1. problem/data description

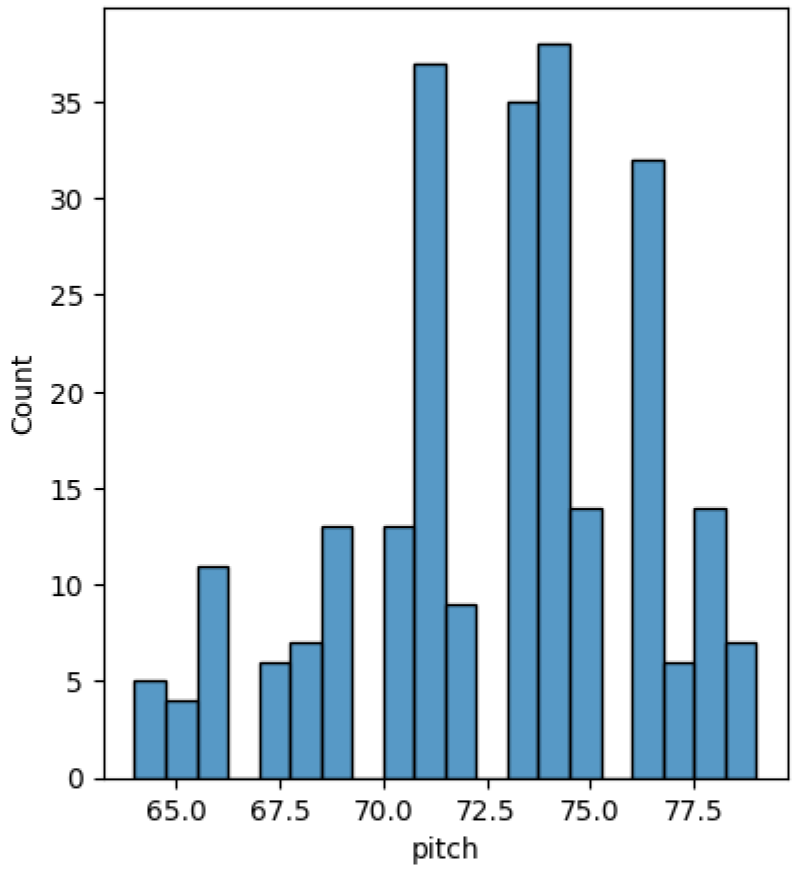
I chose to model classical music. Specifically I wanted to be able to generate classical music automatically with only the need of a prompt of a few notes from a human. And the quality of that music to be indistinguishable to a human from that written by a professional human. The data was scraped from https://www.midiworld.com/. MIDI (Musical Instrument Digital Interface) files are the language digital music synthesizers speak in. A note is represented by pitch, step and duration. There are 1325 total songs and 1,570,367 total notes.

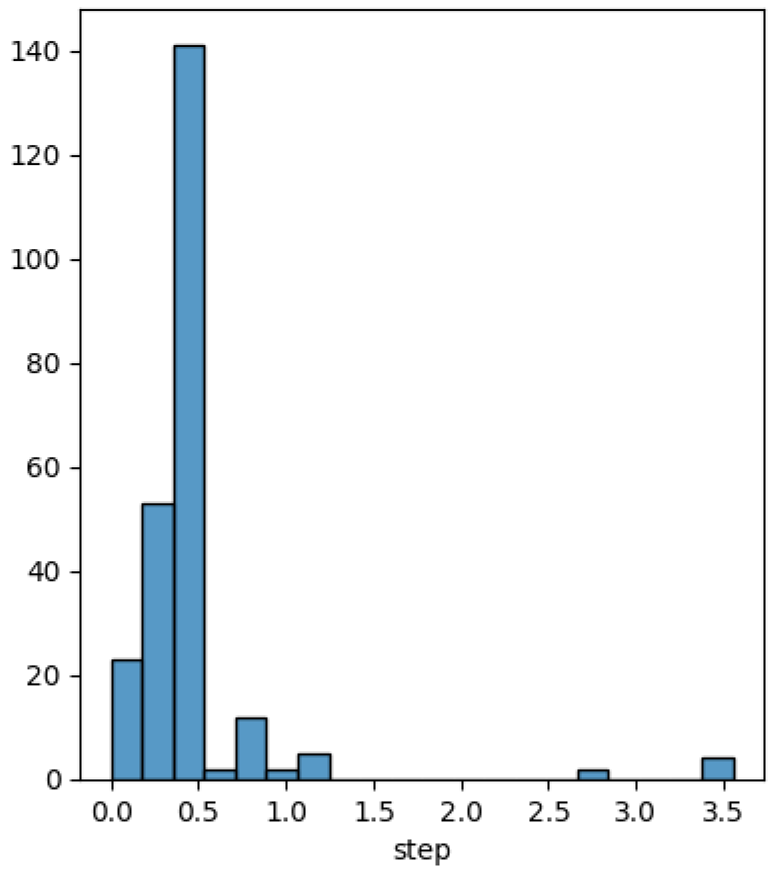
2. ETL (data wrangling)

The data required a decent amount of wrangling to get started. The files on midiworld were not presented in a standard format to make downloading easy. First I had to use some regex and BeautifulSoup to parse the html pages to extract the relevant downloadable files. Once downloaded they are in a .mid format. This needed to be converted to tabular data using pretty_midi library. Ultimately the relevant information which creates a note has a pitch, step, duration and start and end times. We only need pitch, step and duration to train our model. Note that pitch is the perceptual quality of the sound, step is the time elapsed from the previous note or since the beginning of the song and duration is the duration of the note.

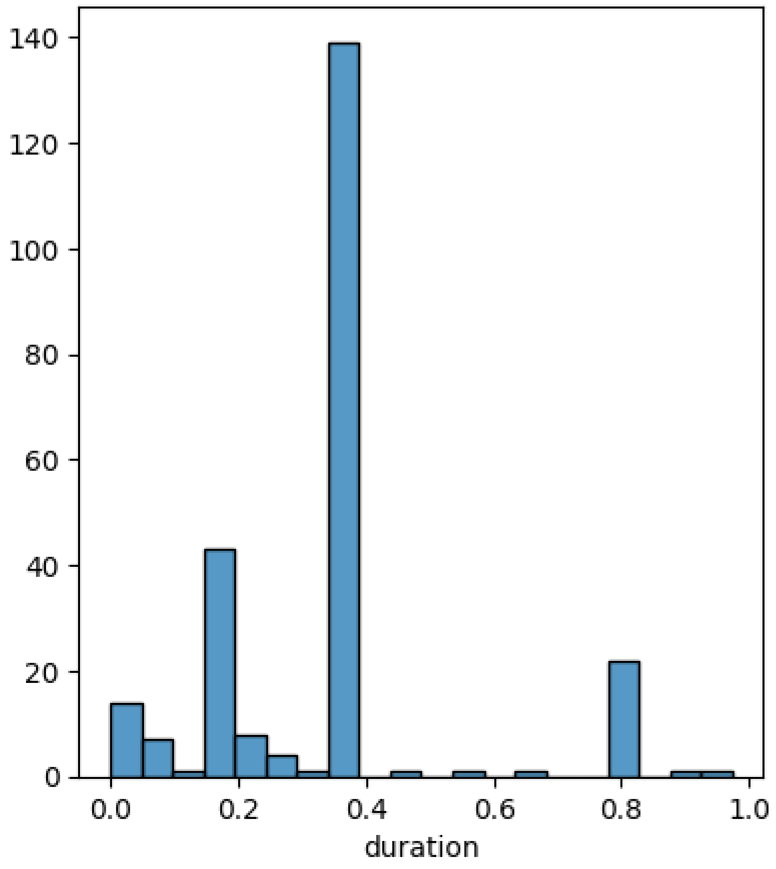
3. EDA (plots, exploration)

Once the data is in tabular form, it was pretty clean. No NAs were present. Here's what the distribution of the pitch, step and duration look like for a sample song.

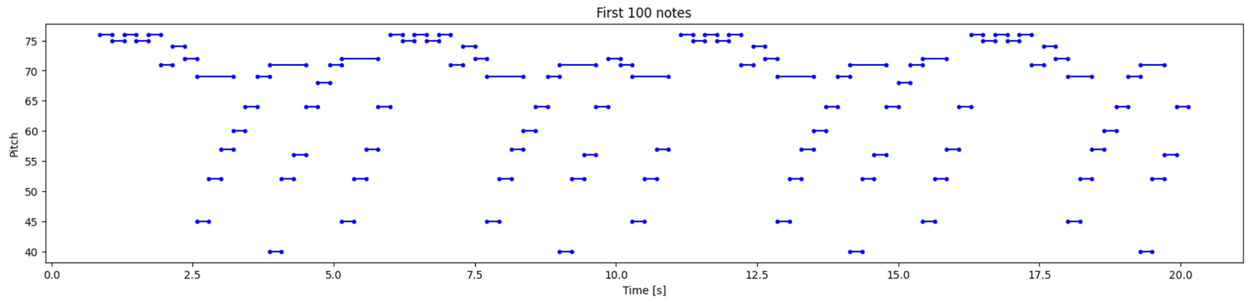




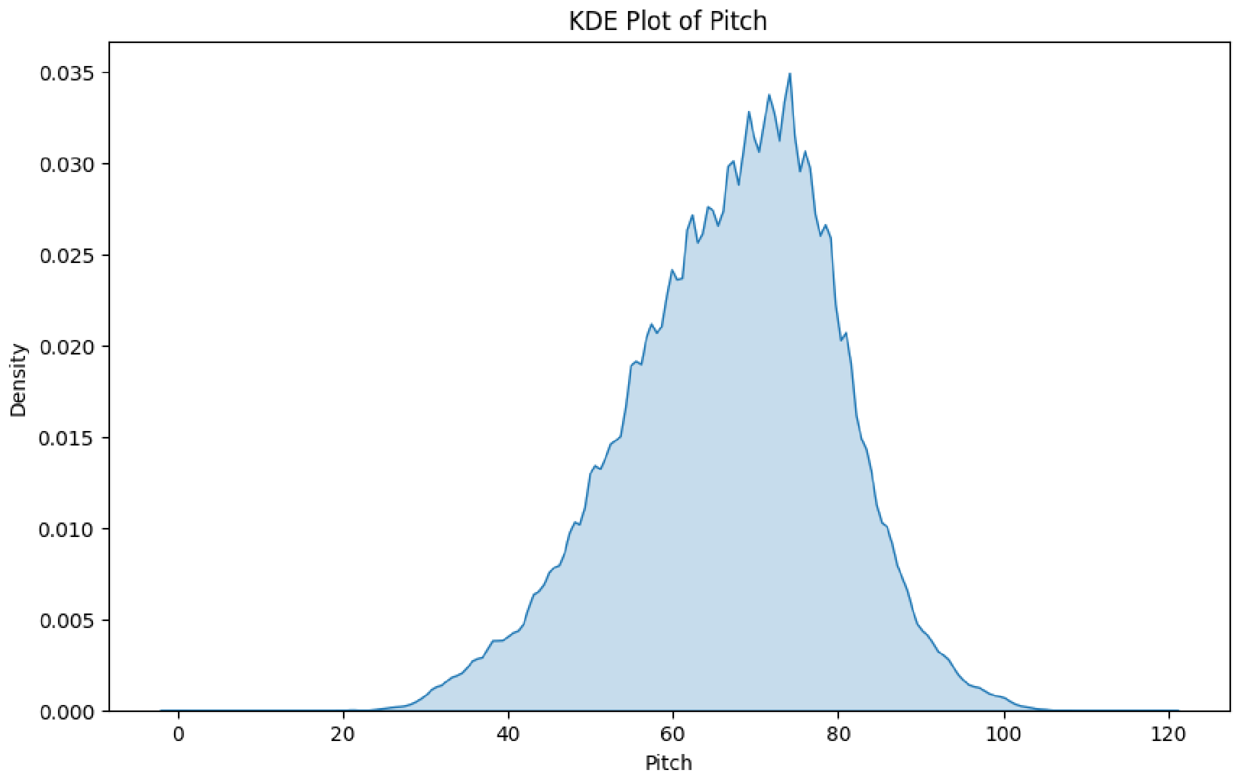
140
120
100
80
60
40
20
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
step

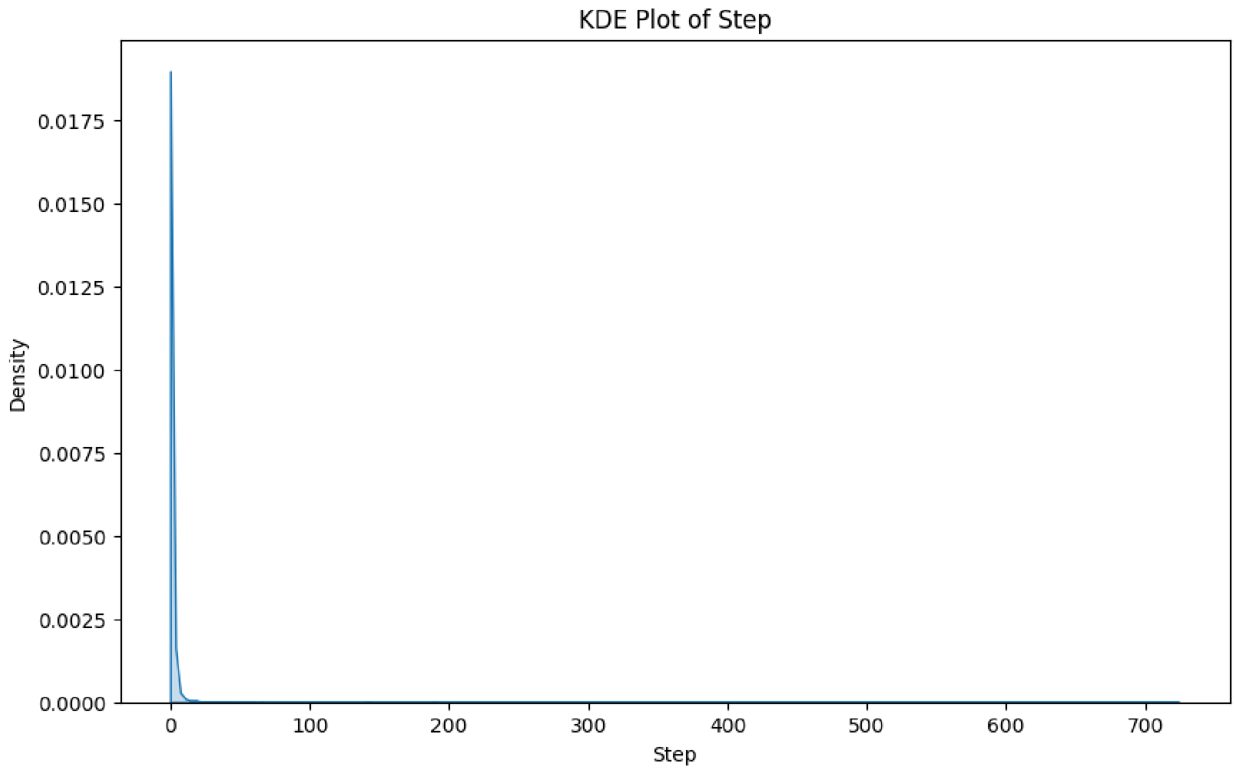


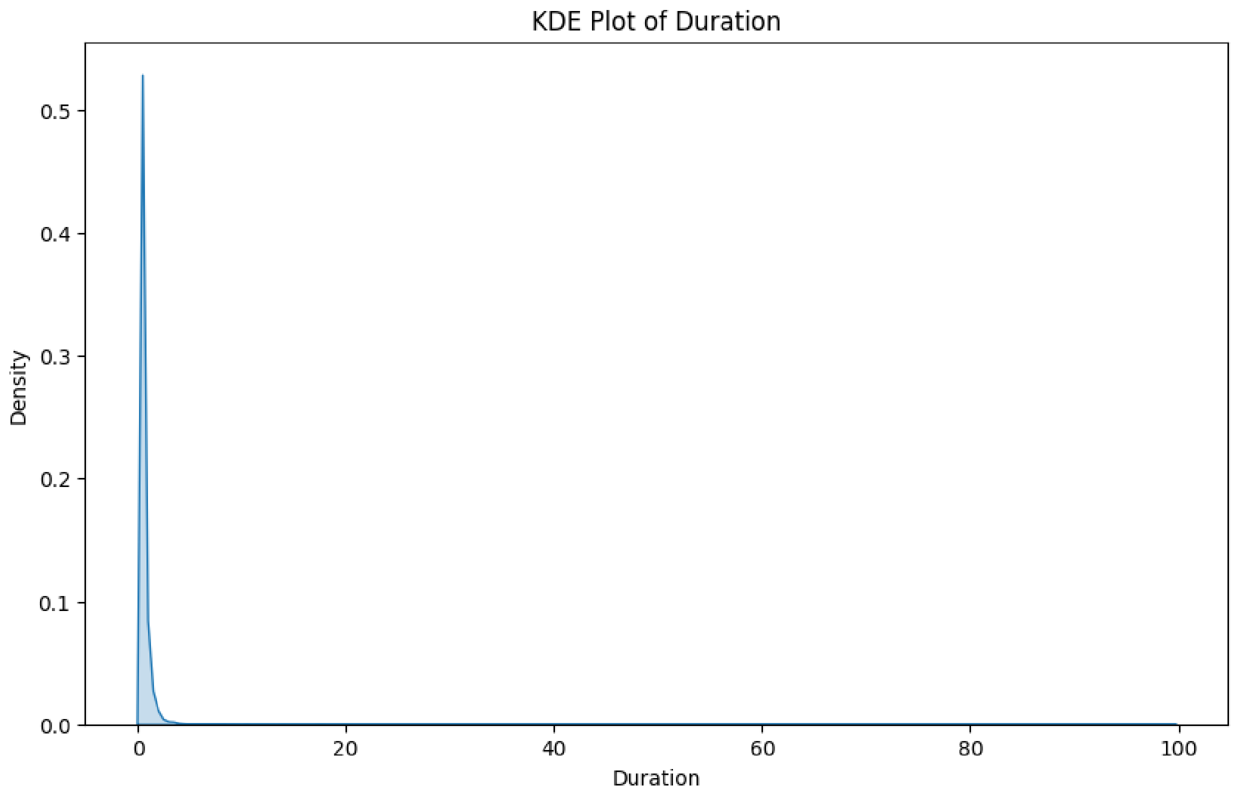
And here's a plot which has pitch on the y-axis, time in the song on the x-axis, and the length of the bars represent the duration.



Here’s the distribution of the entire dataset represented as kernel density plots:

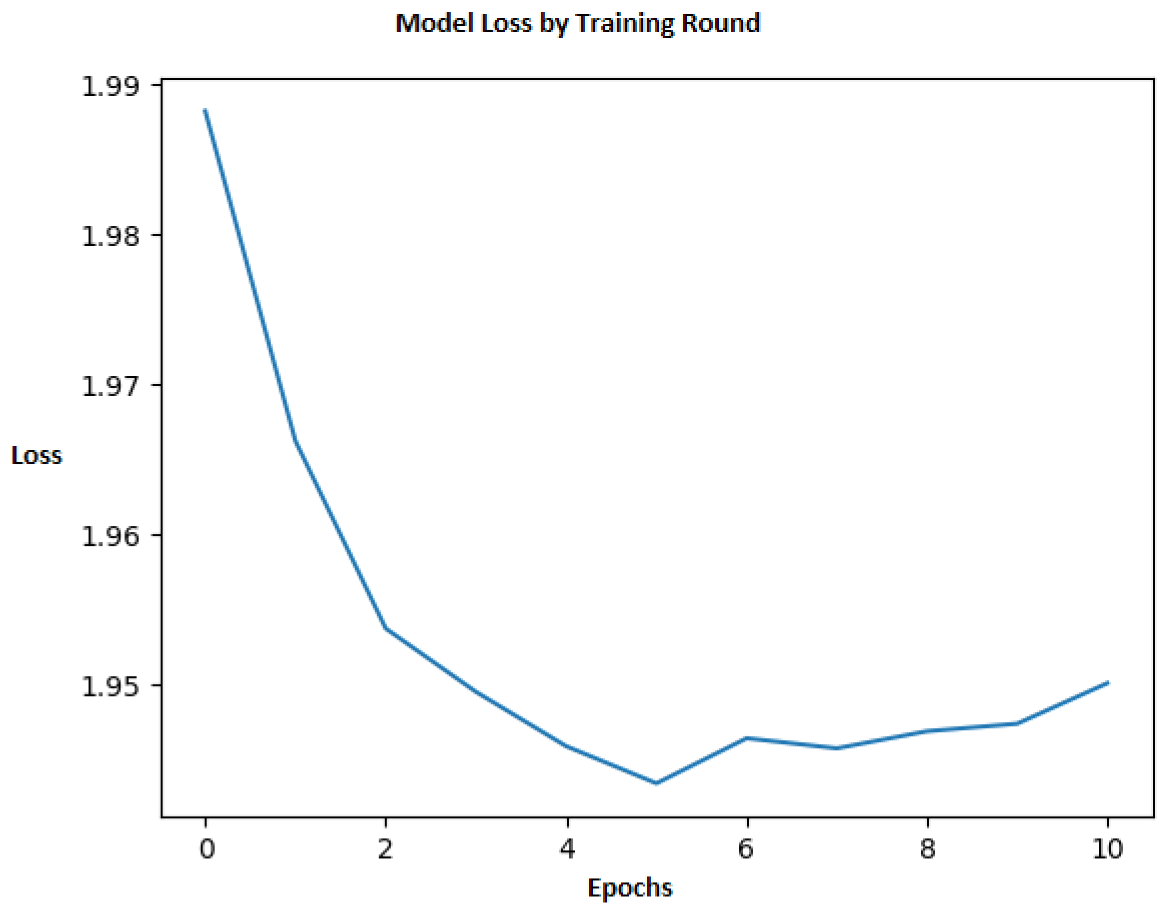




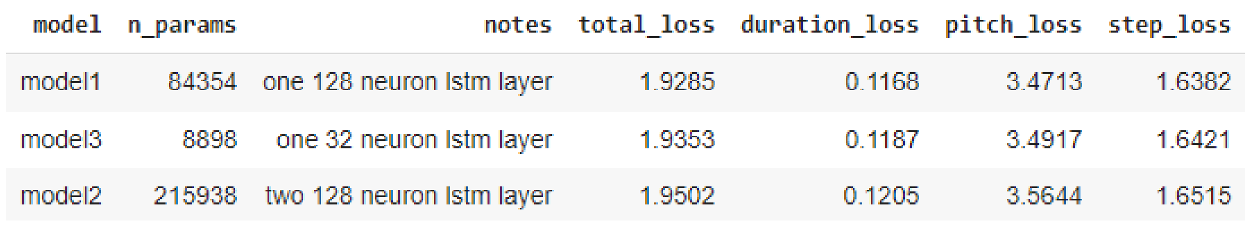


4. Modeling (hyperparam table, insights for data scientists)
I built an initial model with a relatively moderate depth, 128 neurons in the LSTM layer. Total of 84,354 parameters. Next I created a deeper model with another LSTM layer again with 128 neurons, total of 215,938 parameters and finally I wanted to simplify further from the initial model to verify that more neurons/layers will indeed fit the data better but since our goal isn’t necessarily creating models that can fit the data the best, I wanted to see how the generation of this model stacked up against the deeper models. The last and simplest model had 32 neurons in the LSTM layer with 8,898 parameters. I used 50 epochs for training. I used a sequence length of 25 notes so the model takes 25 notes as input and tries to predict the next note.

I used a weighted loss which gave pitch 0.05 weight and left step and duration as 1. This is because the scales for the three are not the same and the pitch loss was dominating without doing this. Also for step and duration a custom MSE loss function was used to put more pressure on the model to generate positive values. Here’s a plot of total loss as a function of the epoch. Total loss being a combination of the loss from pitch, step and duration.



The below hyperparameter table has MSE for total loss, pitch, step and duration for each of the 3 models. I'm also including total # of params and a short note on the model.



model	n_params	notes	total_loss	duration_loss	pitch_loss	step_loss
model1	84354	one 128 neuron lstm layer	1.9285	0.1168	3.4713	1.6382
model3	8898	one 32 neuron lstm layer	1.9353	0.1187	3.4917	1.6421
model2	215938	two 128 neuron lstm layer	1.9502	0.1205	3.5644	1.6515

Interestingly the deepest model has the worst performance per the MSE metric but again that's not necessarily what we're after, more on this later. One comment is that the difference in the losses aren't significantly different for the 3 models however.

5. Conclusion (insights for stakeholders)
Interestingly after I generated songs for each model using the same prompt, the deepest model, the one with the worst MSE metrics, created the most elegant sounding songs by a landslide, consistently as well.

So for any creators, using a deeper model seems to be the way to go, at least with this dataset and problem.

6. Future work (suggestions about what to try in the future to improve the model)
There's always room for improvement, we could try more complex models. There's also no shortage of music so we could train and test models on.

Aside from model improvement, it would be interesting to create an automatic way of scoring the music based on some sort of quality which didn't require any human input at all. What I have used for this project is a human in the loop interaction or feedback. Obviously to do this on many, many songs would require a significant amount of time. Another alternative for automatic scoring would be to take some samples of the length 25 sequences of notes, rank them using my judgment, then after my recurrent neural network encodes that sequence of notes, extract the embeddings. Now use the embeddings as features and my ratings as targets, train a logistic regression (or some other classifier) and use this as an automatic scorer going forward.

As far as deployment and use cases goes, this can be deployed as a web app and require a simple note or handful of notes from someone to generate a song based on the mood the user would like.