1. Performance

After I was done training and tuning my model, it underwent extensive performance testing.

Table 1. List of tests applied to the model to evaluate its performance and the results.

Test	Result
Mean absolute error	2 hPa
Mean squared error	11 hPa^2
Root mean squared error	3 hPa
Pearson correlation coefficient	0.843
Spearman rank correlation	0.899

In total, I applied five tests to the model. Three distance metrics and two correlation metrics. Table 1. contains the results of these tests. Please note that I rounded the error values to the nearest integer because the original scale resolution of relative humidity was 1 in the MET database. I rounded the correlation metrics to three significant figures to ensure balance between reported precision and readability.

In general, distance metrics quantify how far off a model's predictions are from the actual values. The average relative pressure value within the entire MET dataset was 83 hPa, so the mean absolute and root mean square errors accounted to 2.41% and 3.61% of that value respectively. I used root mean squared error instead of mean squared error because it has the same unit as the original metric, meaning it is easier to analyse. These errors were calculated using the following formulae.

Mean absolute error:

$$mae = \frac{1}{n}\sum_{i=1}^{n}|test\ value_i - \ predicted\ value_i|,$$

and mean squared error:

$$mse = \frac{1}{n}\sum_{i=1}^{n}\ (test\ value_i - \ predicted\ value_i)^2,$$

where n is the number of all samples in the dataset. Lastly, root mean squared error is a square root of the *mse* value.

The correlation metrics have shown strong correlation between the predictions and actual values. However, there are differences between the two tests. Pearson correlation:

$$correlation\ coefficient = \frac{\sum(test\ value_i - \overline{test\ value})(predicted\ value_i - \overline{predicted\ value})}{\sqrt{\sum(test\ value_i - \overline{test\ value})^2 \sum(predicted\ value_i - \overline{predicted\ value})^2}}$$

measures the linear relationship between two variables. It also assumes that the data is normally distributed. On the other hand, Spearman correlation:

$$rank\ correlation\ coefficient = 1 - \frac{6\sum d_i^2}{n(n^2-1)},$$

where *d* is the difference between two ranks of each observation and *n* is the total number of samples, measures rank-order relationships between two variables. In other words, it captures whether a relationship is increasing or decreasing. Moreover, this approach does not assume a linear relationship or a linear distribution. This makes it better suited for dealing with outliers and non-linear relations. As seen in Table 1. the Pearson correlation coefficient for my model was 0.843, indicating a strong linear relationship. The Spearman rank coefficient was 0.899, indicating a very strong correlation.

There were 6,629 rows of data that I could use while working on this model. After splitting the data 80-20 for the purposes of training and evaluating the model. In other words, 80% of the available data, or 5303 instances were used for training. The remaining 20%, or 1326 instances were utilized for the sake of evaluating the model.

2. Model

The model I developed for this project is a Multilayer Perceptron, or MLP, a type of artificial neural network. Its purpose was to predict monthly relative humidity based on historical meteorological data.

The MLP model consists of an input layer with 48 neurons, corresponding with the number of features used. What these features are, and how they were structured is described in Section 3. Features and Labels. Following the input layer, the model includes four fully connected, or Dense, hidden layers. These Dense layers consist of 256, 128, 64 and 32 neurons respectively. Each hidden layer uses the Rectified Linear Unit, or ReLU. This activation function is an important element of each layer because it introduces non-linearity. This method is efficient because it includes a comparison and a max operation. In simple terms, if the input of the ReLU function is above 0, the function returns the input. Otherwise, the output is 0. In practice, this means that the ReLU function turns off some neurons of the network, reducing the risk of overfitting.

To further prevent overfitting, I incorporated several robust methods. Each Dense layer is followed by a Dropout layer. These are a regularization technique that help against a model learning the training data too well. Dropout layers “drop out”, or set to 0, a fraction of the neurons during training. This forces the network to learn to learn more robust features.

Since there are four Dense layers in my MLP model, four Dropout layers follow. These dropped out 20%, 30%, 40% and 50% of neurons respectively. So as the hidden layers reduced in the number of neutrons, the dropped fraction increased. Even though this technique might slow down the training process, I believe this trade-off is worth it because a better model will be produced.

Additionally, I applied L2 regularization in each of the hidden layers. It is a method of influencing the loss function by controlling the relationship between fitting training data and keeping weights within the model small.

The loss function within the model was a mean squared error because it is straightforward and widely used. The final loss value obtained through training was 0.0205.

After some testing I decided to keep the regularization parameter $\lambda = 0.006$. Very small values of λ might be ineffective, as the weights in the model would not be penalized enough. Given that I incorporated other overfitting mitigation techniques, my goal was not overly penalizing weights.

The final technique that would reduce the risk of overfitting I incorporated involved an early stopper when using many epochs during training. This method halts the training process when a model's performance on a validation dataset stops improving.

The early stopper monitors validation loss. I set the "patience" parameter of the stopper to 10. So, if there is no improvement within 10 epochs, the training process stops. Besides counteracting overfitting, this approach has the ability to save time and resources. Instead of continued training, the process is stopped earlier.

The last issue that I wanted to address when it comes to the design of my MLP are imbalances in labels. Apart from normalizing the data that was used during the training process, more on which I wrote in Section 4. Preprocessing, I added a class weight adjustment to the model. This technique biases weight between the minority and majority classes. In other words, the class weights adjust the model's loss function to focus more on the minority class or classes which are less represented in the data. This encourages the model to learn equally across all classes, ideally improving its ability to correctly predict the less frequent class. For the purposes of my model, I went with a 10:1 split in favour of the minority class.

3. Features & Labels

The original MET dataset contains a wide range of variables tracked between 2015 and 2022. These were, apart from location identifying information, as follows: days of ground

frost, relative humidity, sea level pressure, vapour pressure, total rainfall, wind speed, days of lying snow, sunshine and air temperature. When preparing data for the purpose of training my model, I focused on a subset of these variables. Namely: relative humidity, sea-level pressure, air temperature and total rainfall.

The humidity values were an obvious choice as it is the target variable I aim to predict. The other three variables were chosen for two main reasons. Firstly, I wanted to reduce the number of required data to simplify the training process as well as to save on computational resources in this part of the process to be able to implement more robust methods within the model itself. Secondly, I determined that these four variables are relatively easy to track and consistently available. Making them practical choices for real-world applications.

In today's day and age, we find ourselves with an interesting problem at our hands. The development of technology has allowed us to collect more data, more information than ever before. While this is exciting and tempting, I believe that it is important to take a step back and give more consideration to the data gathering process. Rather than keeping track of all the variables we possibly can, by being more intentional in data collection, we can ensure that the data we track are directly relevant to our predictive goals, leading to better-prepared studies and more efficient data preprocessing. Instead of collecting information for the sake of collecting information, why not focus on quality over sheer quantity.

I will use the following example to explain what the features and labels are in the case of the model I have developed. If my goal is to predict the relative humidity for a given month, say April, or *hurs_4* as it is called in the MET database, then this value becomes my label. The features used for this instance are *hurs_3, psl_3, tas_3* and *rainfall_3*. Or relative humidity, sea level pressure, air temperature and total rainfall respectively. All collected in the month of March, hence the index 3. During training the model will learn the relationship between these features and the label.

There is an important point I want to bring up when it comes to how my features were designed. I structured the features as a single list. This is because I am trying to predict the next month's humidity based on the previous month's. In a regression problem like this, combining all features into one allows the model to learn how different features interact within a single time step to predict the next time step's humidity. Moreover, this approach keeps the input structure simple. Flattening the features into a single list allows the model to efficiently process them together.

4. Preprocessing

Once I decided which variables from the MET dataset to use and how to structure the features, the extraction process was quite straightforward. My first task was to successfully extract the necessary data. The original file contained 33,235 rows of data. Unfortunately, a significant portion of this data was unusable since it is composed of NaN (Not a Number) values. In other words, these are instances of missing data at the location where information was collected. After getting rid of unnecessary rows, I was left with 6,630 rows of information that could be applied for training. The columns that I used numbered 12 per variable that I chose and described in Section 3. Features & Labels. 48 in total. These columns were organized into a CSV file for further processing.

Raw data was split into features and labels. A feature comprised of variable information for a given month and a corresponding label was the relative humidity value for the following month. This resulted in a data array of shape 6,629 by 48 for the features and a list with 6629 elements for the labels.

As discussed in Section 1. Performance, this data was split into two subsets, depending on what I would use it for. 80% would be used for training and 20% would be used for testing. This split ensures that I would have enough data for the model to learn while reserving a portion for evaluating its performance on unseen data.

The final preprocessing step was normalization. I applied a MinMaxScaler to transform the data into a range between 0 and 1. Normalization ensures that features are on similar scales, improving both the efficiency and effectiveness of the learning process. This step is important, because neural network algorithms converge faster with normalized data. Lastly, I chose the MinMaxScaler for its simplicity and effectiveness in ensuring that all features contribute in an equal measure to the model.