Interview Results

Data Augmentation, Data Imputation, and Correlation

The code in the file with the name data_cleaning.py is used to clean the dataset. Initially, I visually inspected the data to identify inconsistencies and understand the type of each column in the dataset. Additionally, I printed a few rows, the data types for each column, and the statistics. Subsequently, I checked the dataset for null entries and attempted to fill them using a KNN Imputer for appropriate answers. If the imputation failed due to many consecutive rows containing missing values, I removed them. Another solution I considered was checking the column descriptions for keywords like 'rain' or 'snow' and using them to populate the 'Precip Type' column.

The function data_imputation uses KNN data imputation. The main principle of this algorithm is to leverage the information from neighboring data points surrounding the missing values to infer and impute the missing data. Adding -1 to every NaN value to remove it later.

The function remove_outliers finds 1st quantile and 3rd quantile of each numerical column. Afterwards, it calculates the IQR and the upper/lower bound based on a threshold. If the value is lower than the lower bound or higher than the upper bound the row is removed.

The function visualize_correlation is used to visualize the distribution and correlation among the dataset's features. Upon examining the correlation matrix, it becomes clear which features hold greater significance, and which class serves as our label. At then end I create a file with the name Correlation to save the images that are being shown.

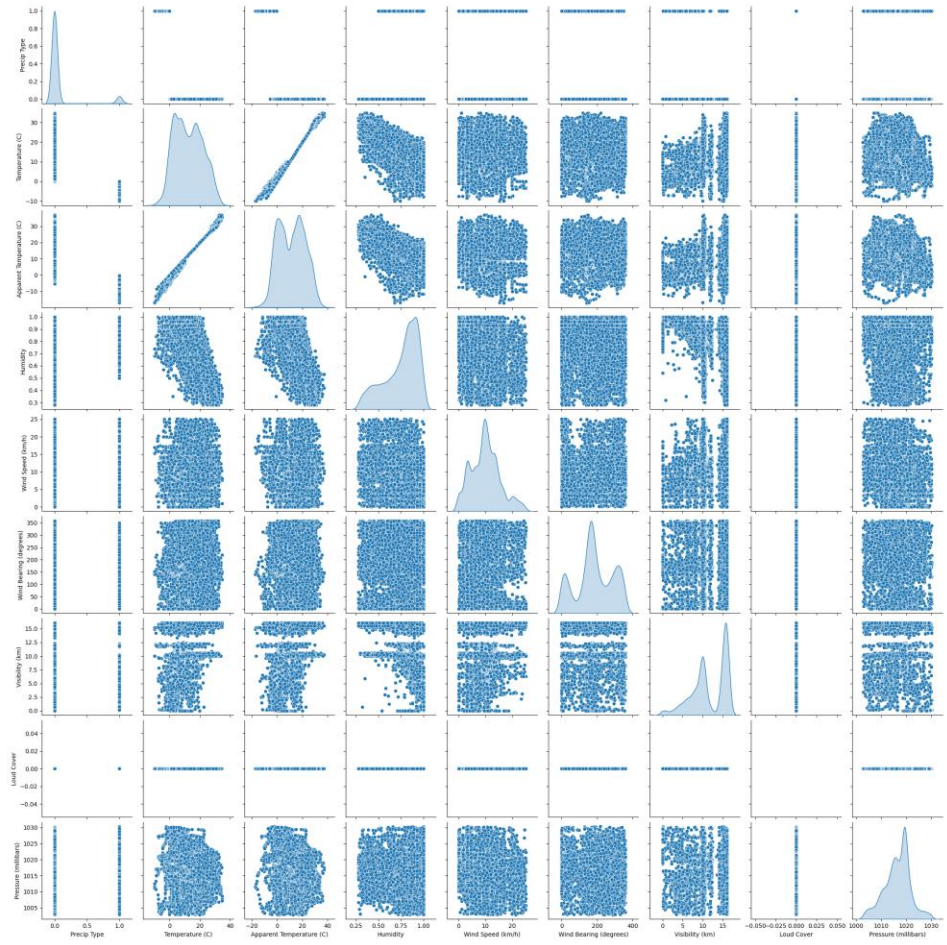

Figure 1 Distributions

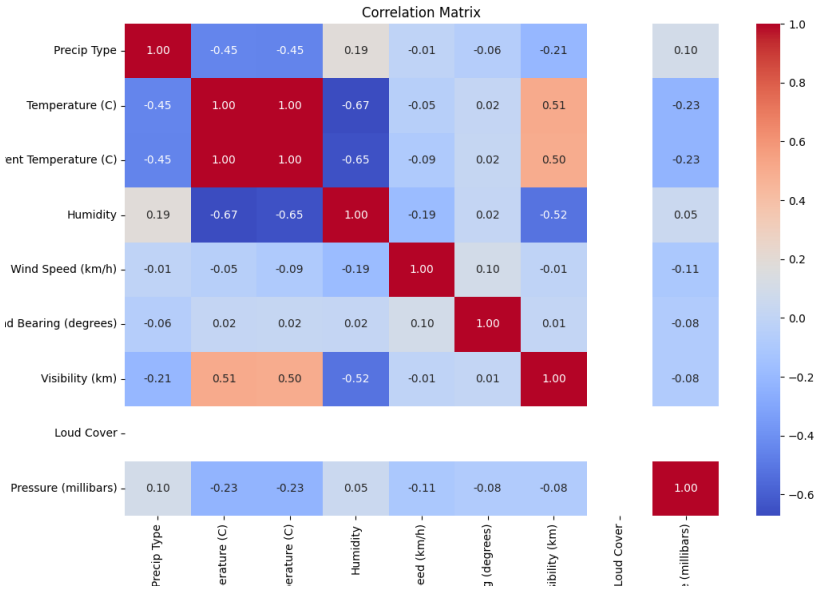

Figure 2 Correlation

Feature Importance

In the file feature_importance.py I created a function to get all the necessary information on the importance of each column. After reviewing the correlation matrix, I proceeded with a more in-depth analysis using classifiers to determine feature importance. Subsequently, I removed columns containing text and determined the total number of classes. The use of different algorithms yielded two distinct sets of results. Both sets will be considered to determine which features yield the optimal model performance. Normalize each column of the features and get extracted features.

Get Hyper-Parameters

For the model I decided on using a Long-Short Memory Model (LSTM). It is a really common neural model for Time-Series data. A few alternative algorithms that could be used are Autoregressive Integrated Moving Average (ARIMA) and Seasonal Autoregressive Integrated Moving-Average (SARIMA). All of the have their pros and cons, but LSTM with our current size was a more optimal choice in my opinion, even if it is more computationally expensive. For the hyperparameters of the model, I used the tuner to try and test different values to get the optimal hyper-parameters. The tuner only has to run one time, giving us the result and saving them to a directory to be re-used. This can be found in the tuner.py file.

The Model

After getting the best model predicted by the tuner, we train it for 100 epochs (I believe that around 50 to 100 would be a great selection) and plot the training history of the model. We can also load the model for further use and training.

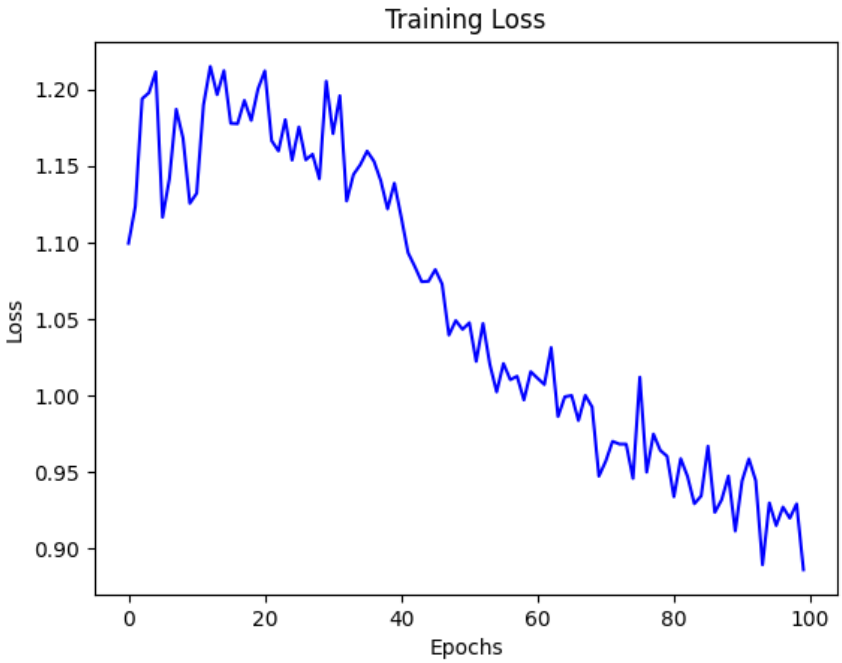

Figure 3 Loss

For the Data Generation

I've designed the following function to iterate through each column. If a column contains numerical data, the function generates new synthetic data. It calculates the mean and standard deviation of each numerical column and generates a new value using a Gaussian distribution centered around the calculated mean with the standard deviation as the spread. These synthetic data points are then added to a new row for prediction.

Running the Code

By running the main.py all the features mentioned above will run after importing all the functions from each .py file. The data is normalized.

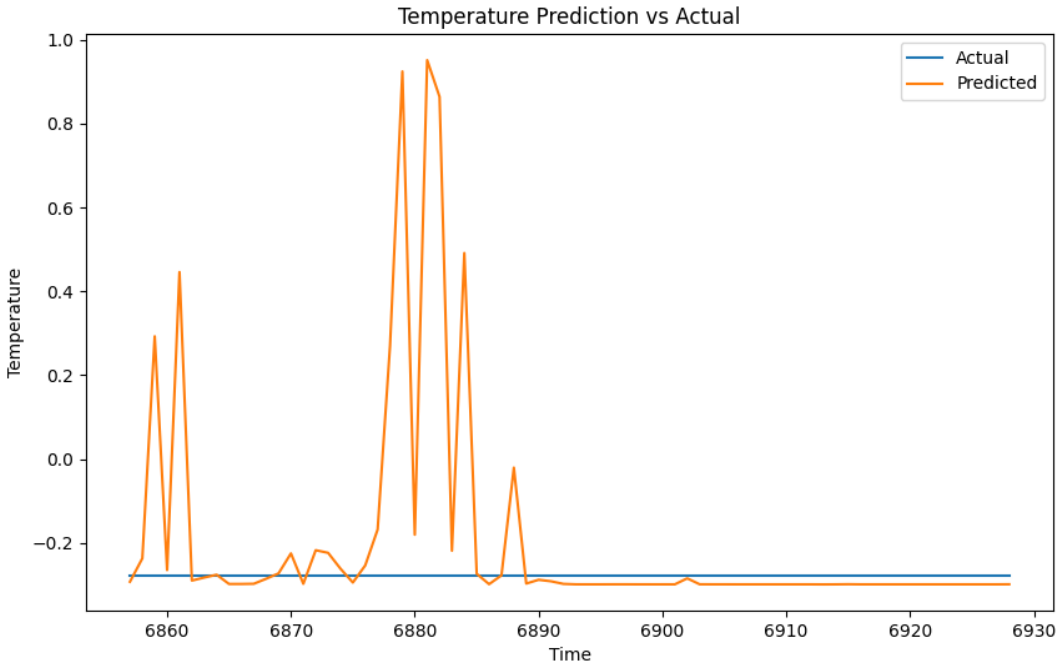

Figure 4 Predictions

Basic API

The database_flask.py is a simple API call and database creation for keeping and retrieving records. The database table has all the features deemed important from the Recursive Feature Elimination. The GET method retrieves all the data from the database as a JSON file. Finally, the POST method is for adding new data to the table.

```
curl -X POST http://127.0.0.1:5000/forecast   -H "Content-Type: application/json"   -d '{"formatted_date": "2024-03-31 13:00:00", "temperature": 28.0, "apparent_temperature": 28.0, "humidity": 0.65, "visibility": 10.0, "pressure": 1010.0, "prediction": "Sunny"}'
```

```
curl http://127.0.0.1:5000/forecast
```

To add another entry to the database, the formatted_date column needs to have a new time or day, because it is the primary key.