# **INTERPRETATIONS:**

**AR:**

Before reading the interpretations, we would ask you to read through the basic understanding of the model in Time Series Machine Learning Models. If you have gone through it, you would know that AR has one hyperparameter – p which is the number of lagged data the model investigates to make a prediction. In weather data, p is assigned the value you choose as window size. You can refer to the code snippet of AR to see how these are implemented.

**Results observation**

If you look at the results, you will notice that AR performs better than other classical models at predicting temperature, worse at predicting humidity, and almost similar to the other classical models at predicting precipitation. The model is also insensitive to the window size

**Results analysis**

A possible reason for insensitivity to window size is that the optimal window size would be 365, i.e., if AR can peek into the past one year’s information, it would be easier for it to make a proper decision. However, the increase in exponential time complexity of the model with higher p values led us to choose smaller values for window which directly reflects in the prediction and the mean absolute score. AR performs relatively better with temperature due to its straightforward trend and reduced fluctuation. Whereas higher fluctuation causes AR to perform poorly at predicting humidity. For precipitation, the model just doesn’t get enough data to predict the highly fluctuating variable.

**ARIMA:**

Before reading the interpretations, we would ask you to read through the basic understanding of the model in Time Series Machine Learning Models. If you have gone through it, you would know that ARIMA has three hyperparameters – p, q, d and it is important to have the perfect combination of these three hyperparameters so that ARIMA works optimally on the data. In weather data, p and q is assigned the value you choose as window size. On the other hand, d is computed by counting the number of times differencing is done so that the correlation score is reduced below 0.05, i.e., the stationarity is removed completely. This is done by the adfuller function of statsmodel. You can refer to the code snippet of ARIMA to see how these are implemented.

**Results observation**

If you look at the results, you will notice that ARIMA doesn’t perform as good as the other models. Temperature and humidity have seasonality and ARIMA doesn’t perform well with seasonal data. On the other hand, precipitation has an interesting result in comparison to temperature and humidity. We observe that precipitation is not seasonal and is quite random. What makes this interesting is that the models don’t perform well either like ARIMA except SVR.

**Results analysis**

A possible reason for this weak performance is seasonality of the trend of temperature and thus the choice of p and q, which in this case would be optimal if chosen as 365, i.e., if ARIMA can peek into the past one year’s information, it would be easier for it to make a proper decision. However, the increase in exponential time complexity of the model with higher p and q values led us to choose smaller values for window which directly reflects in the prediction and the mean absolute score. Another important aspect is that ARIMA heavily depends on the past information and due to the data’s highly changing nature, it is important that there is a large amount of data. In this case, we don’t have enough data

In conclusion, in recent years, other machine learning techniques (e.g., Linear Regression, Random Forest, Support vector machines) and other deep learning techniques (e.g., RNN and LSTM) have replaced these classical machine learning models due to their reduced complexity and reduced need for larger dataset for similar performance. This can be seen in the mean absolute square error comparison with other models especially for humidity and temperature.

**SARIMA:**

Before reading the interpretations, we would ask you to read through the basic understanding of the model in Time Series Machine Learning Models. If you have gone through it, you would know that SARIMA has three non-seasonal hyperparameters – p, q, d and four seasonal hyperparameters – P, Q, D, m.

For our experiments, we just tested the seasonal hyperparameters i.e., leaving the non-seasonal parameters as zeros. It is important to have the perfect combination of these hyperparameters so that SARIMA works optimally on the data. In weather data, P, Q, and d are assigned the value of 1, and m defines the seasonal cycle where we want the model to look. In this case, m is assigned the value you choose as window size. For our data, the ideal value of m would be 365, so the model looks back a year to get an idea of what the weather was at this time of the year previous year. But due to SARIMA’s increased complexity, we stick to smaller m values. You can refer to the code snippet of SARIMA to see how these are implemented.

**Results observation**

If you look at the results, you will notice that SARIMA doesn’t perform as good as the other models. Although SARIMA performs well with seasonal data, in this it doesn’t perform well with temperature and humidity. On the other hand, precipitation has an interesting result in comparison to temperature and humidity. We observe that precipitation is not seasonal and is quite random. What makes this interesting is that the models don’t perform well either like SARIMA except SVR.

**Results analysis**

A possible reason for this is the choice of m, which in this case would be optimal if chosen as 365, i.e., if SARIMA can peek into the past one year’s information, it would be easier for it to make a proper decision. However, the increase in exponential time complexity of the model with higher m values led us to choose smaller values for window which directly reflects in the prediction and the mean absolute score. Another important aspect is that SARIMA heavily depends on the past information and due to the data’s highly changing nature, it is important that there is a large amount of data. In this case, we don’t have enough data.

In conclusion, in recent years, other machine learning techniques (e.g., Linear Regression, Random Forest, Support vector machines) and other deep learning techniques (e.g., RNN and LSTM) have replaced these classical machine learning models due to their reduced complexity and reduced need for larger dataset for similar performance. This can be seen in the mean absolute square error comparison with other models especially for humidity and temperature.

**RNN**

Before reading the interpretations, we would ask you to read through the basic understanding of the model in Time Series Machine Learning Models. If you have gone through it, you would know that Recurrent Neural Networks are deep learning models usually used to solve problems with sequential data such as time series. What helps RNN to work with time series data is its capability to retain a memory. Neural networks usually consist of layers, for RNN specifically, we use SimpleRNN layer of Keras to design our architecture of the model. The two main hyperparameters of an RNN architecture are the number of layers and the number of neurons. Unlike classical models, where we use window to define one of the hyperparameters, for RNN, the window size is used to rearrange the input of the model. It is important to note that the same architecture (i.e., number of layers and number of neurons) doesn’t necessarily work for all inputs and it needs to be tweaked and updated for different scenarios. In our example, we used two different architectures for predicting temperature, humidity, and precipitation. While one architecture worked well with temperature and humidity, we had to redesign for precipitation. Moreover, neural networks can take in more than one features, which weren’t implemented for classical models due to their increased complexity. You can check out the predicted trends for when only the target sequence is used to predict and also when the sequence of entire feature set is used to predict. You can refer to the code snippet of RNN to see how these are implemented.

**Results observation**

If you look at the results, you will notice that RNN has different performance based on the input and output of the model. For temperature, we observe that as we decrease the window size, the performance of model increases. Similarly for humidity, we observe that increasing the window size causes the model to performance adversely. However, adding more features for the prediction of humidity causes the performance to be more stable and insensitive to window size. While it is obvious from the performance comparisons that RNN performs far better than classical models (AR, ARIMA, SARIMA), it is interesting that performance of RNN model isn’t much better than the classical models for prediction of precipitation irrespective of the window size and the features.

**Results analysis**

A possible intuitive reason for the increased performance with decreasing window size in predicting temperature and humidity is that to know the temperature of today, knowing just the temperature of the past week is good enough. Information of the entire month’s temperature doesn’t help much but it shouldn’t reduce the performance, right? Unfortunately, giving the model more information than it requires causes it to overfit, i.e., it forces the model to memorize the data it was trained with, to the point that they can’t think intuitively. Similarly, when we add more features, prediction of temperature doesn’t require that much information leading to overfitting and adversely affecting the performance. However, when we investigate humidity, we see that adding more features gives a more stable performance with respect to window. Intuitively, this means other features help determining the humidity. For example, if it rained on a specific day and it’s not very windy, it is very likely it could be a humid day. Adding more feature helps the model to learn not just the how humidity changes but how other features affects humidity making it more stable in its prediction. Regarding precipitation, RNN doesn’t perform too well with precipitation like other models. This is due to the high frequency of fluctuations in the trend of precipitation, which is quite difficult for the model to differentiate from outliers or anomalies, and it requires more effort in optimizing the model or more appropriate features. In conclusion, it is quite important to find the optimum information and architecture for a specific problem.

**LSTM**

Before reading the interpretations, we would ask you to read through the basic understanding of the model in Time Series Machine Learning Models. If you have gone through it, you would know that Long Short-term memory is a type of recurrent neural networks usually used to solve problems with sequential data such as time series. LSTM is different from RNN in the sense that it consists of extra units that manages the information in memory such as when it enters the memory, how long and how much information may be kept, when it begins to provide output, and when it begins to decay or be forgotten. This gives LSTM an advantage over RNN. Neural networks usually consist of layers, for LSTM specifically, we use LSTM layer of Keras to design our architecture of the model. The two main hyperparameters of an LSTM architecture are the number of layers and the number of neurons. Unlike classical models, where we use window to define one of the hyperparameters, for LSTM, the window size is used to rearrange the input of the model. It is important to note that the same architecture (i.e., number of layers and number of neurons) doesn’t necessarily work for all inputs and it needs to be tweaked and updated for different scenarios. In our example, we used two different architectures for predicting temperature, humidity, and precipitation. While one architecture worked well with temperature and humidity, we had to redesign for precipitation. Moreover, neural networks can take in more than one features, which weren’t implemented for classical models due to their increased complexity. You can check out the predicted trends for when only the target sequence is used to predict and also when the sequence of entire feature set is used to predict. You can refer to the code snippet of LSTM to see how these are implemented.

**Results observation**

If you look at the results, you will notice that LSTM has different performance based on the input and output of the model. Compared to RNN, the performance of LSTM is far more stable when just the sequence of the prediction target only. However, the performance becomes more sensitive to window size when more features are used. On the other hand, while it is obvious from the performance comparisons that LSTM performs far better than classical models (AR, ARIMA, SARIMA), it is interesting that performance of LSTM model isn’t much better than the classical models for prediction of precipitation irrespective of the window size and the features.

**Results analysis**

A possible intuitive reason for the stability in performance with changes in window size is that the extra units help the model to forget the information it doesn’t need in the past, whereas RNN is more susceptible to changes in window, mostly getting overfit with an increase in window size. On the other hand, adding more features causes the LSTM models to become more susceptible to changes in window size. This is because the architecture isn’t optimal for the input with added features. Regarding precipitation, LSTM doesn’t perform too well with precipitation like other models. This is due to the high frequency of fluctuations in the trend of precipitation, which is quite difficult for the model to differentiate from outliers or anomalies, and it requires more effort in optimizing the model or more appropriate features. In conclusion, it is quite important to find the optimum information and architecture for a specific problem.