 7 calls to <function TensorFlowTrainer.make\_predict\_function.<locals>.one\_step\_on\_data\_distributed at 0x0000028FA23F7D80> trigger

2/2 1s 396ms/step

```
plt.figure(figsize=(10,5))  
plt.plot(real_prices, label='Real Stock Price')  
plt.plot(predicted_prices, label='Predicted Stock Price')  
plt.xlabel('Time')  
plt.ylabel('Stock Price')  
plt.title('Stock Price Prediction using RNN')  
plt.legend()  
plt.show()
```

[73] ✓ 0.4s

Python



### LAB Task 3:

#### Sentiment Analysis using RNN

**Objective:** Classify movie reviews as positive or negative using RNN.

**Dataset:** *IMDb Movie Reviews* dataset (available in Keras).

#### Tasks:

1. Load dataset and preprocess text (tokenize and pad sequences).
2. Build RNN with Embedding + SimpleRNN layers.
3. Train for binary classification (positive/negative).
4. Evaluate accuracy on test data.

#### Output:

Accuracy score (e.g., 85%) and prediction for custom input text.

```
import numpy as np
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
```

✓ 0.0s

Python

```
# Load IMdb dataset
vocab_size = 10000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=vocab_size)
```

✓ 10.5s

Python

```
max_len = 200

X_train = pad_sequences(X_train, maxlen=max_len)
X_test = pad_sequences(X_test, maxlen=max_len)
```

✓ 1.7s

Python

```
model = Sequential()
model.add(Embedding(vocab_size, 32, input_length=max_len))
model.add(SimpleRNN(50))
model.add(Dense(1, activation='sigmoid'))

model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)
```

Spaces: 4 CRLF Cell 7 of 7

```
model = Sequential()
model.add(Embedding(vocab_size, 32, input_length=max_len))
model.add(SimpleRNN(50))
model.add(Dense(1, activation='sigmoid'))

model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

model.summary()
```

✓ 0.1s

Python

Model: "sequential\_12"

Layer (type)	Output Shape	Param #
embedding_8 (Embedding)	?	0 (unbuilt)
simple_rnn_16 (SimpleRNN)	?	0 (unbuilt)
dense_12 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)

Trainable params: 0 (0.00 B)

Non-trainable params: 0 (0.00 B)

```
Trainable params: 0 (0.00 B)

Non-trainable params: 0 (0.00 B)

model.fit(X_train, y_train)

460/782 — 39.6s — 23s 74ms/step - accuracy: 0.5449 - loss: 0.6867

loss, accuracy = model.evaluate(X_test, y_test)
print("Test Accuracy:", accuracy)
# Load word index
word_index = imdb.get_word_index()

def encode_review(text):
    words = text.lower().split()
    encoded = [word_index.get(word, 2) for word in words]
    padded = pad_sequences([encoded], maxlen=max_len)
    return padded

782/782 — 15s 18ms/step - accuracy: 0.8096 - loss: 0.4510
Test Accuracy: 0.809599956130981
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb\_word\_index.json
1641221/1641221 — 0s 0us/step

# Custom input
review = "the movie was amazing and enjoyable"
encoded_review = encode_review(review)

prediction = model.predict(encoded_review)

if prediction > 0.5:
    print("Review:", review)
    print("Sentiment: Positive")
else:
    print("Review:", review)
    print("Sentiment: Negative")

1/1 — 0.6s — 0s 452ms/step
Review: the movie was amazing and enjoyable
Sentiment: Negative
```

## LAB Task 4:

### Weather Forecasting using RNN

**Objective:** Predict future temperature based on previous days' readings.

**Dataset:** *Daily temperature dataset* (e.g., "Jena Climate Dataset" from TensorFlow).

#### Tasks:

1. Load and visualize temperature over time.
2. Prepare input-output sequences for time series prediction.
3. Build an RNN to predict next day's temperature.
4. Plot actual vs predicted temperature.

## Output:

Graph showing predicted vs actual temperature trends.

```
import pandas as pd

data = pd.DataFrame({
    "T (degC)": [-8.0, -6.9, -5.4, -3.8, -2.1, -1.0, 0.5, 1.2, 2.0, 3.1,
                4.0, 5.2, 6.0, 6.8, 7.5, 8.1, 7.8, 6.9, 5.5, 4.2,
                3.0, 2.1, 1.0, 0.2, -0.8, -1.9, -2.5, -3.0, -4.2, -5.0]
})

temperature = data['T (degC)']
data.head()
```

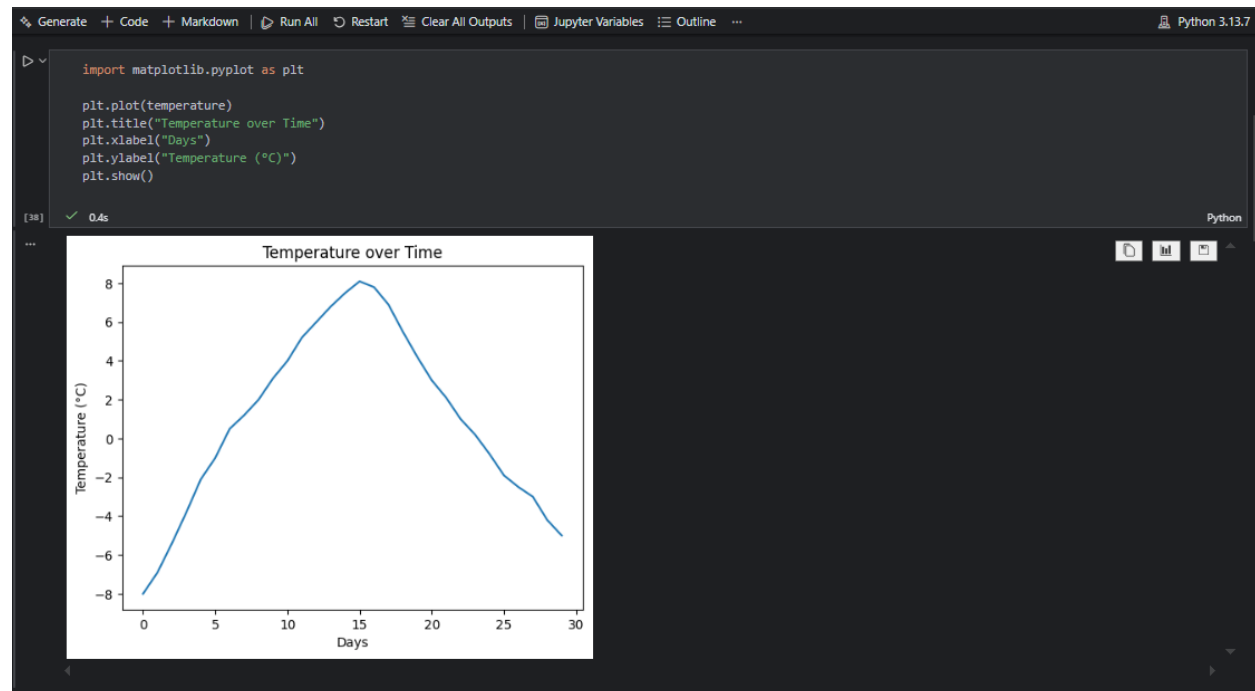
0.0s Python

T (degC)	
0	-8.0
1	-6.9
2	-5.4
3	-3.8
4	-2.1

```
import matplotlib.pyplot as plt

plt.plot(temperature)
plt.title("Temperature over Time")
plt.xlabel("Days")
plt.ylabel("Temperature (°C)")
plt.show()
```

0.4s Python





```
Generate + Code + Markdown | Run All | Restart | Clear All Outputs | Jupyter Variables | Outline | Python 3.13.7

X, y = create_sequences(temp_scaled, 5)

[40] ✓ 0.0s Python

split = int(0.8 * len(X))

X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

[41] ✓ 0.0s Python

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

model = Sequential()
model.add(SimpleRNN(20, activation='tanh', input_shape=(5,1)))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mse')

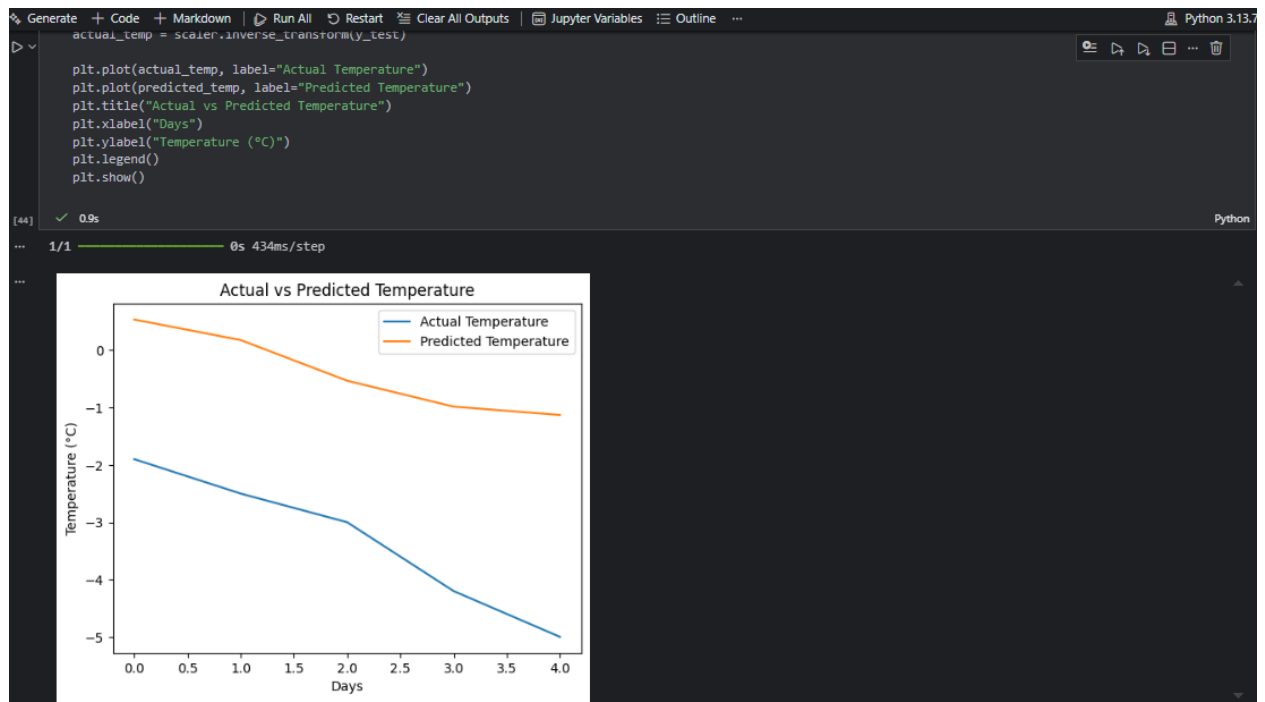
[42] ✓ 0.0s Python

... c:\Users\h\AppData\Local\Programs\Python\Python313\Lib\site-packages\keras\src\layers\rnn\rnn.py:199: UserWarning: Do not pass an `input_shape`/`input_dim` argument to
super().__init__(**kwargs)

model.fit(X_train, y_train, epochs=2, batch_size=2)

[43] ✓ 3.6s Python

... Epoch 1/2
10/10 3s 9ms/step - loss: 0.6637
```



## LAB Task 5:

### Music Note Generation using RNN

**Objective:** Generate new music sequences using RNN.

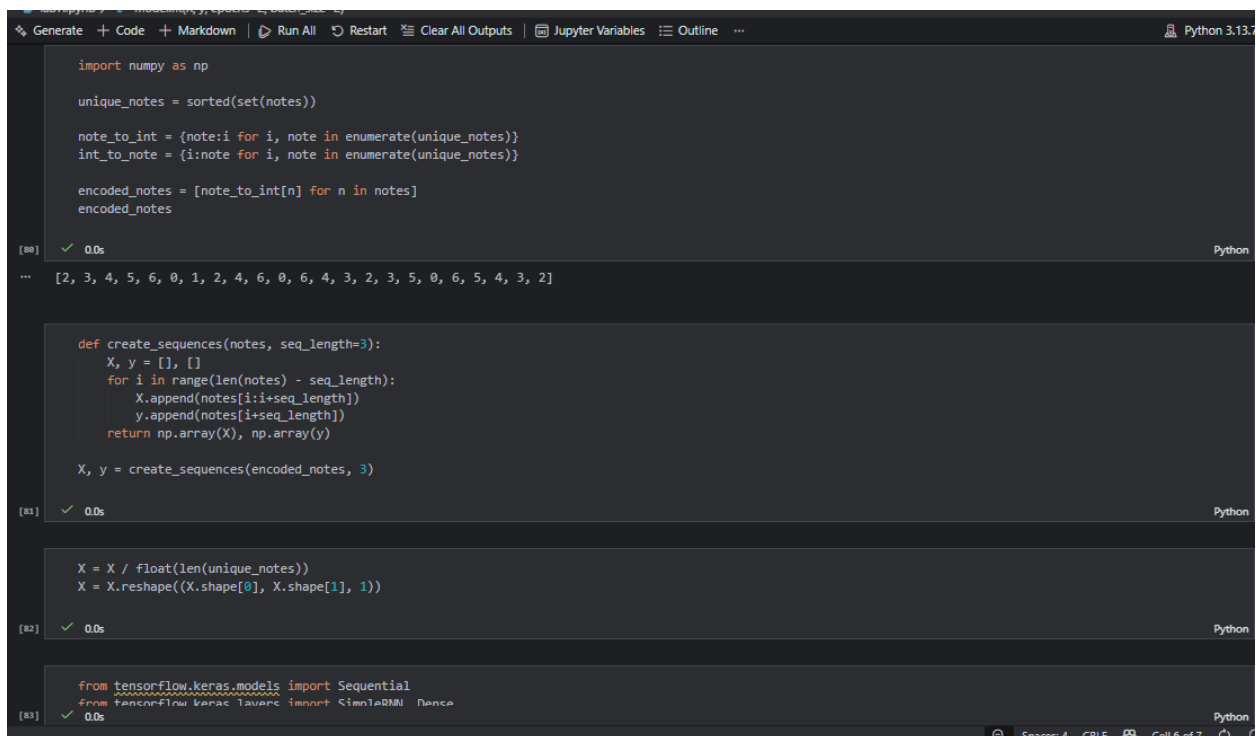
**Dataset:** *MIDI music dataset* (short sequences or melodies).

#### Tasks:

1. Convert MIDI data into integer-encoded notes.
2. Train an RNN on note sequences (input: previous notes  $\rightarrow$  output: next note).
3. Generate a new sequence using the trained model.

#### Output:

A sequence of generated notes that can be converted to a playable MIDI file



```
import numpy as np

unique_notes = sorted(set(notes))

note_to_int = {note:i for i, note in enumerate(unique_notes)}
int_to_note = {i:note for i, note in enumerate(unique_notes)}

encoded_notes = [note_to_int[n] for n in notes]
encoded_notes

[2, 3, 4, 5, 6, 0, 1, 2, 4, 6, 0, 6, 4, 3, 2, 3, 5, 0, 6, 5, 4, 3, 2]

def create_sequences(notes, seq_length=3):
    X, y = [], []
    for i in range(len(notes) - seq_length):
        X.append(notes[i:i+seq_length])
        y.append(notes[i+seq_length])
    return np.array(X), np.array(y)

X, y = create_sequences(encoded_notes, 3)

X = X / float(len(unique_notes))
X = X.reshape((X.shape[0], X.shape[1], 1))

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense
```

```
Generate + Code + Markdown Run All Restart Clear All Outputs Jupyter Variables Outline ... Python 3.13.7

optimizer='adam')

[83] ✓ 0.0s Python

model.fit(X, y, epochs=2, batch_size=2)

[84] ✓ 2.8s Python

... Epoch 1/2
10/10 3s 7ms/step - loss: 1.9417
Epoch 2/2
10/10 0s 6ms/step - loss: 1.8707
... <keras.src.callbacks.history.History at 0x1ea51a3eb10>

start_index = np.random.randint(0, len(X)-1)
pattern = X[start_index]

generated_notes = []

for i in range(10): # generate 10 notes
    prediction = model.predict(pattern.reshape(1,3,1), verbose=0)
    index = np.argmax(prediction)
    generated_notes.append(int_to_note[index])

    next_input = index / float(len(unique_notes))
    next_input = np.array([[next_input]]) # shape (1,1)
    pattern = np.vstack((pattern[1:], next_input))

generated_notes

[85] ✓ 2.7s Python

... ['G', 'C', 'A', 'G', 'E', 'A', 'G', 'C', 'A', 'G']
```