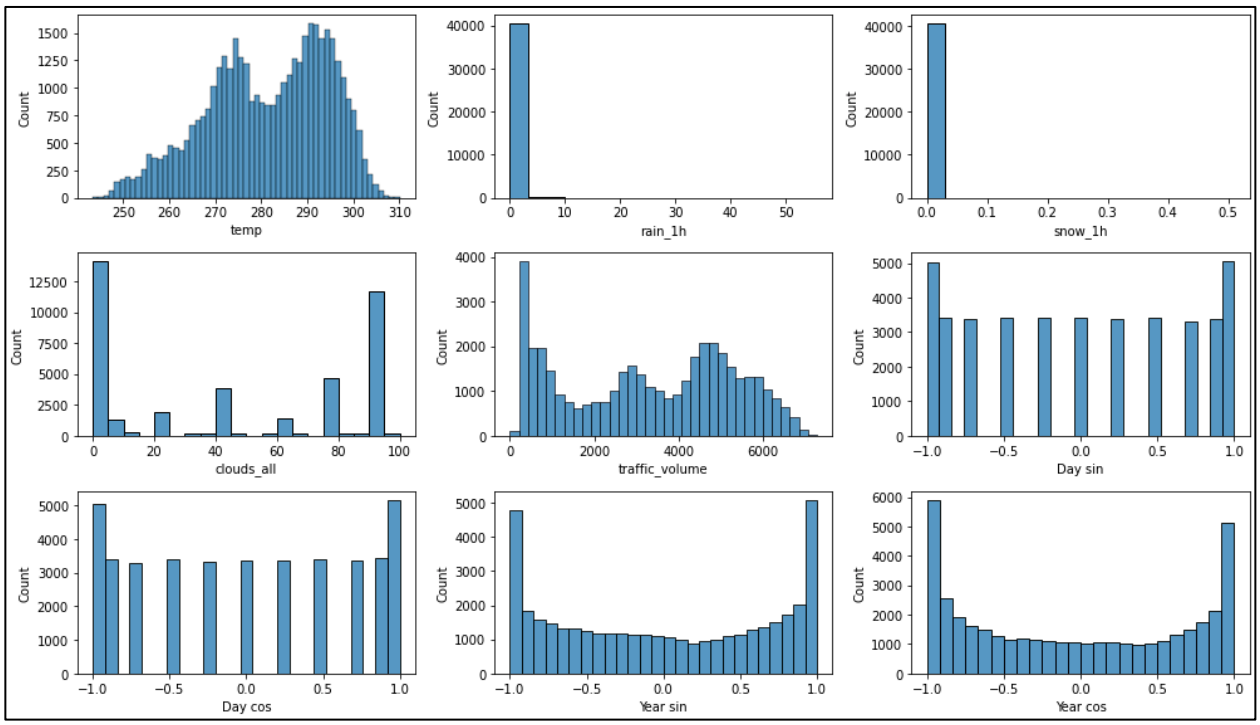
Aaron Gregory
Kaggle Name: AaronGregory23
CSC578 Async

Final Kaggle Competition

Before modeling anything I cleaned the data. I applied the sin and cosine transformation to the time stamps as shown in the example code. I thought this was a very clever of data manipulation that I hadn't seen before. I also got rid of the two weather descriptions as they were strings. Given more time I would've been interesting to add flags on things like fog and snow to see if that could improve the model. I decided to make holiday a binary attribute, either there was a holiday or there wasn't a holiday. Finally I looked for any incorrect data. I found that there was two erroneous entries, one for rain and one for temperature. I set these to the means of the feature set. Below is a set of histograms for the resulting data. Both rain and snow amounts have long tails as it is not precipitating most of the time. I did notice that there aren't data points at every hour so I think that may have some impact on there being spikes at 1 and -1 for the time data.



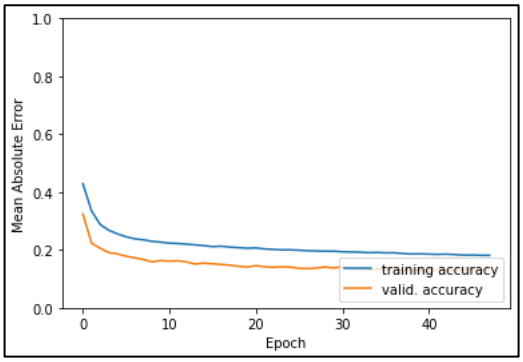
I ended up creating two baseline models for this project. The first was just a standard CNN model that I used to make sure my window slices were correct and that I could get reasonable test results. Once I had this working I created my baseline RNN model which was essentially just taken from the example code. This model only had one LSTM layer and no other optimizations. I didn't want to optimize this model as the goal was to show I could get results from a RNN as well and I would have a baseline for improvement. Setting this model up took me a long time since I didn't realize I was shuffling my data giving me results that didn't match up with where they should.

My first real model was similar to the first but was three LSTM layers with a 50% dropout rate. The last return sequence was turned to false so that I only got the predicted output at the end of the sequence. I fiddled with the dropout rate a little but ended up settling on 50 % since it learned faster and gave better results. In both this project and homework 7 I found having dropout really helped with training speed and accuracy.

I tried to incorporate GRU as the increased efficiency of two gates as opposed to three would logical speed up learning and improve results. When I added a GRU layer I saw neither effect. Maybe our dataset was too small to see the improvements but I got rid of it because it didn't seem to change anything for me. I also tried adding CNN to the model but couldn't get it to work correctly as my sizes were not correct. I chose not to explore statefulness as it did not seem applicable to this situation. LSTM cells are inherently statefull even though they "reset". Since weather changes frequently it feels unlikely that there would be necessary ringing effects that need to be considered from batch to batch.

```
lstm_model2 = tf.keras.models.Sequential([
    tf.keras.layers.LSTM(150, return_sequences=True),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.LSTM(150, return_sequences=True),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.LSTM(150, return_sequences=False),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(units=num_features)
```

My Final model was a three layer LSTM model with dropout. Just like in homework seven adding more layers helped to a certain point. I ended at three layers and then worked to optimize the number of units. I found that going to high lead to the model stopping learning early while going to low got bad results and learned very slowly. Even so I had to increase my max epochs in order for the model to finish learning. I update the patience to a larger number as well as sometimes the learning would stall out briefly before resuming.



Overall I was very happy to rank as high as I did for how much I struggled with this project. I really found it challenging looking at other people examples and trying to apply them to my own problem since everyone's code is tackling a different dataset and a different problem. In a class like this it sometimes feels overwhelming to understand everything if you don't break it down into smaller pieces. I ran into this problem when my first RNN model was giving me nonsensible results. I ended up using a standard CNN model that I could more easily understand to locate the problem. Start small and build from there has never felt so true to me as it does after completing this project. If I were to start this project from scratch I think I would prepare my data a little differently so it was easier to see sizes and shapes. I noticed that a lot of the RNN tutorials had more of a manual process to splitting their data so that there was an explicit x_train, x_label ,etc. I think this could be beneficial for trouble shooting as I struggled to figure out what dimensions my data was in at different points.