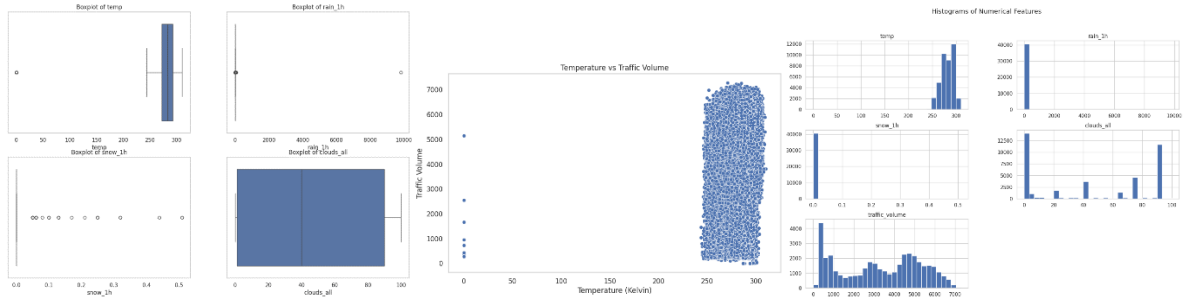
CSC578-910 – Final Project Documentation

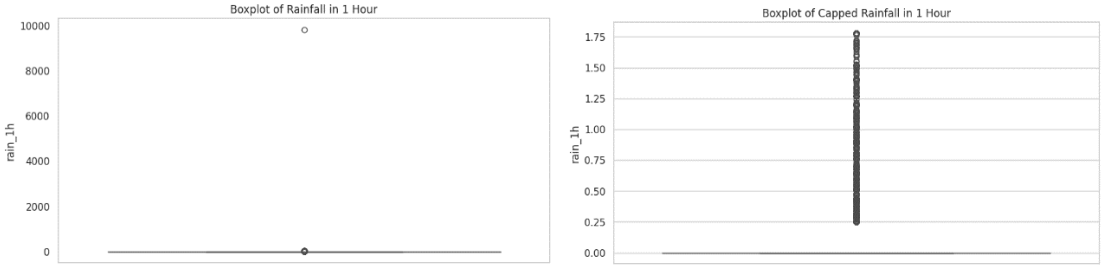
By Artemiy Yalovenko

Kaggle username: artemiyyalovenko (16th place 🤓)

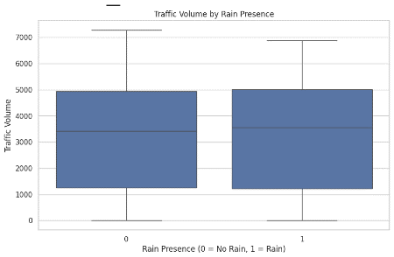
EDA was first performed to identify any issues that required attention during data preprocessing. The check for missing values showed that the holiday column had 99% missing values. An investigation of the rain_1h variable showed a significant outlier with a value of 9831.3 mm (clearly being an error) with only ~5% of values being non-zero. A similar investigation into the snow_1h variable showed that there were even fewer snow days, but compared to the rain variable they were more evenly spread out than the rain variable. Additionally, through visualization I was able to locate particularly weird temperature values of 0 Kelvin which is above -200 degrees Celsius… To visualize these extreme values and understand their distribution, boxplots and histograms were plotted for all numerical features.



Based on the EDA findings, preprocessing steps were taken. First, The holiday column was dropped. Instances with temp values around 0 K were replaced with the median temperature value. Values in rain_1h were capped at the 99th percentile to mitigate the impact of the outliers. This changed the distribution visualization of the rain variable to a more workable one:

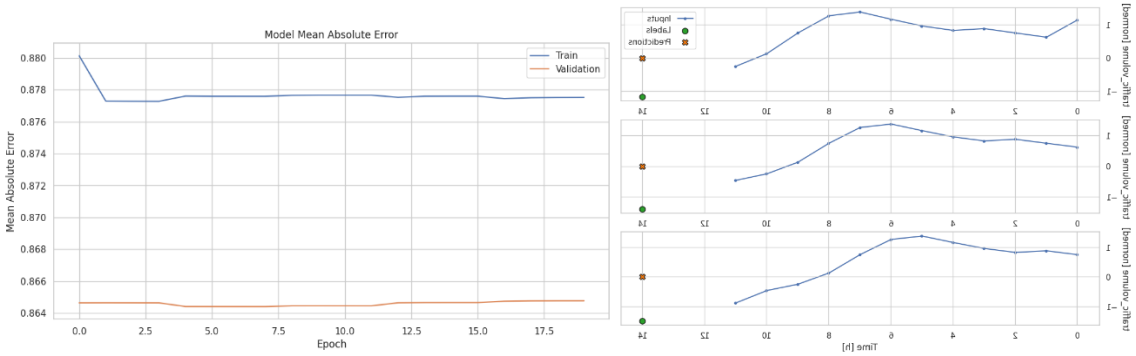


New columns for hour, day, month, and year were created from the date_time column. Sinusoidal encoding was applied to cyclical features like hour and month. The weather_main and weather_description columns were converted to dummy variables. A new binary variable rain_present was created to indicate whether there was any rainfall, which was likely useless due to low correlation. A correlation plot was also generated to make sure there wasn't need for multicollinearity reduction. After these preprocessing steps, the data was split into test, validation, and training sets and using sklearn StandardScaler we normalized the data using the z-score method. The WindowGeneratorClass was then copied according to the tutorial provided and a WindowGenerator entity (w1) with input width=12, label_with, and offset (or shift) of 3 was creates. After constructing a Baseline and Baseline LSTM model, changes were made to decrease loss parameters and reduce underfitting at first, and later attend to model overfitting.



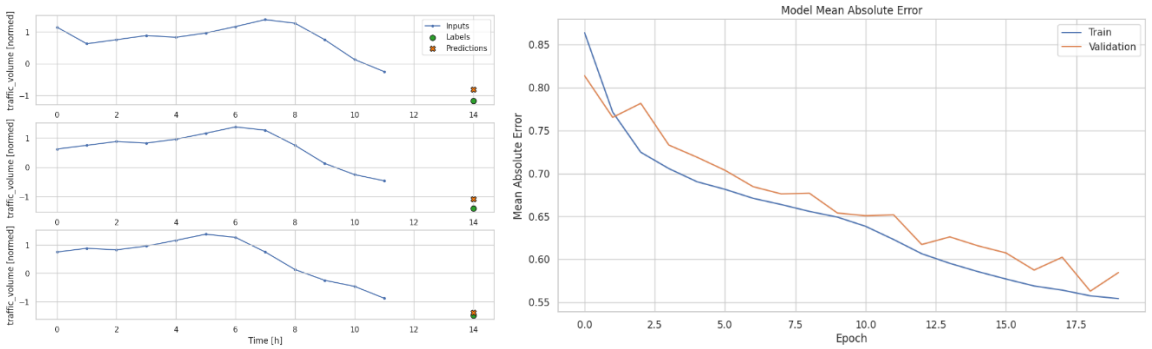
The baseline model for this project consisted of a an LSTM model with an LSTM layer and a dense output layer; the same one as in the tutorial. The only adjustment was that the baseline model included an Adam optimizer with a reduction of LR on plateau, which was also used throughout the project though changing in patience from 3 to 5 on some models. For this project all of the model used MAE as the loss parameter. Looking at the visuals and the results its clear that the baseline model was clearly underperforming with high loss and MAE value and was not improving even as the learning rate dropped after multiple (3 epoch) plateaus.

```
Validation performance: {'loss': 0.9752177000045776, 'mean absolute error': 0.8647704720497131}
Test performance: {'loss': 1.0007541179656982, 'mean absolute error': 0.8763428330421448}
```



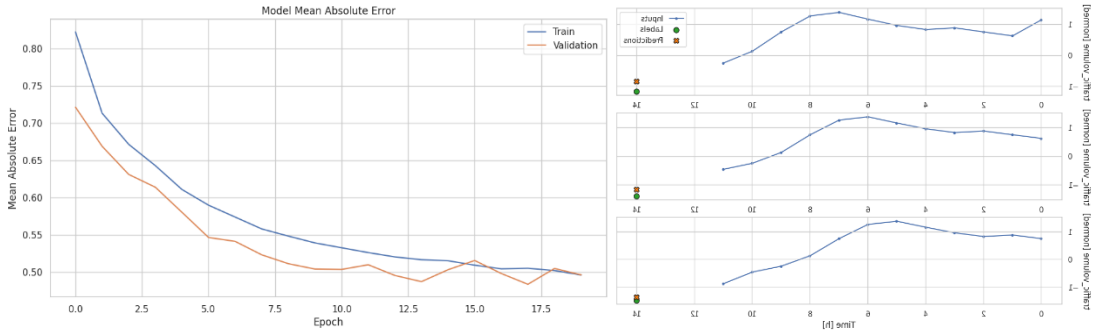
This started the process of attempting to first increase model complexity to decrease loss metrics, until overfitting was observed. We started with adding more units and a dropout layer where each layer had 64 recurrent units and a dense layer with a single unit. Then after setting the learning rate to 0.001 (Adam optimizer standard) we reduced it to 0.0001. This caused the model to overfit leading to implementation of two dropout layers with 0.2 rate after each lstm layer. Noticing that the plateau based auto scheduler reduced eta too early, causing overfitting, a patience adjustment from 3 to 5 was made leading to the first significant decrease in model MAE loss.

```
Validation performance: {'loss': 0.5355097651481628, 'mean_absolute_error': 0.5846064686775208}
Test performance: {'loss': 0.3971576988697052, 'mean_absolute_error': 0.4928072690963745}
```



This maintained to be the best model for a while. Attempts at increasing epochs caused overfitting at any number which even increase in dropout rate to .3 wouldn't help. Recurrent units were then increased to 128 at each LSTM layer while leaving the dropout at .3 which lead to another MAE drop, but with overfitting tendency and lots of instability. This lead to L2 regularization adoption, which conceptually made a lot of sense since EDA showed variables like rain, snow and temperature having lesser effect than was initially anticipated. This model was the best for a while and was submitted to Kaggle decreasing my score by 1100 points which speaks to how poor the other models were. This was the most surprising model to me personally, because it dropped MAE by around 0.25 even though the adjustment was simple regularization even with a penalty that was lessened to 0.001 from default 0.01.

```
Validation performance: {'loss': 0.4739739000797272, 'mean_absolute_error': 0.4966530203819275}
Test performance: {'loss': 0.3405143916606903, 'mean_absolute_error': 0.3972609341144562}
```

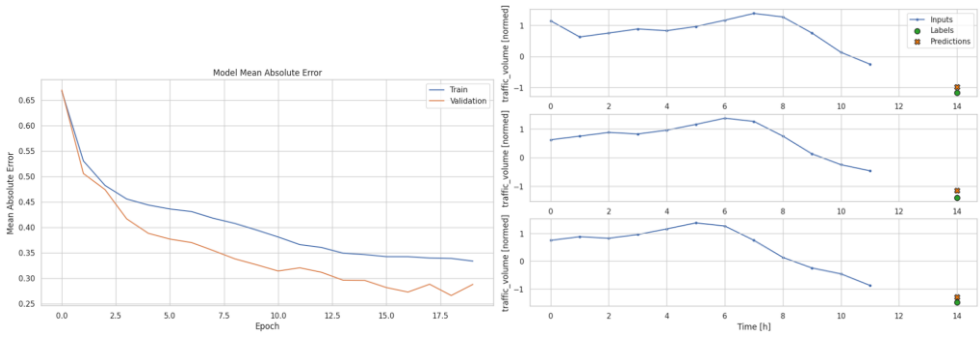


I personally believe this model to be the runner up because of its simplicity even though it placed closer to 4th in terms of results across all of the models. This model only narrowly lost to 1D convolutional neural network with GRU, which in turn narrowly lost to 1D convolutional neural network with RNN.

We attempted more advanced models after multiple iterations of the RNN+LSTM with regularization and two 128-unit layers yielded inconclusive results. CNN with RNN was tried and tested with the best one having two Conv1D layers each set at 64 units, max pooling layers, followed by dense and reshape layers, as well as dropout and L2 regularization at 0.001 penalty. A CNN and GRU model that yielded best results had two Conv1D layers each set at 64 units, max pooling layers, followed by dense and reshape layers, followed by two GRU layers with Dropout and L2 regularization. But no model performed better than bidirectional LSTM RNN which uses three Bidirectional LSTM layers with 128 units each, paired with dropout layers set at .0.2 for regularization, and an output dense layer with L2 regularization set at 0.001 to reduce overfitting. It yields very consistent and accurate results, while not compromising too much on complexity compared to the CNN based models.

Kaggle submission:

```
Validation performance: {'loss': 0.2195841670036316, 'mean_absolute_error': 0.28828078508377075}
Test performance: {'loss': 0.19987863302230835, 'mean_absolute_error': 0.2913227379322052}
```



Most recent run

```
Validation performance: {'loss': 0.2097020447254181, 'mean_absolute_error': 0.27295613288879395}
Test performance: {'loss': 0.1465262919664383, 'mean_absolute_error': 0.21986787021160126}
```