```
n_notes = len(all_notes)
print("Number of notes parsed:", n_notes)
```

✓ 0.0s

```
Number of notes parsed: 15435
```

Calculates and prints the total number of notes that have been parsed and stored in the all_notes DataFrame

```
#create a dataset from the parsed notes
key_order = ["pitch", "step", "duration"]
train_notes = np.stack([all_notes[key] for key in key_order], axis = 1)
```

✓ 0.0s

Combines all the arrays along a new axis

```
notes_ds = tf.data.Dataset.from_tensor_slices(train_notes)
notes_ds.element_spec
```

✓ 0.1s

Converts the NumPy array train_notes into a tensorflow object
Spec tells us that the dataset has been correctly formatted

```
TensorSpec(shape=(3,), dtype=tf.float64, name=None)
```

```python
#we are training the model on batches of sequences of notes
def create_sequences(
    dataset: tf.data.Dataset,
    seq_length: int,
    vocab_size = 128,
) -> tf.data.Dataset:

  seq_length = seq_length + 1

  # Take 1 extra for the labels
  windows = dataset.window(seq_length, shift=1, stride=1,
                           drop_remainder=True)
  # `flat_map` flattens the" dataset of datasets" into a dataset of tensors
  flatten = lambda x: x.batch(seq_length, drop_remainder=True)
  sequences = windows.flat_map(flatten)

  # Normalize note pitch
  def scale_pitch(x):
    x = x/[vocab_size,1.0,1.0]
    return x

  # Split the labels
  def split_labels(sequences):
    inputs = sequences[:-1]
    labels_dense = sequences[-1]
    labels = {key:labels_dense[i] for i,key in enumerate(key_order)}

    return scale_pitch(inputs), labels

  return sequences.map(split_labels, num_parallel_calls=tf.data.AUTOTUNE)
```

dataset is the input dataset, or notes_ds from earlier.
seq_length specifies how many notes to include in each sequence.
vocab_size specifies the total # of unique possible note pitches.

Adds 1, since each input sequence will have a corresponding label (next note in the sequence)

Creates sequences of length seq_length from the dataset. Creates. overlapping windows (sub-sequnces). For ex, first window starts at note 0, the second starts at note 1, the third at note 2...

Ensures the result is a single flattened dataset of note sequences, rather than datasets within datasets

Standardizes the pitch vales

Sequences are split into inputs (all but last note) and labels (target label for prediction)

At the end, we hope to convert a dataset of individual notes into a sequence of notes that we can use to train our model.

```python
#setting the sequence length
#vocab size 128 represents all the pitches supported by pretty_midi
seq_length = 25
vocab_size = 128
seq_ds = create_sequences(notes_ds, seq_length, vocab_size)
seq_ds.element_spec
```

✓ 0.1s

Each sequence fed into the model will be 25 notes long. The possible pitch values (MIDI numbers) range from 0-127.

create_sequences is a function that processes our 25 notes to make sequences. element_spec, like earlier, displays the data type to ensure it's formatted as expected.

```
(TensorSpec(shape=(25, 3), dtype=tf.float64, name=None),
 {'pitch': TensorSpec(shape=(), dtype=tf.float64, name=None),
  'step': TensorSpec(shape=(), dtype=tf.float64, name=None),
  'duration': TensorSpec(shape=(), dtype=tf.float64, name=None)})
```

Notice (25, 3): this is 25 notes with 3 attributes each. Each attribute is type float64, meaning they are 64-bit floats.

```python
#shape of dataset is (100,1) so the model will take 100 notes as input and learn the predict
#the following note as output
for seq, target in seq_ds.take(1):
  print('sequence shape:', seq.shape)
  print('sequence elements (first 10):', seq[0: 10])
  print()
  print('target:', target)
```

✓ 0.1s

Loop extracts a single sequence of length 25 and will print the first 10 notes in the sequence. Each containing values for pitch, step, and duration.

```
sequence shape: (25, 3)
sequence elements (first 10): tf.Tensor(
[[0.6015625  0.         0.09244792]
 [0.3828125  0.00390625 0.40234375]
 [0.5703125  0.109375   0.06510417]
 [0.53125    0.09895833 0.06119792]
 [0.5703125  0.10807292 0.05989583]
 [0.4765625  0.06640625 0.05338542]
 [0.6015625  0.01302083 0.0703125 ]
 [0.5703125  0.11979167 0.07682292]
 [0.3984375  0.109375   0.43619792]
 [0.609375   0.00130208 0.09505208]], shape=(10, 3), dtype=float64)
```

```python
#batch the examples, configure the dataset
batch_size = 64
buffer_size = n_notes - seq_length  #this is the number of items in the dataset
train_ds = (seq_ds
            .shuffle(buffer_size)
            .batch(batch_size, drop_remainder=True)
            .cache()
            .prefetch(tf.data.experimental.AUTOTUNE))
```

✓ 0.0s

We are configuring our dataset to optimize it for model training. First, we set the batch size to 64, so the model will process 64 sequences at a time. Buffer_size essentially tells us how much data is available for random shuffling before creating batches.

.shuffle randomizes our dataset. .batch groups them into 64 sequence groups. drop_remainder gets rid of any remainder after batching. We then cache the dataset into memory. Lastly, prefetch preps the next batch while the current one is being processed!

```python
train_ds.element_spec
```

✓ 0.0s

```
(TensorSpec(shape=(64, 25, 3), dtype=tf.float64, name=None),
 {'pitch': TensorSpec(shape=(64,), dtype=tf.float64, name=None),
  'step': TensorSpec(shape=(64,), dtype=tf.float64, name=None),
  'duration': TensorSpec(shape=(64,), dtype=tf.float64, name=None)})
```

Again, we will examine the structure of our data. (64, 25, 3) -> 64 batches, 25 sequences, 3 attributes per sequence. All values are 64-bit floats.

```python
#the model will have three outputs, for step, pitch, duration
#custom loss funciton based on eman squared error to encourage the model to output pos. values
def mse_with_positive_pressure(y_true: tf.Tensor, y_pred: tf.Tensor):
    mse = (y_true - y_pred) ** 2
    positive_pressure = 10 * tf.maximum(-y_pred, 0.0)
    return tf.reduce_mean(mse + positive_pressure)
```

This line ensures that any negative predicted y-vals have a “penalty,” which is then scaled up (x10) to encourage our model to avoid negative values

```python
input_shape = (seq_length, 3)
learning_rate = 0.005

inputs = tf.keras.Input(input_shape)
x = tf.keras.layers.LSTM(128)(inputs)

outputs = {
  'pitch': tf.keras.layers.Dense(128, name='pitch')(x),
  'step': tf.keras.layers.Dense(1, name='step')(x),
  'duration': tf.keras.layers.Dense(1, name='duration')(x),
}

model = tf.keras.Model(inputs, outputs)

loss = {
      'pitch': tf.keras.losses.SparseCategoricalCrossentropy(
          from_logits=True),
      'step': mse_with_positive_pressure,
      'duration': mse_with_positive_pressure,
}

optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)

model.compile(loss=loss, optimizer=optimizer)

model.summary()
```

✓ 0.3s

The model expects sequences of length 25, with 3 features each. The learning rate determines the magnitude of updates to the model's weights.

Defines the input of the model with shape (None, 25, 3) as the batch size can vary depending on how many samples are processed at once. An LSTM is added, which outputs shape (None, 128): Varying batch sizes and 128 is the number of units in the LSTM layer. Basically each of the 128 values is a learned feature.

Dense layers (layers that connect every neuron in the previous later to every neuron in the current layer) are created, with 128 units each. Pitch has 128 units since pitch is represented as 1 of 128 MIDI pitch values, but step and duration are 1 unit since it's a regression task (the output is a real number, not a class label like pitch)

The loss for pitch (classification task) uses SparseCategoricalCrossentropy. For step and duration (regression task), out function from before is used.

An Adam optimizer is used, and the model is compiled with the specified loss functions now.

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 25, 3)	0	-
lstm (LSTM)	(None, 128)	67,584	input_layer[0][0]
duration (Dense)	(None, 1)	129	lstm[0][0]
pitch (Dense)	(None, 128)	16,512	lstm[0][0]
step (Dense)	(None, 1)	129	lstm[0][0]

Total params: 84,354 (329.51 KB)

Trainable params: 84,354 (329.51 KB)

Non-trainable params: 0 (0.00 B)

model.summary shows us a summary of the model's structure. As mentioned, the input is (None, 25, 3), with the LSTM creating the output of shape (None, 128). Param # 67,584 includes the weights and biases in each of the LSTM's gates.

For example, an LSTM with 128 units, 3 input features, and 4 gates has 4 * ((128 * (128 + 3)) + 128) params.

Duration and step have output shapes of (None, 1) while pitch has (None, 128).

The total/trainable # params is the total weights and biases in the model.

```
#we can see the pitch loss is significantly greater than step and duration
#loss is the total loss computed by summing all other losses
losses = model.evaluate(train_ds, return_dict=True)
losses
```

✓ 4.5s

```
240/240 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - duration_loss: 0.1634 - loss: 5.0313
2025-01-07 13:12:14.420351: W tensorflow/core/framework/local_rendezvous.cc:404]
        [[{{node IteratorGetNext}}]]
/Users/akalgi/work/default/lib/python3.12/site-packages/keras/src/trainers/epoch
  self._interrupted_warning()
```

```
{'duration_loss': 0.1556001901626587,
 'loss': 5.008670330047607,
 'pitch_loss': 4.8336052894592285,
 'step_loss': 0.019464299082756042}
```

We are evaluating the model's performance based on train_ds, the dataset of sequencs of notes that the model is trying to predict.

return_dict ensures that our total loss for pitch, step, duration is returned as a dictionary

240/240 -> all 240 batches in the dataset have been processed

From the output, we can assume the model is finding it more challenging to predict pitch values accurately.

```
#use the loss_weights argument to compile
model.compile(
    loss=loss,
    loss_weights={
        'pitch': 0.05,
        'step': 1.0,
        'duration':1.0,
    },
    optimizer=optimizer,
)
```

✓ 0.0s

We are now adjusting the individual losses. By setting pitch very low, we are giving more importance to to step and duration predictions, both of which have 1.0 (standard importance)

```
#loss becomes the weighted sum of the individual losses
model.evaluate(train_ds, return_dict = True)
```

✓ 3.3s

```
240/240 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - duration_loss: 0.1634 - loss: 0.420
2025-01-07 13:50:51.069802: W tensorflow/core/framework/local_rendezvous.cc:404]
        [[{{node IteratorGetNext}}]]
```

After adjusting the loss weight, we evaluate the model again, which shows a decrease in total loss, however pitch is still challenging for the model to predict. The model is also slightly more efficient as time taken per step is shorter (from 5.7s/step to 3.1s/step).

```python
#now train the model
callbacks = [
    tf.keras.callbacks.ModelCheckpoint(
        filepath='./training_checkpoints/ckpt_{epoch}.weights.h5',
        save_weights_only=True),
    tf.keras.callbacks.EarlyStopping(
        monitor='loss',
        patience=5,
        verbose=1,
        restore_best_weights=True),
]
```

✓ 0.0s

Callbacks are a list of functions that will be used during training. ModelCheckpoint: saves the model's weights during training. The following lines ensure that the weights are saved, not the entiremodel, which saves us space.

EarlyStopping stops the training early if the model's performance stops improving. Specifically, it's monitoring the training loss, and will stop if it doesn't improve after 5 epochs. If early stopping does hapen, then the best weight during the model's performance will be restored.

```python
epochs = 5

history = model.fit(
    train_ds,
    epochs=epochs,
    callbacks=callbacks,
)
```

✓ 19.6s

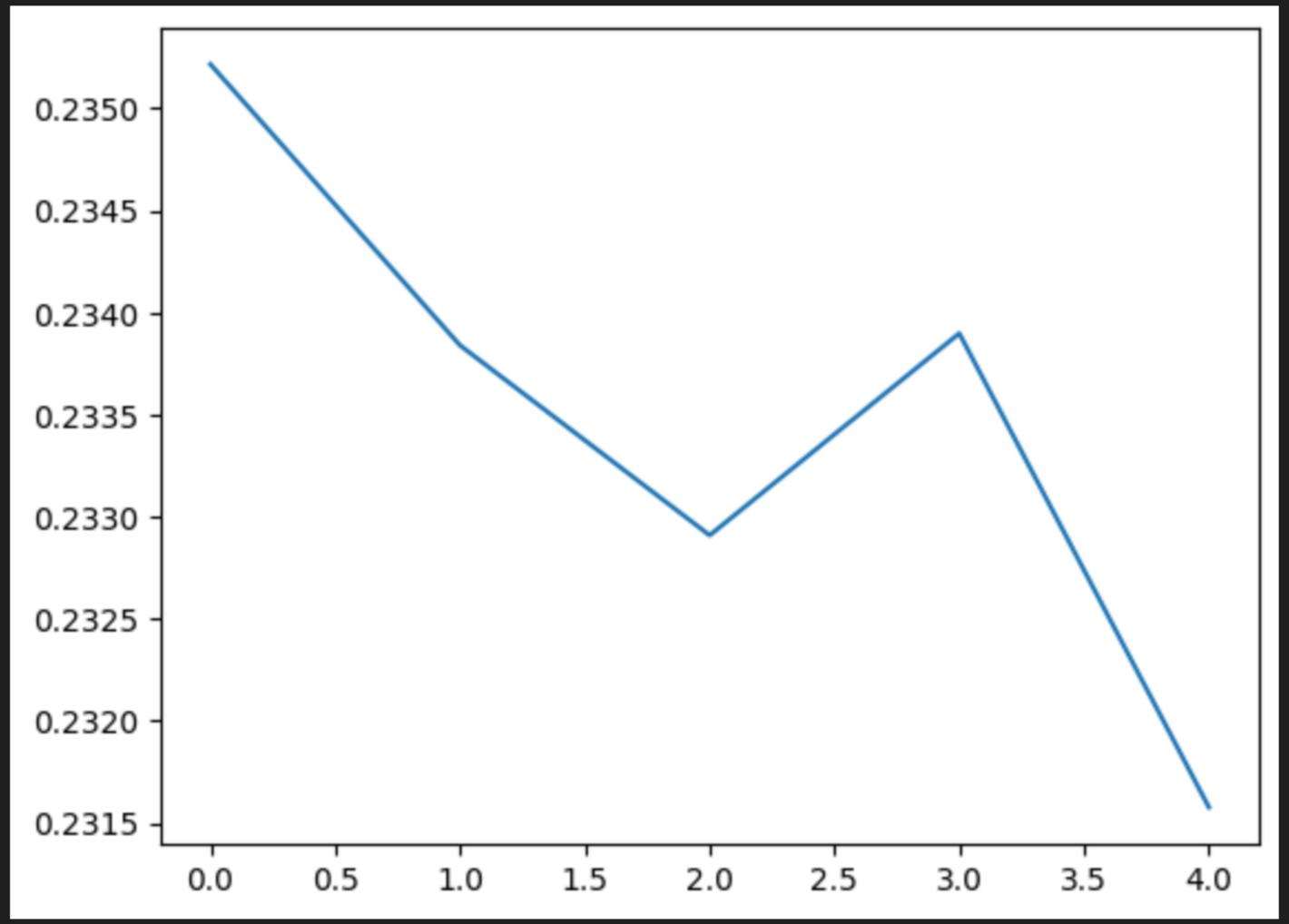
Epochs = one full pass through the entire training dataset.
The following lines just confirm the training dataset, number of epochs, and callback functions.

Each epoch has 240/240 batches processed. After each epoch, the time taken per step, as well as the loss for all three attributes is displayed.

```
Epoch 1/5
240/240 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - duration_loss: 0.0438 - loss: 0.2355 - pitch_loss: 3.6095 - step_loss: 0.0113
Epoch 2/5
  9/240 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - duration_loss: 0.1010 - loss: 0.2922 - pitch_loss: 3.5865 - step_loss: 0.0119
2025-01-07 14:08:17.911003: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
        [[{{node IteratorGetNext}}]]
240/240 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - duration_loss: 0.0430 - loss: 0.2338 - pitch_loss: 3.5893 - step_loss: 0.0112
Epoch 3/5
  8/240 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - duration_loss: 0.1069 - loss: 0.2975 - pitch_loss: 3.5709 - step_loss: 0.0121
2025-01-07 14:08:21.582871: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
        [[{{node IteratorGetNext}}]]
240/240 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - duration_loss: 0.0429 - loss: 0.2330 - pitch_loss: 3.5798 - step_loss: 0.0111
Epoch 4/5
  8/240 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - duration_loss: 0.1074 - loss: 0.2984 - pitch_loss: 3.5781 - step_loss: 0.0121
2025-01-07 14:08:25.573404: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
        [[{{node IteratorGetNext}}]]
240/240 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - duration_loss: 0.0429 - loss: 0.2349 - pitch_loss: 3.5941 - step_loss: 0.0123
Epoch 5/5
  4/240 ━━━━━━━━━━━━━━━━━━━━ 5s 22ms/step - duration_loss: 0.1512 - loss: 0.3438 - pitch_loss: 3.5736 - step_loss: 0.0139
2025-01-07 14:08:29.337215: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
        [[{{node IteratorGetNext}}]]
240/240 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - duration_loss: 0.0429 - loss: 0.2321 - pitch_loss: 3.5597 - step_loss: 0.0112
Restoring model weights from the end of the best epoch: 5.
2025-01-07 14:08:33.261056: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
        [[{{node IteratorGetNext}}]]
```

```
plt.plot(history.epoch, history.history['loss'], label='total loss')
plt.show()
```

✓ 0.1s



We are plotting the training loss over the epochs.
History.epoch is the list of epochs the model has completed.
history.history is a dictionary containing the total loss over time.

```python
#now use the model to generate notes
#first need to provide a starting sequence of notes
def predict_next_note(
    notes: np.ndarray,
    model: tf.keras.Model,
    temperature: float = 1.0) -> tuple[int, float, float]:

  assert temperature > 0

  inputs = tf.expand_dims(notes, 0)

  predictions = model.predict(inputs)
  pitch_logits = predictions['pitch']
  step = predictions['step']
  duration = predictions['duration']

  pitch_logits /= temperature
  pitch = tf.random.categorical(pitch_logits, num_samples=1)
  pitch = tf.squeeze(pitch, axis=-1)
  duration = tf.squeeze(duration, axis=-1)
  step = tf.squeeze(step, axis=-1)

  # `step` and `duration` values should be non-negative
  step = tf.maximum(0, step)
  duration = tf.maximum(0, duration)

  return int(pitch), float(step), float(duration)
```

This is a numpy array containing the sequence of notes that the model uses to predict the next note. The array's shape is typically (sequence_length, 3).

The model is the trained Keras model used to predict the next pitch, step, and duration values.

Temperature lets us control the creativity of the pitch prediction. >1 is more random, while <1 is more conservative.

Incase the function is called with temperatures of 0 or negative, this assertation is included.

This adds a batch dimension to the notes array, since the model is expecting a batch of sequences as input.

Based off the inputs, our model returns a dictionary of predicted values of pitch, step, and duration.

Logits are raw outputs from the model, with higher logits meaning the model is more confident its predicted pitch is correct. Here we temperature scale, which adjusts them to be more or less confident.

Here we sample just one pitch. Squeeze then removes any extra unecessary dimensions. For example, before we might have had shape (1,25,3), showing a batch of 1 sequence with 25 time steps and 3 features. Afterwards, however, we might have (25,3).

Ensures any negative values for step and duration are replaced with 0

This function returns the predicted pitch value as an integer, and step and duration as a float.

```python
#now generate some notes
temperature = 1.5
num_predictions = 120

sample_notes = np.stack([raw_notes[key] for key in key_order], axis=1)

#pitch is normalized
input_notes = (
    sample_notes[:seq_length] / np.array([vocab_size, 1, 1]))

generated_notes = []
prev_start = 0
for _ in range(num_predictions):
  pitch, step, duration = predict_next_note(input_notes, model, temperature)
  start = prev_start + step
  end = start + duration
  input_note = (pitch, step, duration)
  generated_notes.append((*input_note, start, end))
  input_notes = np.delete(input_notes, 0, axis=0)
  input_notes = np.append(input_notes, np.expand_dims(input_note, 0), axis=0)
  prev_start = start

generated_notes = pd.DataFrame(
    generated_notes, columns=(*key_order, 'start', 'end'))
```

Now we increase the temperature to make the model a little more creative. We will start with the model generating 120 notes.

We stack the raw note data, here is what it looks like

Divides the pitch values by 128 so they represent a value between 0 and 1. the other 1,1 is step and duration which remain unchanged.

```
sample_notes
✓ 0.0s
array([[7.70000000e+01, 0.00000000e+00, 1.01302083e+00],
       [4.90000000e+01, 2.99479167e-02, 6.19791667e-01],
       [5.60000000e+01, 6.30208333e-01, 6.34114583e-01],
       ...,
       [5.60000000e+01, 5.20833333e-03, 6.05208333e+00],
       [3.80000000e+01, 9.07161458e+00, 1.70572917e-01],
       [4.50000000e+01, 1.70572917e-01, 1.43229167e-02]])
```

One by one, each note's pitch, step, duration are calculated, while the array of inputted notes is deleted one by one, and the new note is appended onto the existing sequence of notes

```
generated_notes.iloc[:10]
✓ 0.0s
```

	pitch	step	duration	start	end
0	67	0.180211	0.390485	0.180211	0.570696
1	99	0.427793	0.000000	0.608003	0.608003
2	99	0.422942	0.000000	1.030945	1.030945
3	87	0.414570	0.000000	1.445515	1.445515
4	97	0.411752	0.000000	1.857267	1.857267
5	87	0.409289	0.000000	2.266557	2.266557
6	78	0.408841	0.000000	2.675398	2.675398
7	92	0.409060	0.000000	3.084457	3.084457
8	67	0.408063	0.000000	3.492521	3.492521
9	89	0.410143	0.000000	3.902664	3.902664

At the end it creates a dataframe with all of our generated notes! Here we display just the first 10.

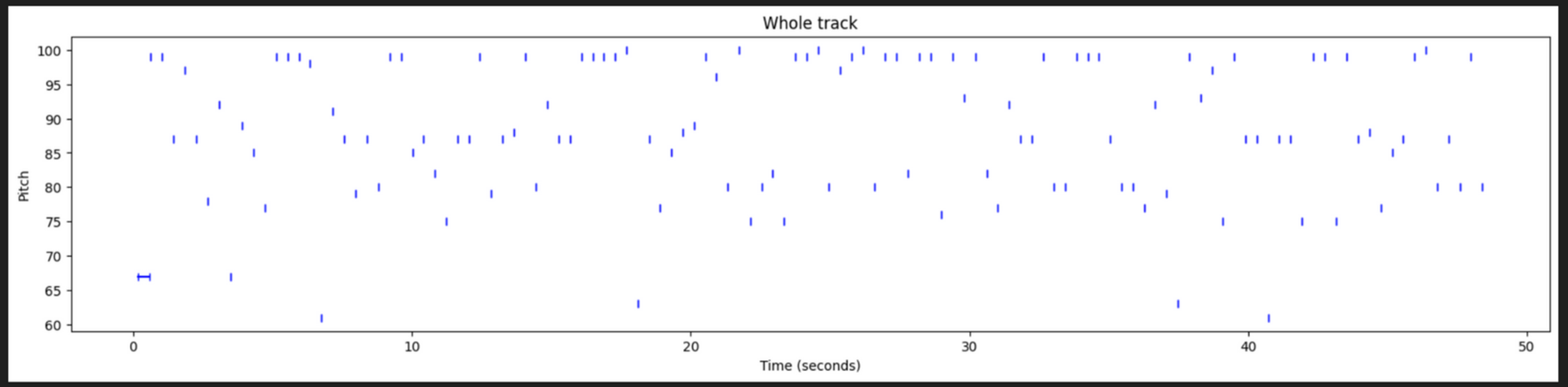
```
out_file = 'output.mid'
out_pm = notes_to_midi(
    generated_notes, out_file=out_file, instrument_name=instrument_name)
display_audio(pm)
```

✓ 2.0s

0:00 / 0:30

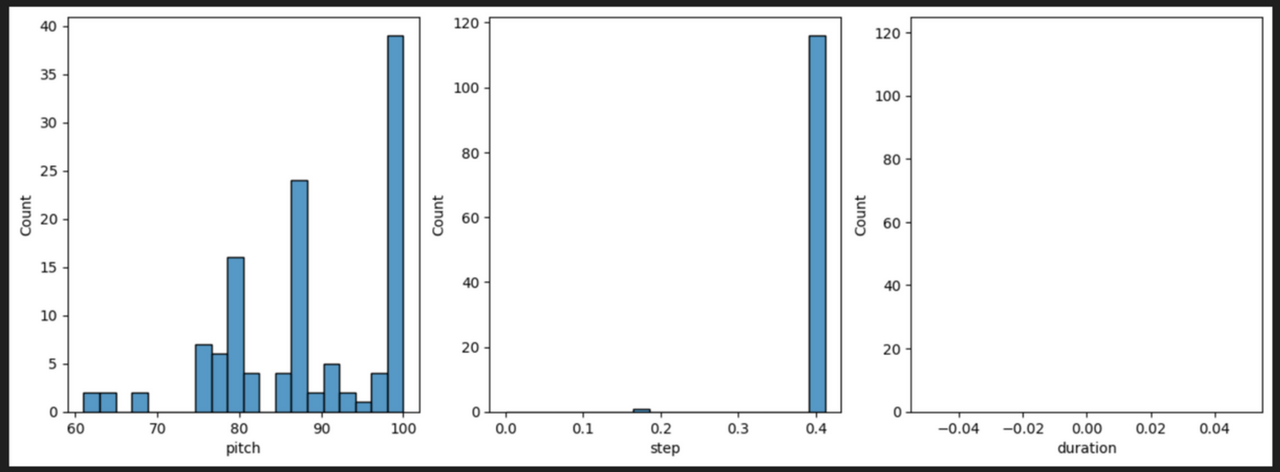
```
plot_piano_roll(generated_notes)
```

✓ 0.3s Pytho




```
#check the distrubutions of step, pitch, duration
plot_distributions(generated_notes)
#notice the change in distribution of the note variables
#because of the feedback loop between the model's outputs & inputs, it tends to generate
#similar sequences of outputs to minimize the loss
#relevant for step and duration, which use the mean squared loss
```

✓ 0.4s Pythor



We can listen to the audio and visualize the notes as we did earlier!