# Weather Patrol
# EE 400A, TinyML

Aditya Krishna, Pujan Patel
6-6-2024

**Problem Statement:**

Seattle Weather is quite unpredictable, as the last few weeks have exemplified. You may look out the window in the morning on a gloomy rainy day, only to find the sun baking your winter clothing as you leave for work. Alternatively, you may wake up to a sunny clear morning, only to be cursing yourself as you are chilled by the frigid winds waiting at the bus stop. It's only natural to dress up based on the assumed weather, and this becomes increasingly difficult due to the unpredictability of Seattle Weather. Thus, our project aims to predict a crucial aspect of weather—rain. Based on the current month, temperature, relative humidity, and air pressure—all data gathered by an Arduino Nano 33 BLE and BME 680, we attempt to predict whether it will rain in the next 30 minutes.

**Data Set:**

We were attempting to predict weather patterns in the Seattle region, so it only made sense to have data that measured past weather patterns within Seattle. Fortunately, the University of Washington's Atmospheric Sciences department has a weather station that has been collecting exhaustive data at intervals of one minute since July of 1999. They have also been kind enough to publish this data on their website[2], so we thought this would be the perfect data set for our purposes.

**Approach:**

We began by conducting a preliminary analysis on the dataset. Figure 1 shows a sample of the raw data. As we can see, it contains several environmental readings collected every minute. However, we see a 'M' in the pressure column suggesting that this reading was missing at the time. Since we do not want to train our model on incomplete data, we began by removing all rows (data entries) where at least one of the columns (environmental readings) were incomplete. Next, we plotted some of these environmental precursors against the rain column to see whether there was noticeable correlation. Notable plots are provided in Figures 2. These figures reinforce logical assumptions that we might make about the weather. Namely, it's unlikely to rain when it's either very hot or very cold, and it's more likely to rain as we get further away from summer in either direction. Further analysis while respecting the cost of these sensors led us to conclude that the ideal parameters to measure would be the Month, Temperature, Relative Humidity, and the Atmospheric Pressure. Additionally, we recognized that climate change may lead to poor generalization and concluded that training with data from 2022 onwards would minimize this and lead to the best results. Finally, our initial problem statement had a binary classification—no rain or rain represented as 0.0 and 1.0 respectively.

| | datetime_UTC | Pseudo-Julian-Date | Date | Time | Temperature | Dewpoint | Relhum | Speed | Gust | Direction | Pressure | Solar | SumRain | Rain |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1999-07-30 00:09:00 | 2.451390e+06 | 1999-07-30 | 00:09 | 62.200000 | 54.500000 | 75.9 | 6.600000 | 9.100000 | 317 | M | 223.6 | 0.0 | 0.00 |
| 1 | 1999-07-30 00:10:00 | 2.451390e+06 | 1999-07-30 | 00:10 | 62.100000 | 54.100000 | 75.2 | 7.500000 | 12.000000 | 357 | M | 216.8 | 0.0 | 0.00 |
| 2 | 1999-07-30 00:11:00 | 2.451390e+06 | 1999-07-30 | 00:11 | 61.500000 | 53.600000 | 75.2 | 6.600000 | 9.100000 | 9 | M | 212.6 | 0.0 | 0.00 |
| 3 | 1999-07-30 00:12:00 | 2.451390e+06 | 1999-07-30 | 00:12 | 62.100000 | 54.300000 | 75.7 | 6.300000 | 9.100000 | 345 | M | 211.6 | 0.0 | 0.00 |
| 4 | 1999-07-30 00:13:00 | 2.451390e+06 | 1999-07-30 | 00:13 | 62.600000 | 54.800000 | 75.4 | 4.700000 | 9.100000 | 336 | M | 213.2 | 0.0 | 0.00 |

Figure 1: A sample of the raw time series data provided by the Atmospheric Sciences Weather Station.

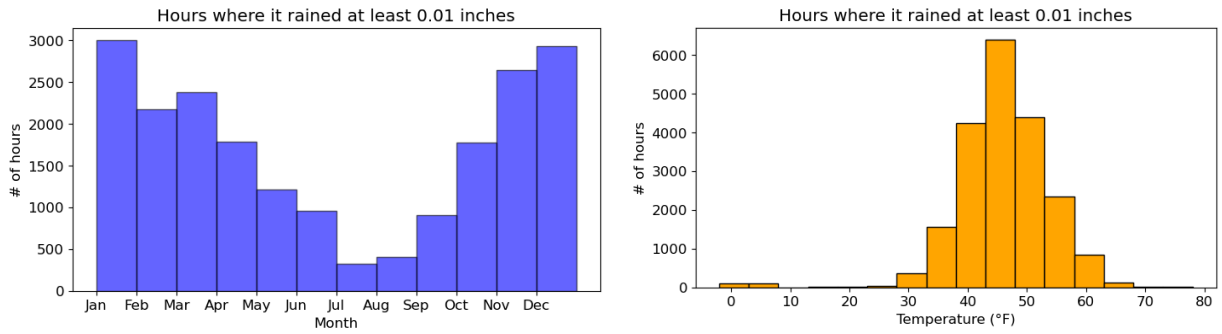


Figure 2: The number of hours where it rained at least 0.01 inches plotted against the temperature (Right) and the month (Left) at those hours.

These were the initial steps of preprocessing, and the results can be seen in Figure 3. Here we see each row (data entry) has a month, temperature, relative humidity, atmospheric pressure, and a classification. The first four columns then represent the inputs into our hypothetical model and the last column represents what we wish the output to be. Finally, we normalized the data entries and serialized the rain using one hot encoding. Thus, rainfall > 0 was encoded as [0, 1] and rainfall == 0 was encoded as [1, 0] as seen in Figure 4. Creating a 60/20/20 split, we were ready for training. Figure 5 shows our sensor and its output, showing a proof of concept of how we were going to deploy and measure these parameters.

| | month | temperature | relhum | pressure | class |
|---|---|---|---|---|---|
| 0 | 5 | 44.4 | 88.5 | 1014.6 | 0.0 |
| 1 | 7 | 61.2 | 72.5 | 1023.6 | 0.0 |
| 2 | 11 | 59.9 | 97.2 | 1001.6 | 0.0 |
| 3 | 12 | 43.8 | 85.1 | 1002.9 | 0.0 |
| 4 | 12 | 50.0 | 75.8 | 996.0 | 0.0 |

Figure 3: A sample of the intermediately processed time series data

| | n_month | n_temp | n_relhum | n_press | p_norain | p_rain |
|---|---|---|---|---|---|---|
| 0 | -1.349449 | 0.063454 | 1.185632 | 0.399001 | 0.0 | 1.0 |
| 1 | -0.812923 | -1.220894 | 0.592793 | -1.657773 | 0.0 | 1.0 |
| 2 | -0.276397 | 0.177618 | 0.335297 | 0.286813 | 0.0 | 1.0 |
| 3 | 1.601444 | -0.726182 | 1.095808 | 0.386535 | 0.0 | 1.0 |
| 4 | -1.081186 | -1.677551 | 1.155690 | -1.732565 | 0.0 | 1.0 |

Figure 4: A sample of the final processed time series data

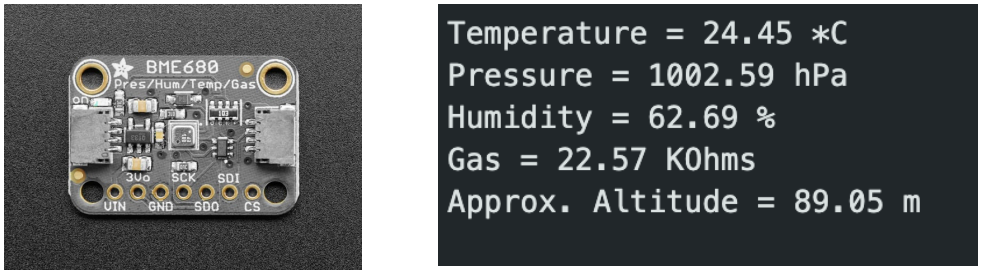


Figure 5: Sensor (Left) and Test Output (Right)

Training a machine learning model with this processed data, we reached a Training Accuracy of 93% which seemed phenomenal. However, Figure 6 paints a completely different story. Our model had simply learned to predict zeros (no rain). Figure 7 (left) does an excellent job of suggesting why this was the case. Since we had ~30000 entries with No-Rain and only ~2500 entries with Rain, our model learned to be biased towards No-Rain. Since No-Rain was statistically ~12 times more likely to appear than rain, our model was right 93% of the time when predicting No-Rain.

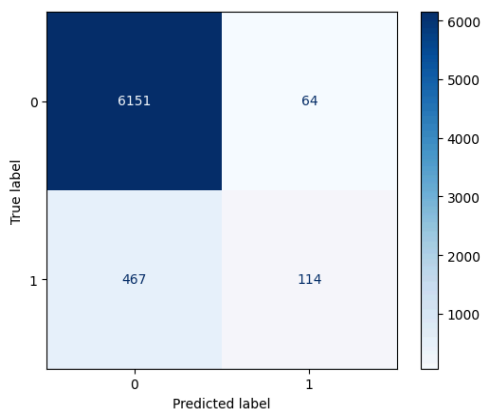


Figure 6: Initial Confusion Matrix

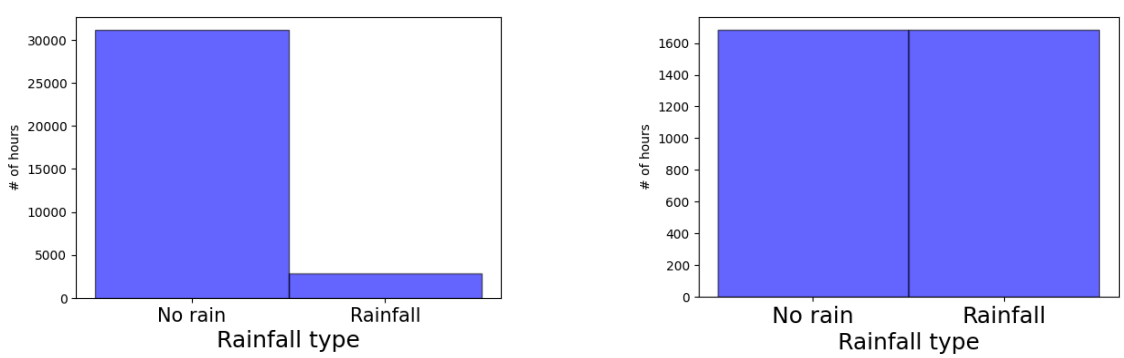


Figure 7: Frequency of each classification in the training set for pre-sampling (left) and post-sampling (right)

This led us to attempt methods of subsampling to unlearn this bias. We began with SMOTE sampling, which attempts to interpolate the underrepresented classification. In our case, it

interpolated among the Rain entries to generate new data until we had approximately equal numbers of No-Rain and Rain events. However, this produced sub-optimal results. We believe this could be because of the interpolative method SMOTE uses to generate synthetic data. We were unsure if this interpolated data was sufficiently representative of true weather patterns so we decided to set this aside. Next, we utilized the principles of Bootstrap Sampling which states that if one has a good representation of the population, sampling within it will still produce a good representation. We incorporated some Gaussian noise into this (deviating from theory a bit) to generate new data samples for No-Rain until we had approximately equal numbers of No-Rain and Rain events. This also did not produce promising results. Finally, we attempted to under sample the dominant classification—No-Rain leading to Figure 7 (right). Resampling within the No-Rain class to gather approximately equal numbers of No-Rain and Rain events, the results were quite respectable as we will discuss further in the Results and Analysis section.

Finally, we began tuning the Model Architecture to better fit the task. It didn't make sense to use convolutional layers since each parameter held its own significance and range of values, so we would be losing information if we were to do so. Dense layers made the most sense because we wanted to perform linear combinations across each of the four parameters to predict the rain, with some parameters having heavier "weights" than others. Finally, adding a dropout layer helped reduce the likelihood of overfitting to the data, something we were conscious about after the previous results. The resultant architecture can be seen in Figure 8. The unquantized model was approximately 600kB which was within the 1MB constraint on the Arduino Nano, but we still quantized it since some space would be occupied by the sensor drivers and communication protocols.

**Model: "sequential_2"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| Input (Dense) | (None, 4) | 20 |
| dense_6 (Dense) | (None, 128) | 640 |
| dense_7 (Dense) | (None, 128) | 16,512 |
| dense_8 (Dense) | (None, 64) | 8,256 |
| dropout_1 (Dropout) | (None, 64) | 0 |
| Output (Dense) | (None, 2) | 130 |

**Total params:** 76,676 (299.52 KB)

**Trainable params:** 25,558 (99.84 KB)

**Non-trainable params:** 0 (0.00 B)

**Optimizer params:** 51,118 (199.68 KB)

Figure 8: Model Architecture of the Pre-Quantized Model

**Experimental Results:**

Once we settled on our model architecture, we evaluated our model on test data we had set aside from the initial 60/20/20 split. We performed this evaluation by generating a confusion matrix along with F-1 scores for the model's predictions in comparison to the ground truths.

Figure 9 shows this confusion matrix where the accuracy for each class is > 75% and the F-1 scores are 0.823 and 0.813 respectively for no rain and rain.

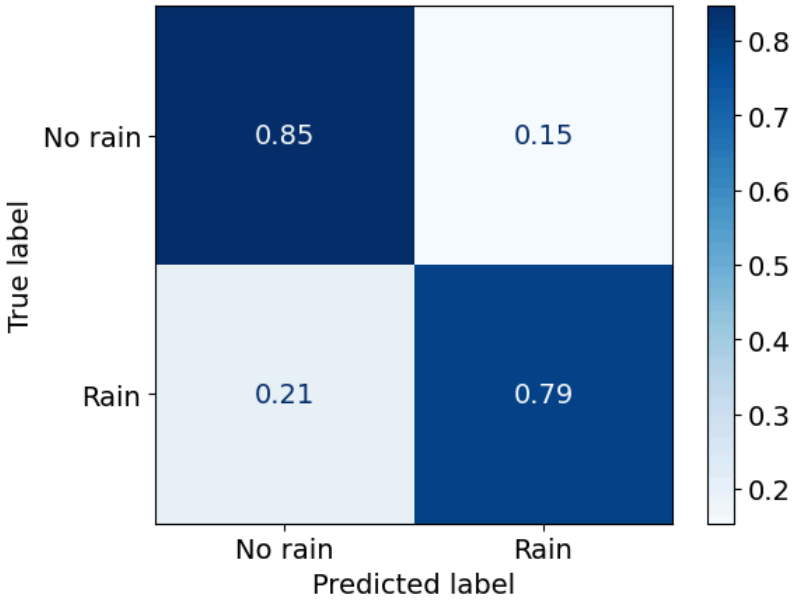


Figure 9: Confusion matrix of final model trained on balanced data

We believe this is good performance from our model and further supports balancing our training data to have equal distribution of classes. According to literature, F-1 scores > 0.8 are good for predictions. Furthermore, we identified that our rain prediction model needed to perform well for specific months. A proper rain prediction system should perform well regardless of the month by understanding their respective weather nuances.

To understand how our model performed across specific months, we split our test dataset into month subsets and predicted on each subset to generate corresponding confusion matrices. Below in Figure 10, we have included the matrices for all of our individual months. The two notable month comparisons are September and August. For September, our model's F-1 scores were 0.903 and 0.899 respectively for no rain and rain. These F-1 scores indicate an excellent performance from our model. On the other hand, for August, our model's F-1 scores were 0.721 and 0.41 respectively for no rain and rain. These F-1 scores indicate very poor performance. Still, generally our model does well over each month so our overall confusion matrix turned out to have > 0.8 F-1 scores for each class. We believe the August poor performance stems from how we purposely wanted our model to understand that Summer months have very low rain. So if it did have any rainy conditions in Seattle, our model still had bias towards saying it would not rain if it was during a Summer month. We believe incorporating a dew point sensor to our system would help alleviate some of these issues since dew point looks at the temperature needed to cause precipitation. We also believe that we could oversample the July-August rainfall events to alleviate these issues as well so the model understands what conditions would potentially cause rainfall during the Summer months.

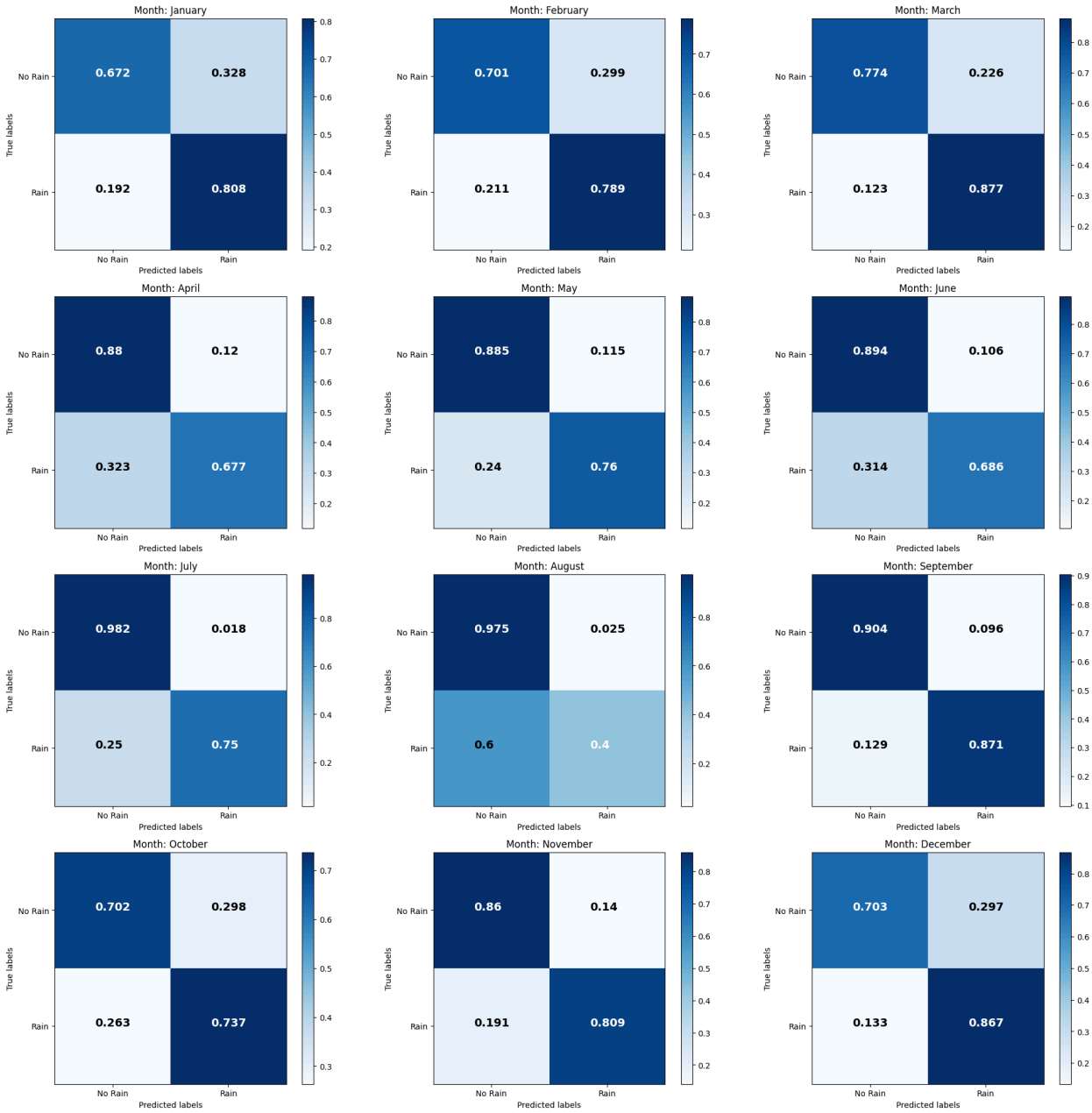


Figure 10: Confusion matrices split across each month to show variability in performance

**Deployment:**

With the results we achieved, we were satisfied with moving forward and deploying our model. For this, we used TensorflowLite to quantize and reduce our model size to be deployed. A quick evaluation of the new TensorflowLite model's performance showed very little drop in performance which we were very pleased with. We then developed Arduino IDE code to measure 4 input variables, invoke the TensorflowLite interpreter, and provide predictions whether it would rain in the next 30 minutes or not. In order to make our system deployable, we opted to receive these predictions via bluetooth low-energy by using the Arduino Nano's nRF52840 ARM Cortex-M4F CPU processor. We then uploaded this code onto our Nano.

We then began packaging our Nano along with its sensor into a system that could be easily deployable. We began by connecting the BME 680 sensor to the Nano utilizing I2C protocol and placed it on a breadboard as shown in Figure 9 (left). We also added a battery enclosure to provide a consistent power source so we could disconnect our laptop from the Nano. Next, we placed the breadboard into a water-proof enclosure as seen in Figure 9 (right). We made sure to leave some space when we closed the container so air can still enter. This enabled the sensor to read accurate measurements while still remaining safe from any water or environment damage. We also tied a rubber cord around it, so that it could be tied around a tree.

To access the bluetooth low-energy notifications, we used the LightBlue app downloaded on our smartphones to connect to the Nano. Figure 10 shows the device in deployment outside of Henderson Hall and Figure 11 shows the accompanying message received on Aditya's phone.

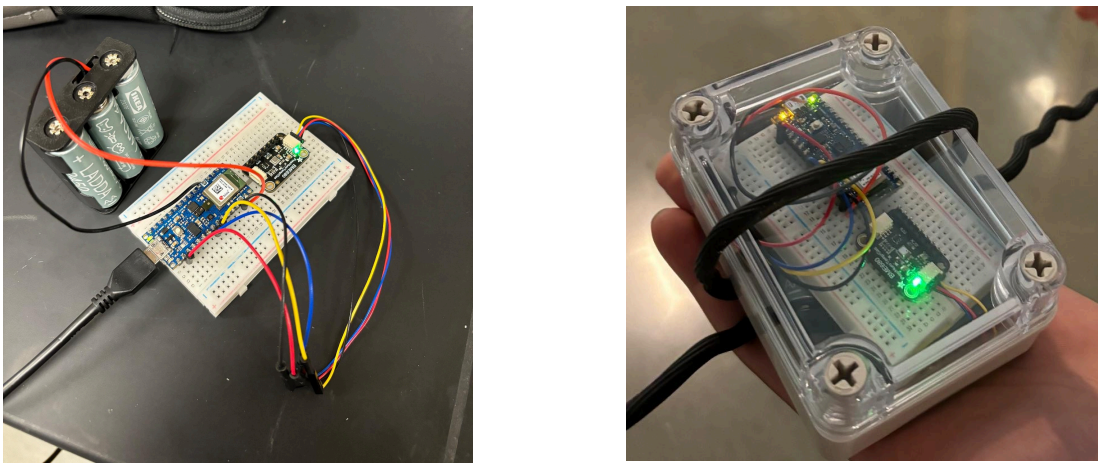

Figure 9: The breadboard configuration (left) and deployment configuration (right)

Figure 10: Deployment outside of Benjamin Hall

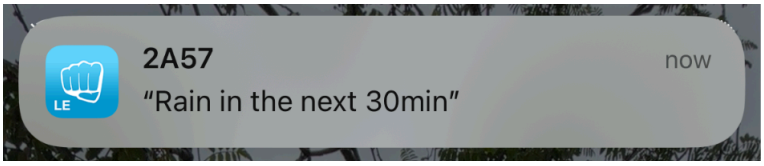


Figure 11: Notification on Aditya's Phone from the Arduino Nano BLE 33

References:

[1] Code: https://github.com/aditya-uw/WeatherPatrol

[2] "NW_Weather," Live From K-12 Earth & Mars. http://www-k12.atmos.washington.edu/k12/grayskies/nw_weather.html (accessed Jun. 06, 2024)