AI Weather Forecasting

COMP8047-Major Project

Jason Zhou – A01005069
10-29-2023

Table of Contents

1. Introduction

1.1. Student Background

My name is Jason Zhou. I am a CST graduate and also a second-year student in the B-TECH Network Security Option. I have a strong interest in cybersecurity and App development. I have built a few apps using Android studio and Google APIs. I am familiar with Java, C++, Python, NodeJS, HTML, SQL and I am always passionate to learn new technologies.

1.2. Project Description

The project idea is to use Artificial Intelligence to create an application that forecasts weather and nature disasters. Weather plays an important role in everyone's life as it can help us to know when it will be rainy and when it will be sunny. Knowing the weather ahead can help us plan our trips or decide our outfits more easily. Meanwhile, weather predictions can help improve transportation safety, help farmers reduce losses from extreme weather and inform people to evacuate beforehand to prevent deaths and injuries. The reason to use machine learning to predict weather and is because it provides faster and more accurate predictions, not only improves weather forecasting but also bring convenience to people.

1.2.1. Essential Problems

Traditional weather forecasting has been used for centuries to predict the weather. However, it has several limitations. One of the main problems with traditional weather forecasting is that it relies on human forecasters to interpret data and make predictions. This can lead to errors and inconsistencies in the forecasts, especially for complex weather patterns.

AI weather forecasting, on the other hand, uses machine learning algorithms to analyze vast amounts of data and make predictions. This approach has several advantages over traditional forecasting, including greater accuracy, faster processing times, and the ability to handle complex patterns in the data. For instance, AI tools can interact directly with data sets without a human operator's input, making the predictions more reliable and efficient.

1.2.2. Goals and Objectives

The goal of this project is to develop an AI weather forecasting model that can provide fast and accurate predictions, as well as learning more modeling techniques and other advanced techniques. The objective of this project will be delivered through a python application. The application will ask user to input a location, and returns future weather forecasting for that location.

2. Body

2.1. Background

Weather forecasting plays an important role in day-to-day lives, it can help people decide when to carry an umbrella or choose what outfit to wear. It also improves transportation safety, and it is beneficial to farmers. Currently, the most commonly used weather forecasting technology relies on the combination of computer calculation and human expert analysis. However, this type of forecast

is not always accurate. There were several times the weather forecast said it was sunny and it turned out to be rainy, and because of that I got a cold from getting wet. Moreover, inaccurate predictions can lead to more serious consequences. It involves scenarios in which natural catastrophes occur in places where people haven't received proper warning, causing deaths or injuries that could have been prevented.

Therefore, I always wondering why is weather so hard to predict. After researches, I found out that the ability for meteorologists to predict the weather is limited by three factors [1]: the amount of available data; the time available to analyze it; and the complexity of weather events. However, Artificial intelligence can solve all these problems. First, AI is good at dealing with large amount of data and analyzing complex events in a very short amount of time. Even if the amount of data is narrow or partially missing, with an appropriate algorithm, AI model can still find patterns among the data. Therefore, a well-trained machine learning model can provide a more faster and accurate weather prediction than meteorologists.

2.2. Project Statement

The project will be a demonstration of the capabilities of using Random Forest (R.F.) classification and regression model for time-series weather forecasting. The trained AI models will be loaded and make prediction through a python application. Once the user inputs a place, the application provides future weather forecasting for that specific location.

2.3. Possible Alternative Solutions

Currently there are a few different solutions that I could have used during this project instead of what I went with. Some of them are

Time Series Forecasting with Neural Prophet- NeuralProphet is a Python library for modeling time series data based on neural networks. It is built on top of PyTorch and combines neural network and traditional time series algorithms, inspired by Facebook Prophet and AR-Net. The advantage of NeuralProphet is It can handle seasonality at different time scales, such as yearly, daily, weekly, and hourly, using Fourier terms. However, it may not be able to handle complex non-linear relationships or interactions among the forecast components. NeuralProphet is a relatively new framework for time series forecasting, and it does not used by any mainstream forecasting solutions yet.

Time series Forecasting with LSTM Recurrent Neural Networks- Neural networks like Long Short-Term Memory (LSTM) recurrent neural networks are able to almost seamlessly model problems with multiple input variables. This is a great benefit in time series forecasting, where classical linear methods can be difficult to adapt to multivariate or multiple input forecasting problems. LSTM is not a new algorithm, but I don’t see it widely used by the mainstream weather agencies.

Time series Forecasting with GraphCast- This is a machine learning model developed by Google DeepMind that uses a graph neural network to learn from historical weather data and predict global weather conditions up to 10 days in advance. It outperformed conventional methods in 90 percent of 1,380 metrics, such as temperature, pressure, wind speed and direction, and humidity. The precision of this system surpasses the industry-recognized benchmarks of the

High-Resolution Weather Modeling System (HRES) of the European Center for Medium-Range Weather Forecasts (ECMWF). GraphCast is used by several weather agencies, including ECMWF.

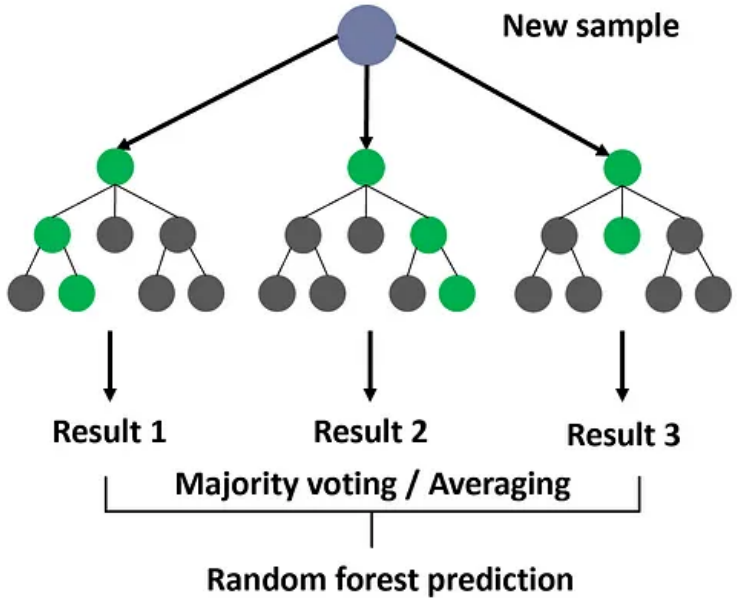
2.4. Chosen Solution

I will build my weather forecasting application with Random Forest model. The main reason to use Random Forest algorithm is because it can capture complex temporal dependencies in the data and handle a wide range of time series patterns, including trends, seasonality, and cycles. Another reason to use random forest is because it does not require any knowledge about neural network and deep learning, as I don't have a good laptop GPU for handle deep learning problems. Also, use an algorithm that is not designed for weather forecasting would increase project's complexity.

Random Forest is a machine learning ensemble model that consists of multiple decision trees, it can be used for both classification and regression tasks. Bagging, also known as Bootstrap Aggregation, serves as the ensemble technique in the Random Forest algorithm. In bagging, a group of models is trained on different subsets of the dataset, and the final output is generated by collating the outputs of all the different models. In the case of random forest, the base model is a decision tree.

Random Forest Algorithm follows these steps

- Step 1: In the Random Forest model, a subset of data points and a subset of features is selected for constructing each decision tree. Simply put, n random records and m features are taken from the data set having k number of records.
- Step 2: Individual decision trees are constructed for each sample.
- Step 3: Each decision tree will generate an output.
- Step 4: Final output is considered based on Majority Voting or Averaging for Classification and regression, respectively.



In the context of forecasting time series weather data, the choice between a random forest regression model and a random forest classification model is guided by the nature of the target variables.

For numerical target variables such as temperature, humidity, and pressure, a random forest regression model is deemed more suitable. Regression models are adept at predicting continuous values and can effectively capture the underlying patterns in the temporal structure of time series data. Given that each observation is dependent on previous ones in time series data, a regression model aligns well with the continuous and interconnected nature of numerical weather features.

Conversely, for label and categorical target variables like weather, and weather description, employing a random forest classification model proves advantageous. Classification models are designed to handle discrete categories or classes, making them well-suited for predicting categorical weather conditions.

2.5. Details of Design and Development

2.5.1. Feasibility Assessment Report

Project Overview:

The time series weather forecasting project focuses on predicting future weather values using various techniques, with a primary emphasis on Random Forest classification/regression, walk-forward validation, and sliding window representation. These methodologies aim to enhance the accuracy and adaptability of the forecasting models, particularly in the context of time series data.

Feasibility Assessment:

Technical Feasibility:

- Algorithmic Suitability: Random Forest regression has proven to be a robust choice for both classification and regression tasks, including time series forecasting. Its versatility allows the project to address a wide range of predictive modeling problems effectively.
- Implementation Technologies: The project leverages established machine learning libraries and frameworks for Random Forest, with basic python coding and API invocation, ensuring technical feasibility and ease of integration.

Methodological Feasibility:

- Walk-Forward Validation: The utilization of walk-forward validation enhances the model's adaptability by updating with new data continuously. This method provides a realistic evaluation of performance on unseen data, contributing to the project's methodological feasibility.
- Sliding Window Representation: The transformation of time series data into a supervised learning problem using sliding window representation aligns with established practices, offering a feasible approach for feature engineering.

Data Feasibility:

- Availability and Quality: The project uses historical time series data from a well-known weather service platform openWeatherMap for training and evaluation. Ensuring the quality and consistency of the dataset is crucial for the success of the forecasting models.

Resource Feasibility:

- Computational Resources: Random Forest algorithm, walk-forward validation, and sliding window representation do not require extensive computational resources. The project's resource requirements align with standard machine learning practices.
- Skill Requirements: Adequate expertise in machine learning, specifically in Random Forest regression and time series forecasting, is essential. The project team's existing skills and knowledge contribute positively to resource feasibility.

Conclusion:

The feasibility report suggests that the project is technically, methodologically, and resource-wise feasible. Leveraging Random Forest regression, walk-forward validation, and sliding window representation enhances the project's capability to handle time series forecasting effectively. With the appropriate data quality and skilled personnel, the project is well-positioned for successful implementation.

2.5.2. Model API use

In this project, the random forest model will be developed using the sklearn's API.

One of the reasons to choose sklearn over other libraries for random forest regression is that sklearn offers a comprehensive documentation, a rich set of tools for data preprocessing, model selection, evaluation, and visualization, and a large community of users and developers. Another reason is that scikit-learn works well on CPUs, and it does not require a powerful GPU. Sklearn is also compatible with other Python libraries, such as numpy, scipy, pandas, and matplotlib.

The sklearn random forest classification and regression model is implemented by the RandomForestClassifier and RandomForestRegressor class in the sklearn.ensemble module. This class inherits from the BaseForest class, which is a base class for forest of trees-based estimators. The parameters of the sklearn random forest classifier and regressor are mostly the same, except for the criterion parameter, which specifies the function to measure the quality of a split. For the classifier, the criterion can be either “gini” or “entropy”, which are two ways of calculating the impurity or uncertainty of a node. For the regressor, the criterion can be either “squared_error”, “absolute_error”, “friedman_mse”, or “poisson”, which are different ways of calculating the error or deviation of a node.

Some of the common parameters of the sklearn random forest classifier and regressor are:

- n_estimators: The number of trees in the forest. A larger number of trees may improve the accuracy and reduce the variance, but also increase the computation time and memory usage.

- max_depth: The maximum depth of the tree. A deeper tree may capture more complex patterns, but also increase the risk of overfitting and reduce the interpretability.
- min_samples_split: The minimum number of samples required to split an internal node. A smaller value may create more splits and increase the complexity, but also reduce the bias and increase the variance.
- min_samples_leaf: The minimum number of samples required to be at a leaf node. A larger value may create fewer leaves and reduce the complexity, but also increase the bias and reduce the variance.
- max_features: The number of features to consider when looking for the best split. A smaller value may reduce the correlation among the trees, but also increase the bias and reduce the diversity.
- bootstrap: Whether to use bootstrap samples when building the trees. Bootstrap samples are random samples with replacement from the original data. Using bootstrap samples may reduce the variance and improve the accuracy, but also introduce some bias and reduce the diversity.

2.5.3. Data collection and processing

In order to build an effective time series weather forecasting model, you need to carefully select the dataset. A good dataset should contain temporal features like data and time, and meteorological Features like temperature, humidity, pressure and wind speed. After weeks of research, I found out that the openWeatherMap API offers a comprehensive history dataset encompassing all the desired features. The next step is to process the dataset into the desired format.

Data Processing is one of the most important steps in model developing, especially for time series forecasting. The json list below is the weather data requested from the openWeatherMap API:


```
{"message":"Count:1","cod":"200","city_id":6173331,"calctime":0.0028231
71,"cnt":1,"list":[{"dt":1698386400,"main":{"temp":37.65,"feels_like":3
3.13,"pressure":1027,"humidity":86,"temp_min":34.29,"temp_max":41.23},"
wind":{"speed":5.75,"deg":70},"clouds":{"all":40},"weather":[{"id":802,
"main":"Clouds","description":"scattered clouds","icon":"03n"}]}]}
```

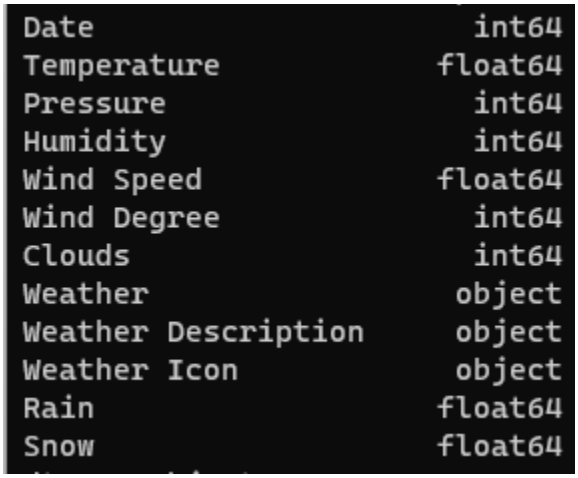

The json object contains hourly weather data, each count equals to one hour. The complete list contains multiple pairs of key value pairs, to let the data become readable, we need to convert it into a csv file. First, we initialize a list to store the weather data,

```
weather_data_list = []
data = response.json()
weather_data_list.extend(data['list'])
```

Next, define the CSV headers by iteratively adding data to the header columns by choosing features that appear to be related to weather forecasting. Since the openWeatherMap dataset is well-organized and comprehensive, I won't need to spend a lot of time thoroughly cleaning the data.

Up to seven days of historical data can be requested via the openWeatherAPI per time, however this is obviously insufficient for testing and training models. After repeating this step 52 times, I was able to gather a dataset consisting of 8762 lines. This dataset contains Vancouver's weather data from November 1, 2022, to November 1, 2023, covering a year's worth of information. Additionally, the validation set includes weather data from December 2, 2022, to December 3, 2023, but this time from Seattle.

The next step is find out each feature's datatype, by typing `print(df.dtypes)`, we get the datatypes for all columns.



As you can see, the weather feature's data type is object. However, random forest does not accept object input, we need to import a label encoder to convert object input into numerical values.

```
le_weather = LabelEncoder()
df['Weather'] = le_weather.fit_transform(df['Weather'])
```

Now all the data has been prepared, we can proceed with the training process.

2.5.4.Validation Method

In order to implement time-series forecasting tasks, we must choose an appropriate validation method. Random Forest can be utilized for time series forecasting by transforming the time series dataset into a supervised learning problem. It also requires the use of a specialized technique for evaluating the model called walk-forward validation, as the use of traditional cross validation techniques like k-fold cross validation for evaluating the model would lead to optimistically biased outcomes, because they do not account for the temporal aspect.

Walk-forward validation is a technique that involves training a model on historical data up to a certain point in time, making predictions for the next period, and then updating the model with the actual value for that period. This process is repeated for each subsequent period in the time series, with the model being updated at each step. Walk-forward validation can help to ensure that the model is robust and can generalize well to new data.

Another technique that can be used alongside walk-forward validation is sliding window which involves using a fixed-size window of historical data to make predictions for the next period. The window is moved forward one step at a time, and the process is repeated for each subsequent period in the time series. Sliding window can help to capture trends and patterns in the data and make accurate predictions Here is an example of implementing sliding window:

```
df = list(range(1, 51))  # Create a list with numbers from 1 to 50
for i in range(7, len(df)):
    x = df[i-7:i]
    print(f"i={i}, x={x}")
```

In this loop, x is a subsequence of df that contains the last 7 elements, starting from index i-7 up to index i-1. Let's see the values of x after each iteration:

1. i=7, x=[1, 2, 3, 4, 5, 6, 7]
2. i=8, x=[2, 3, 4, 5, 6, 7, 8]
3. i=9, x=[3, 4, 5, 6, 7, 8, 9]
4. i=10, x=[4, 5, 6, 7, 8, 9, 10]

   …

   …

42. i=48, x=[42, 43, 44, 45, 46, 47, 48]
43. i=49, x=[43, 44, 45, 46, 47, 48, 49]
44. i=50, x=[44, 45, 46, 47, 48, 49, 50]

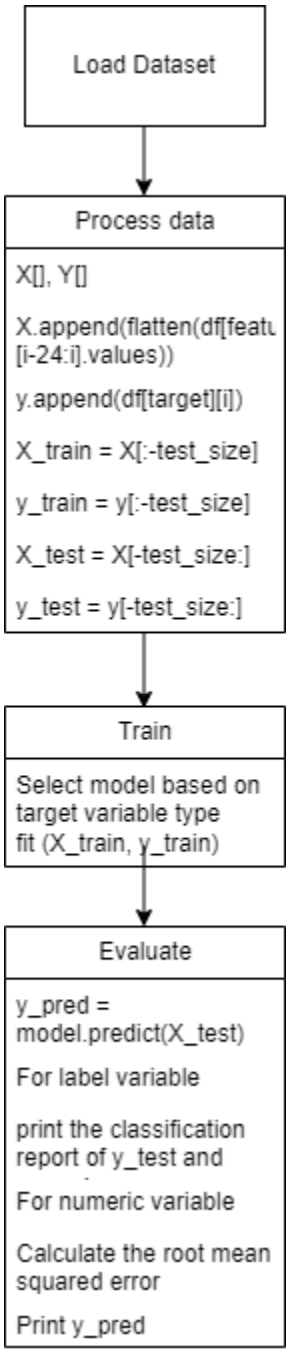
During each iteration, a feature list is generated with a specific window size, and subsequently advanced by one element. Utilizing the sliding window technique, we can effortlessly obtain a collection of input features. This proves to be highly advantageous for models as it enables them to be trained on historical data and identify patterns effectively.

2.5.5. Training and Testing

Now we can start to write a function to train and evaluate the random forest model. The pseudo code list below describes how the function works.

```
# Import the dataset
df = read_csv('vancouver_weather.csv')
# Define a function to train and evaluate a random forest model
function train_and_evaluate(df, features, target):
# Initialize empty lists for input features and target values
  X = []
  y = []
# Loop through the data from index 24 to the end
  for i in range(24, len(df)):
# Append the values of the previous 24 observations as a feature vector
```

```
    X.append(flatten(df[features][i-24:i].values))
# Append the value of the current observation as a target value
    y.append(df[target][i])
# Convert the lists to arrays
  X = array(X)
  y = array(y)
# Define the size of the test set
  test_size = 2400
# Split the data into training and testing sets follows temporal order
  X_train = X[:-test_size]
  y_train = y[:-test_size]
  X_test = X[-test_size:]
  y_test = y[-test_size:]
# Choose the model based on the target variable type
  if target.lower() == 'weather':
    # Classification case
    model=RandomForestClassifier(n_estimators=100, min_samples_split=2)
  else:
    # Regression case
    model=RandomForestRegressor(n_estimators=100, min_samples_split=2)
# Train the model with the training set
  model.fit(X_train, y_train)
# Make predictions with the model on the test set
  y_pred = model.predict(X_test)
  print 'Testing results:'
# Evaluate the model performance based on the target variable
  if target.lower() == 'weather':
    # Classification case
    print the classification report of y_test and y_pred
  else:
    # Regression case
    # Calculate the root mean squared error
    rmse = square root of the mean squared error of y_test and y_pred
    print 'Root Mean Squared Error:', rmse
# End of function
```



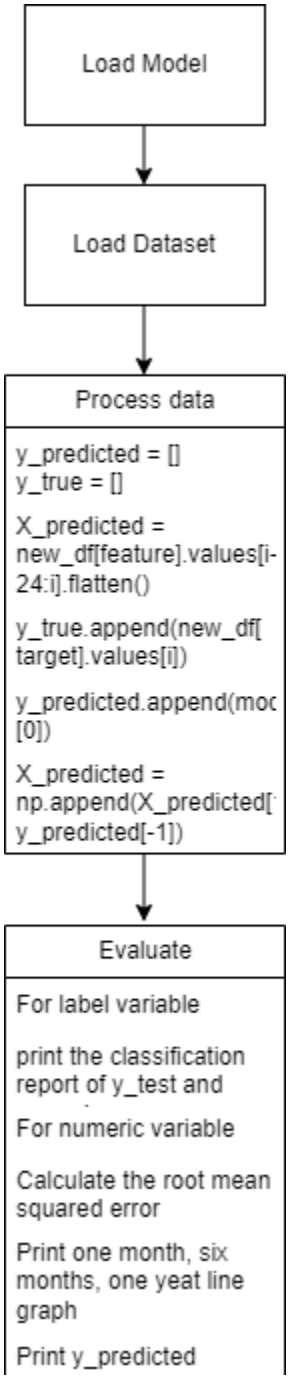
First, we load the dataset. Then we create list X[], y[] and using sliding window to make X[] a list of feature vectors, as well as append target variable to the y[]. Next we divide the X[] and y[] into train and test list, and fit X_train, y_train to a R.F. classification or a regression model based on the target type. Additionally, we evaluate the model's performance based on whether it is a classification or regression model. To measure the accuracy of a classification model, we use a classification report, while for a regression model, we use RMSE score. Finally, the predicted result is printed by the function.

2.5.6. Model Validating

The validation process is similar to the testing process. Initially, a fresh dataset with unseen data is imported. Subsequently, the data is processed utilizing the sliding window technique to generate a collection of input features. Lastly, the model is employed to make predictions based on these features. By continuously updating the predicted outcome with the input set through sliding window, the model has the capability to continuous forecast twenty-four hours weather conditions or even longer. Finally, we measure the model performance based on whether it is a classification or regression model.

```
# Import the test dataset
new_df = read_csv('testCsv.csv')
# Use the first 24 rows of the data as input to predict next 24 hours weather
values = new_df[features].values
row = flatten(values[:24])
X_predicted = row
y_true = values[24:48,-1]
y_predicted = []
# Loop for 24 times
for i in range(24):
  # Append the predicted value to y_predicted
  y_predicted.append(model.predict([X_predicted])[0])
  # Update X_predicted by removing the first element and adding the last predicted value
  X_predicted = append(X_predicted[1:], y_predicted[-1])
# Convert y_predicted to an array
y_predicted = array(y_predicted)
# Evaluate the model performance based on the target variable
if target.lower() == 'weather':
  # Classification case
  print the classification report of y_true and y_predicted
else:
  # Regression case
  # Calculate the root mean squared error
  rmse= square root of the mean squared error of y_true and y_predicted
  print f'RMSE for {target} predictions: {rmse:.2f}'
  # Create a line plot of the predicted values and the actual values
  Plot 3 lines, y_predicted and y_true
  label the x-axis as 'Date'
  label the y-axis as {target}
  add a title f'{target} Predictions vs Actual Values'
```

```
  show the plot
# Print the predicted weather for next 24 hours
print f'Predicted {target} for Next 24 hours:', y_predicted
```

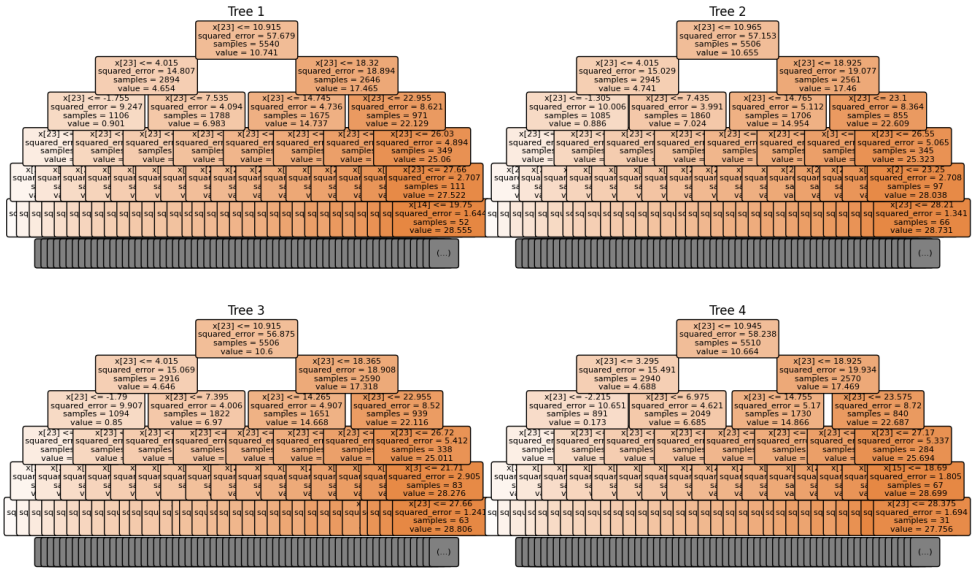


2.5.7.Decision trees

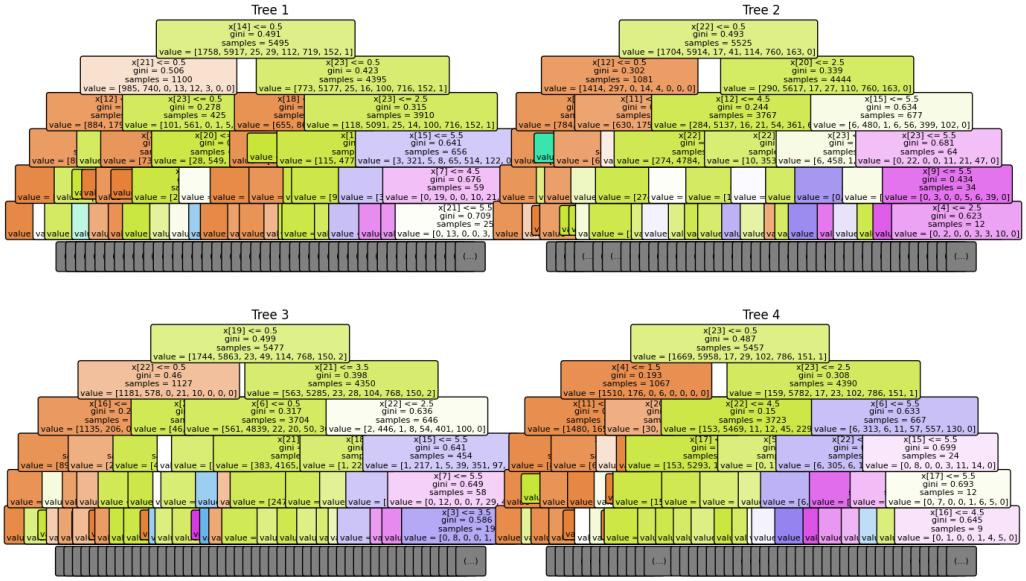
As I mentioned in section 2.4, Random Forest is a machine learning ensemble model that consists of multiple decision trees. In this section, I will show the decision trees of each model.

In regard to the readability, each plot's max_depth is set to 5, and it only shows the first four trees out of the hundred trees of each model.

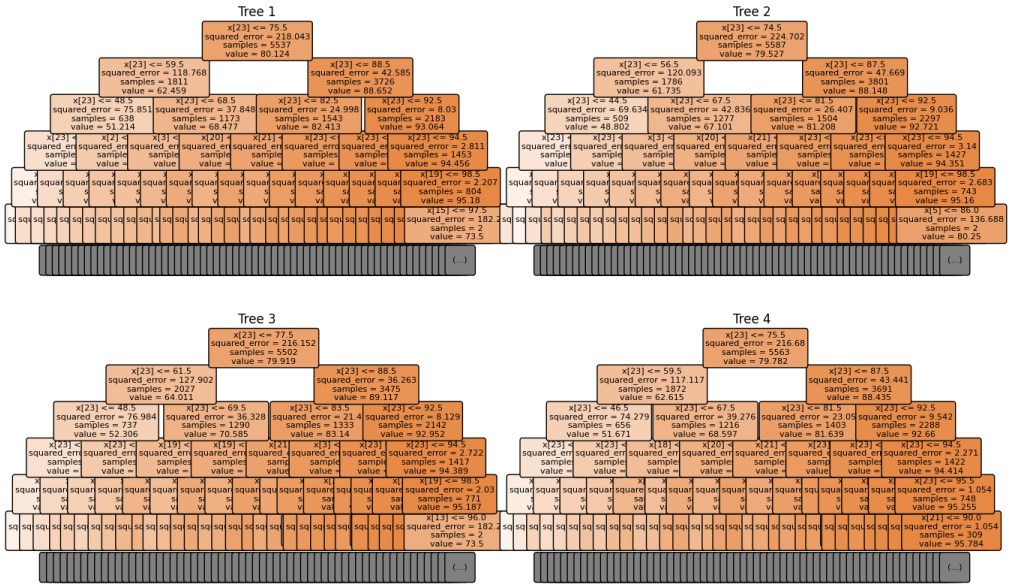
Temperature model



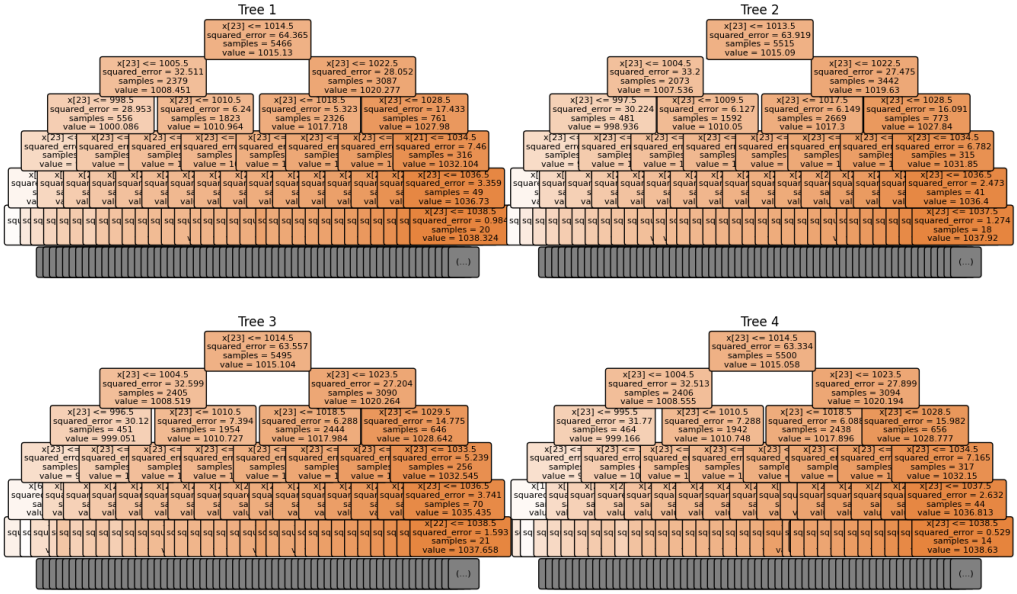
Weather model



Humidity model



Pressure model



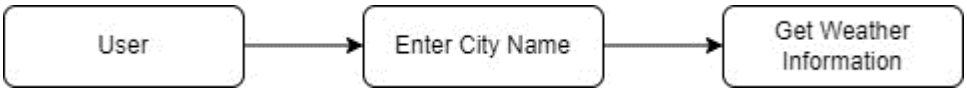
In decision trees, the root node is the node with the most predictive power, which means that it reduces the impurity or uncertainty of the data the most. In my decision trees, majority of the root node start with feature x[23], it is because I am creating a sliding window of size 24 for each row of the data frame. This means that each row of the input matrix X will contain 24 values of the feature, corresponding to the previous 24 hours. The x[23] is the name of the last value in each row, which is the weather/temperature/humidity/pressure of the hour before the target hour.

As the decision tree progresses, each internal node further refines the decision process by identifying splits based on features that optimize the reduction in impurity. These splits are determined by evaluating conditions on features, effectively partitioning the data into subsets that enhance the homogeneity of observations within each branch.

Throughout the tree's structure, the terminal nodes, or leaves, represent the final predictive outcomes. For each leaf, predictions are made based on the majority class (in classification tasks) or the average value (in regression tasks) of the instances falling within that specific branch.

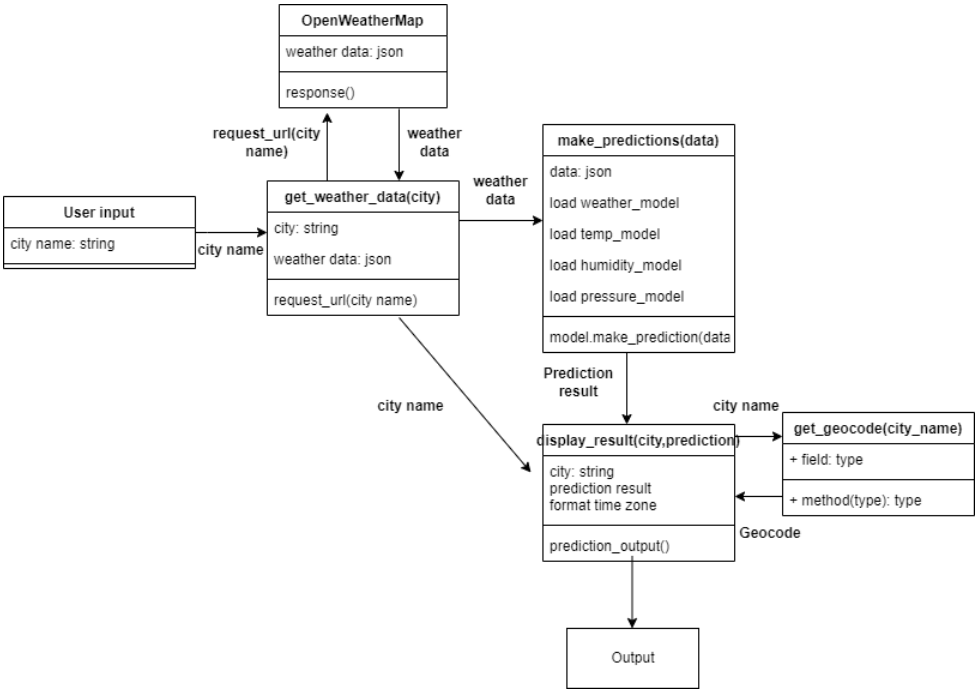
2.5.8. Application Design

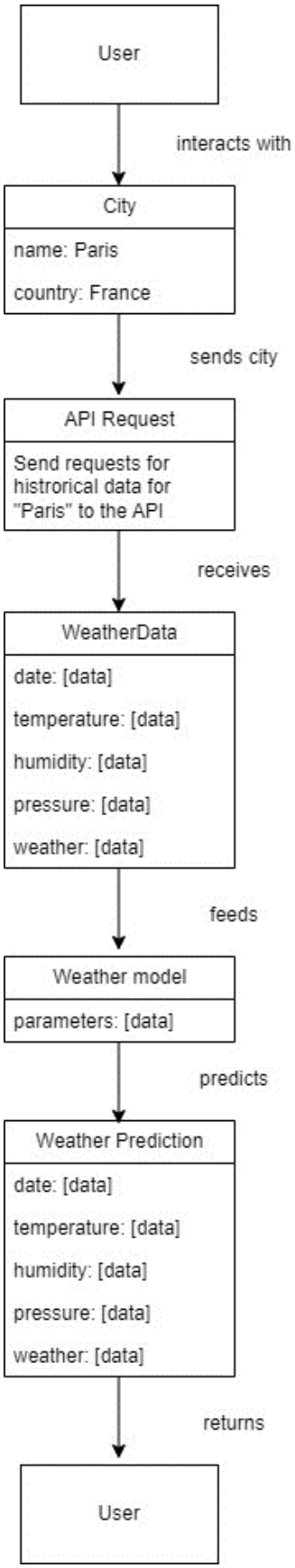
Use case Diagram



The use case of this application is very simple. The user enters a city name and get the weather information for response.

The Class and Sequence Diagram of the application displayed below




User
interacts with
City
name: Paris
country: France
sends city
API Request
Send requests for histrorical data for "Paris" to the API
receives
WeatherData
date: [data]
temperature: [data]
humidity: [data]
pressure: [data]
weather: [data]
feeds
Weather model
parameters: [data]
predicts
Weather Prediction
date: [data]
temperature: [data]
humidity: [data]
pressure: [data]
weather: [data]
returns
User

The application's design is quite simple; it's essentially an interface that shows how the weather models can process various inputs and produce precise forecasts. The application has four functions. `def get_weather_data(), def make_predictions(data), def get_geocode(city_name), and def display_result(city, weather_prediction, temp_prediction, humid_prediction, pressure_prediction).`

The `get_weather_data()` function asks user to input a city name, and then request weather data from the openWeatherMap API based on the city name, and finally pass the weather data and city name to the `make_prediction(data)` function and `display_result(city, weather_prediction, temp_prediction, humid_prediction, pressure_prediction`)function. The `make_prediction(data)` function loads models and feed the processed data to the models to make predictions, and finally pass the predictions to the `display_result(city, weather_prediction, temp_prediction, humid_prediction, pressure_prediction`) function. I the end the `display_result(city, weather_prediction, temp_prediction, humid_prediction, pressure_prediction`)function call the `get_geocode(city_name)` function to get the time zone of the target city and format the prediction results to output. The application's design is very straight forward, it does not store data, store user information or prediction results. Therefore, I did not create any classes in the file, the class graph primarily displays the relation between the functions.

2.5.9. Deliverables

The deliverables for this project are as follows:

- Python files – app.py (the application), train_and_evaluate.py (train, test, and valid models), process_data.py (process data)
- Datasets - vancouver_weather.csv (train and test set), testCsv.csv (validation set)
- Demos - screen shots, videos, graphs
- Final report - the overall report on the major project.

2.5.10. Installation guide

In order to run the application, just type `python app.py` The user may need to install the packages list below:

- requests: A Python library for making HTTP requests.
- pandas: A data manipulation library for Python.
- numpy: A numerical computing library for Python.
- joblib: A library for running CPU-bound tasks in parallel.
- pytz: A Python library for working with time zones.
- geopy: A Python library for geocoding and reverse geocoding.

- timezonefinder: A Python library for finding the time zone of a location.
- scikit-learn: a popular Python library for machine learning.

To install a package, open a terminal or command prompt and type:

```
pip install <package-name>
```

2.6. Testing Details and Results

2.6.1 Model Performance measurement

Accuracy, precision, and recalls are some common metrics that measures how often the classification model predicts the correct class for a given input. For example, if the model is predicting whether a person has diabetes or not, accuracy is the percentage of cases where the model correctly identifies the person's condition, and precision is the ratio of correctly predicted positive observations to the total predicted positives. These metrics are suitable for classification models, which predict discrete values such as categories, labels, or classes.

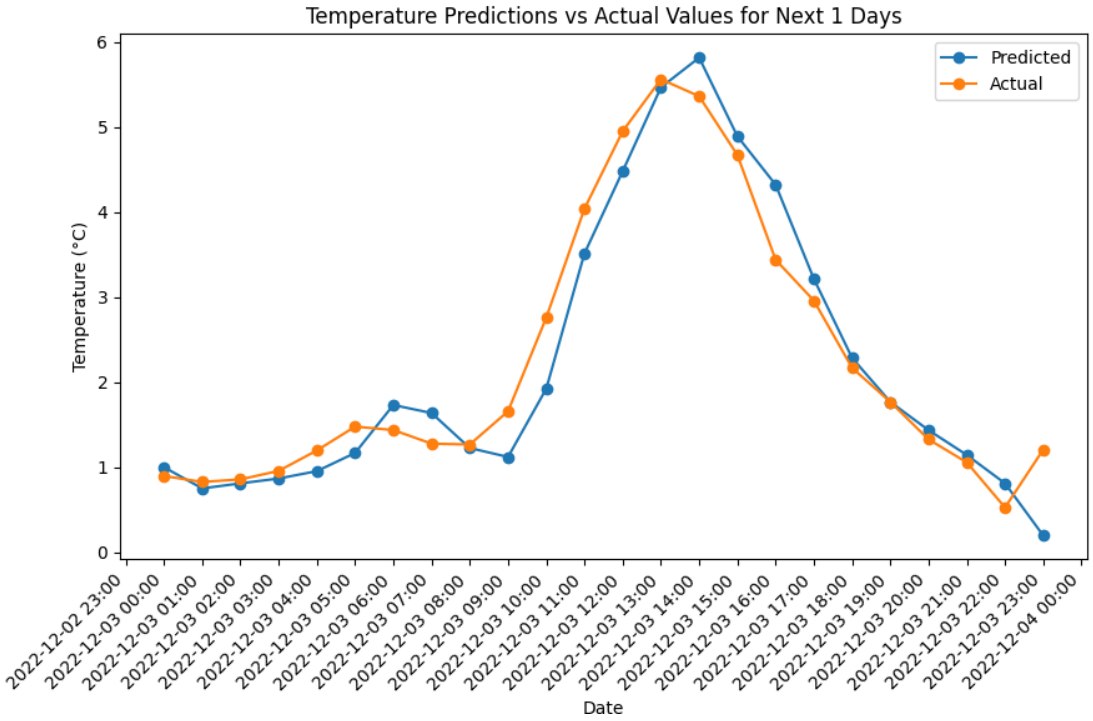
However, accuracy, precision and recalls are not suitable for regression models, which predict continuous values such as numbers, scores, or prices. For example, if the model is predicting the temperature of a city, accuracy is not a meaningful metric, because the model's prediction is rarely exactly the same as the actual temperature. Even if the model's prediction is very close to the actual temperature, it is still considered incorrect by the accuracy metric. Therefore, accuracy does not reflect how well the model fits the data or how precise the model's predictions are.

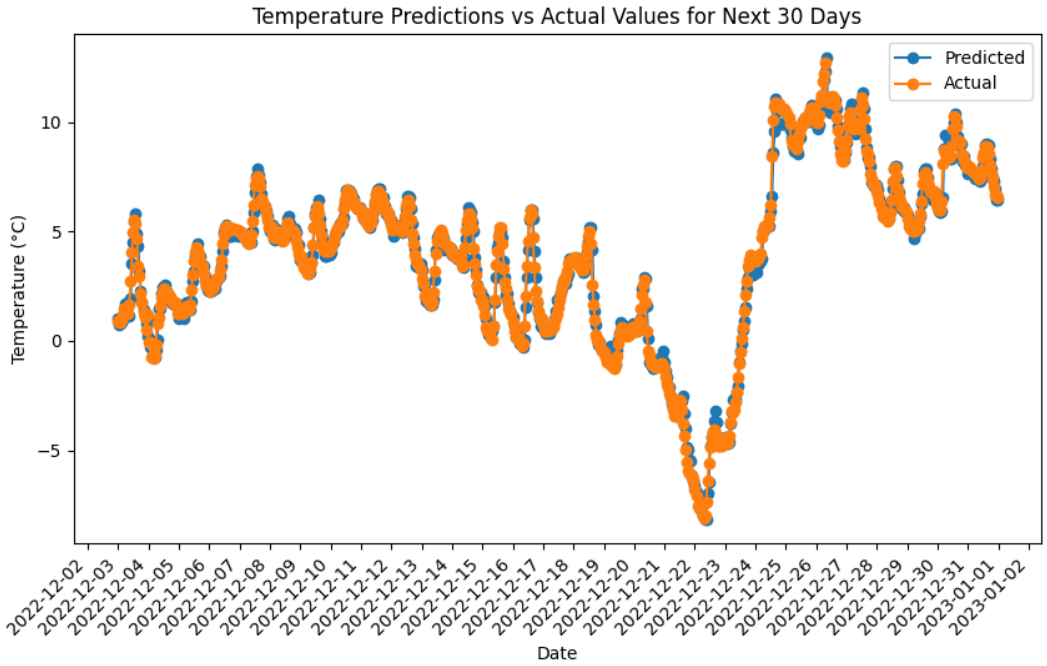
Instead of accuracy, regression models are evaluated using metrics that measure the difference between the predicted values and the actual values. These metrics are also known as error metrics, because they quantify how much error the model makes in its predictions. Some of the common error metrics for regression models are:

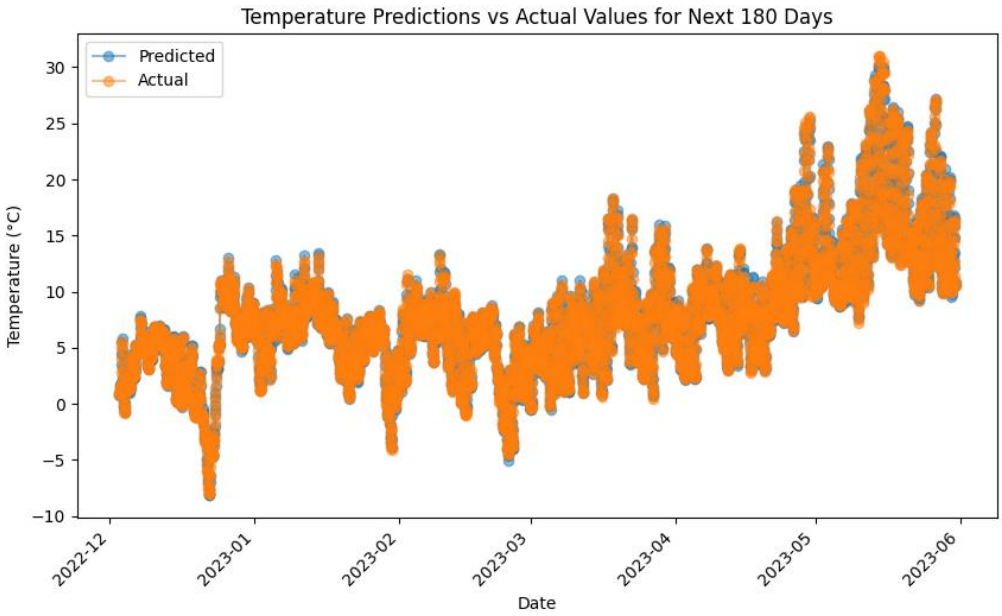
- Mean absolute error (MAE): This is the average of the absolute values of the differences between the predicted and actual values. It measures how much the model deviates from the actual values on average. A lower MAE means a better model.
- Mean squared error (MSE): This is the average of the squared values of the differences between the predicted and actual values. It measures how much the model deviates from the actual values on average, but it gives more weight to larger errors. A lower MSE means a better model.
- Root mean squared error (RMSE): This is the square root of the MSE. It measures how much the model deviates from the actual values on average, but it has the same unit as the target variable. A lower RMSE means a better model.

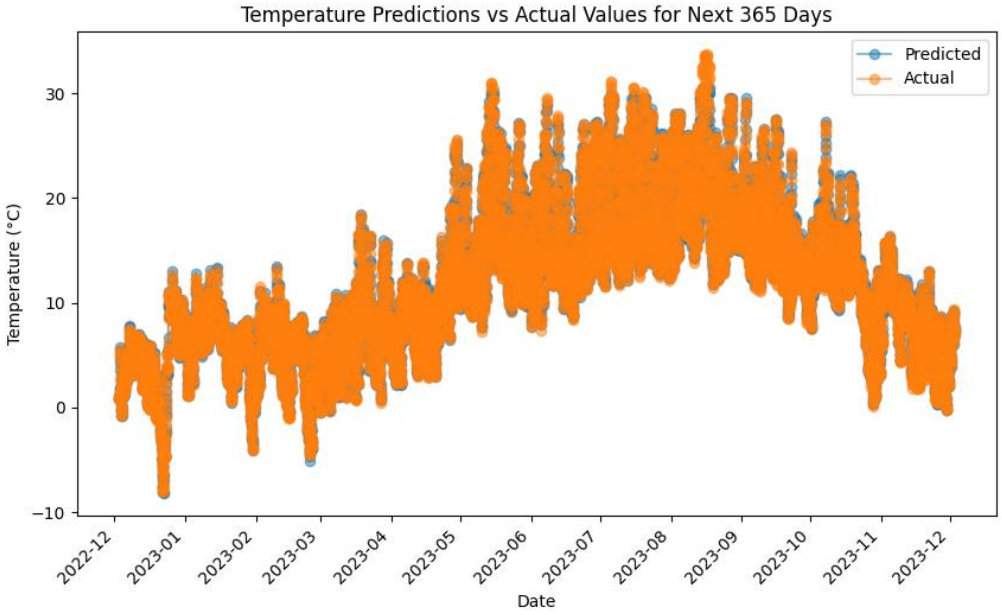
In this project, I will use the line plots and RMSE score to measure regression models (Temperature, Humidity and Pressure model), as it effectively demonstrates numerical differences between actual and predicted values, and use classification report to measure classification model (Weather model).

Temperature model










```
Testing results:
Root Mean Squared Error: 0.6776577952772326
Predicted Temperature for Next 24 hours: [ 5.4317  5.2182  4.8218  3.9407  3.8556  5.0372  5.0324  4.782   5.0964
  6.6902  6.2816  9.0444 10.3904 11.511  11.7755 12.3832 11.4542  9.3919
  7.0775  6.3089  6.04    5.8928  5.32    5.0896]
Validating results:
RMSE for Temperature predictions: 0.40
Predicted Temperature for Next 1 Days: [1.0618 0.7445 0.854  0.9404 0.97   1.2141 1.7606 1.6782 1.2747 1.1776
 1.8922 3.5927 4.4331 5.4095 5.8123 4.8873 4.1686 3.1058 2.349  1.7887
 1.4772 1.1453 0.8967 0.2356]
```

The feature selected for the temp model is `["Temperature"]`, and the hyperparameter is `n_estimators=100, min_samples_split = 2` The testing and validation results indicate a variance of 0.67℃ and 0.40℃ in temperature, as measured by the RMSE values of 0.67 and 0.40 respectively. Overall, the variance of the model is quite low, which means the model has a good performance.

Weather model

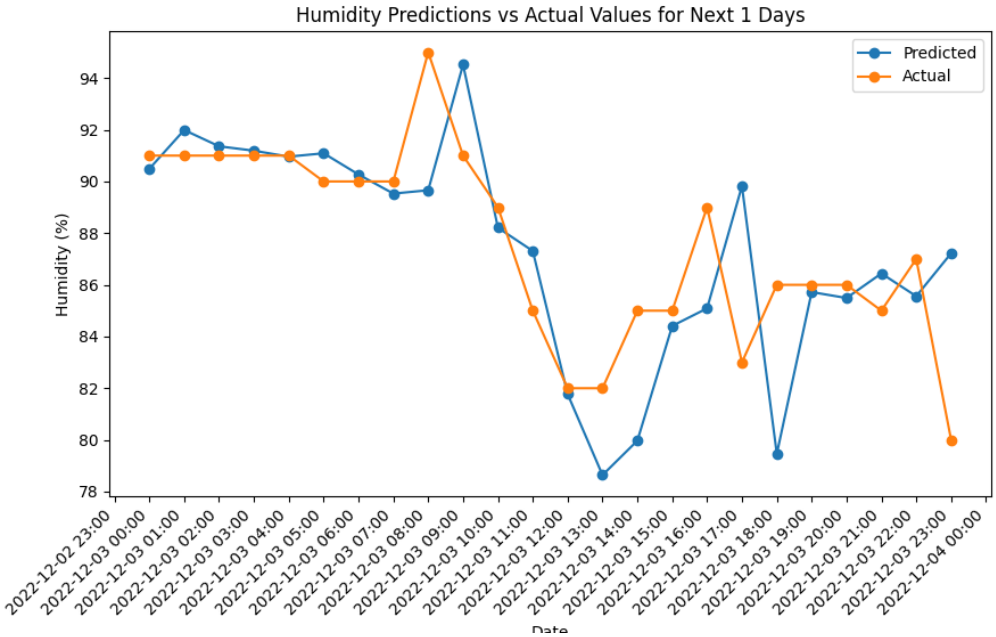
```
Testing results:
              precision    recall  f1-score   support

           1       0.78      0.81      0.80       790
           2       0.72      0.80      0.76      1104
           3       1.00      0.00      0.00         3
           5       0.74      0.35      0.47        84
           6       0.00      0.00      0.00        18
           7       0.52      0.29      0.37       164
           8       0.53      0.40      0.46       195
          10       0.70      0.93      0.80        42
          11       0.00      1.00      0.00         0

    accuracy                           0.72      2400
   macro avg       0.56      0.51      0.41      2400
weighted avg       0.71      0.72      0.70      2400

Predicted Weather for Next 24 hours: ['Smoke' 'Smoke' 'Sm
Validating results:
              precision    recall  f1-score   support

           1       0.70      0.74      0.72       370
           2       0.85      0.90      0.87      3061
           3       1.00      0.00      0.00        22
           5       1.00      0.03      0.06        32
           6       0.00      1.00      0.00         0
           7       0.11      0.15      0.13        65
           8       0.64      0.53      0.58       621
          11       0.61      0.31      0.41       149

    accuracy                           0.79      4320
   macro avg       0.61      0.46      0.35      4320
weighted avg       0.79      0.79      0.78      4320
```

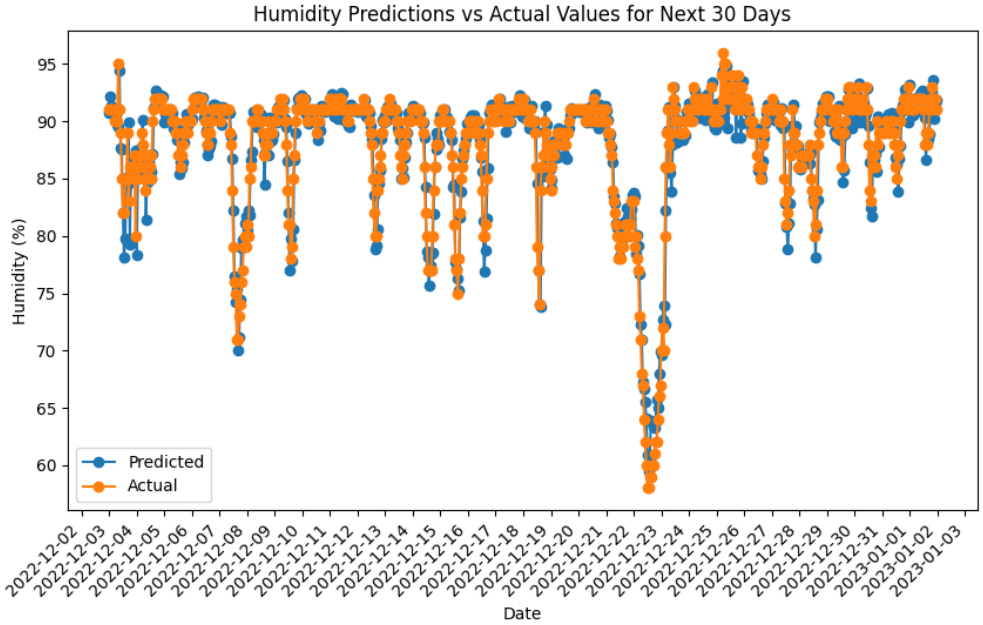
The feature selected for the weather model is `["Weather"]`, and the hyperparameter is `n_estimators=100, min_samples_split = 2`.

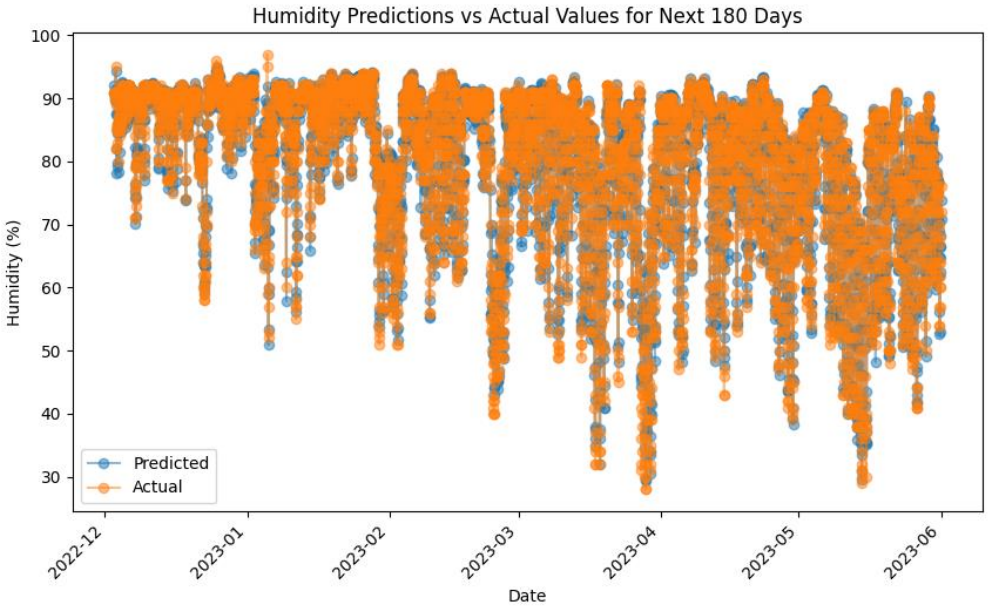
According to openWeatherMap's dataset documents, the weather condition field has following classes: ['Ash' 'Clear' 'Clouds' 'Drizzle' 'Dust' 'Fog' 'Haze' 'Mist' 'Rain' 'Sand' 'Smoke' 'Snow' 'Squall' 'Thunderstorm' 'Tornado'], and they are encoded to 0-14 before fitting the model. The testing and validation results indicate an accuracy of 0.72 and 0.79, and a macro avg score of 0.41 and 0.35. The test result is a bit lower than the validation result is because the test set is relatively small, and random fluctuations in the data could have a more significant impact on model evaluation. A larger validation set might better capture the overall distribution of the data. Moreover, weather like Haze, Mist, Snow, and Thunderstorm is uncommon in day to day

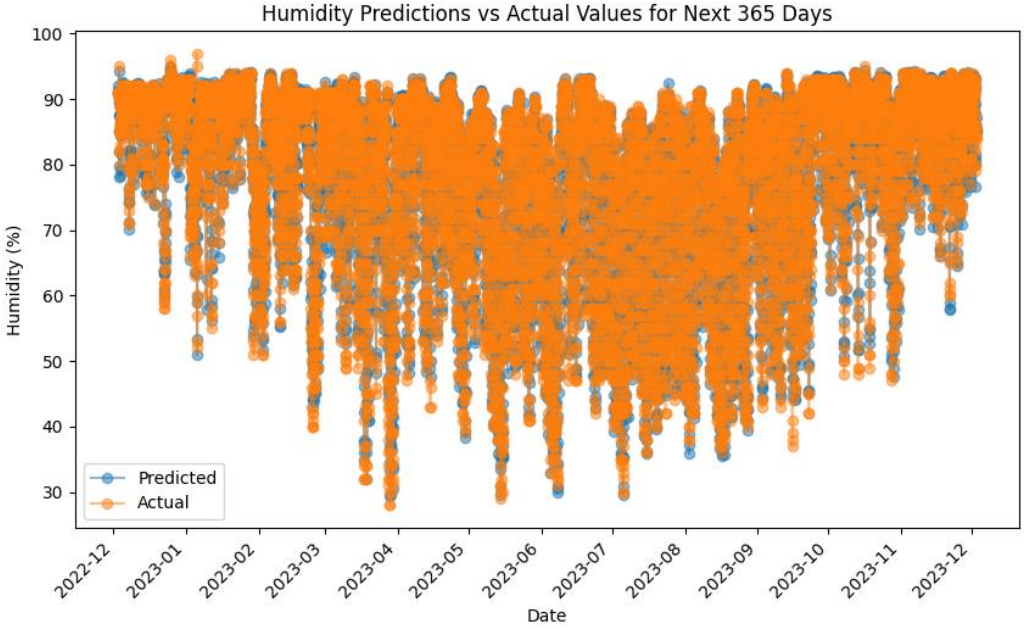
lives. The dataset contains limited instances of these weather phenomena, posing a challenge in accurately predicting such conditions correctly. Despite the above issue, the model's overall performance remains acceptable, as evidenced by satisfactory accuracy and macro-average scores.

Humidity Model





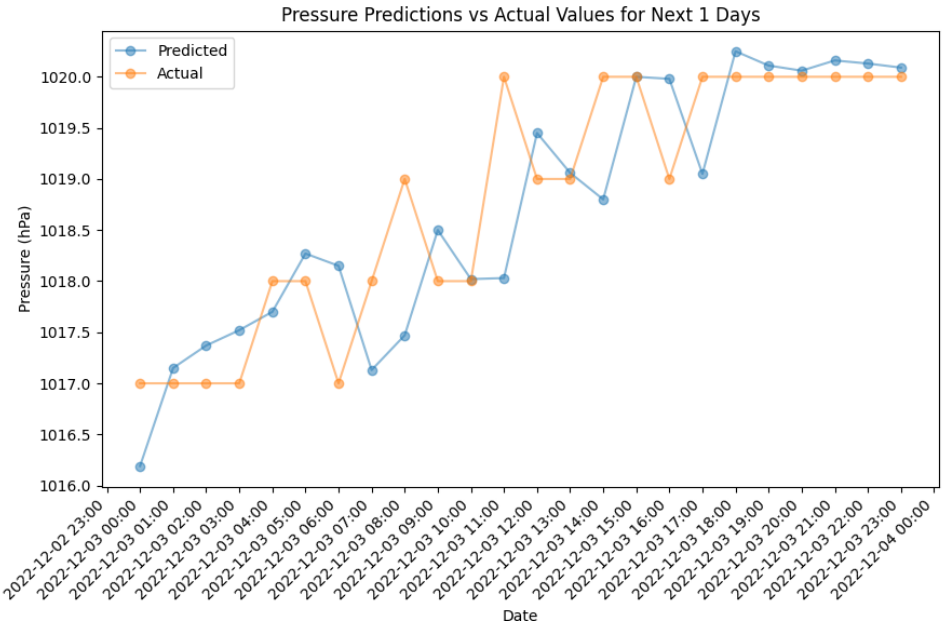


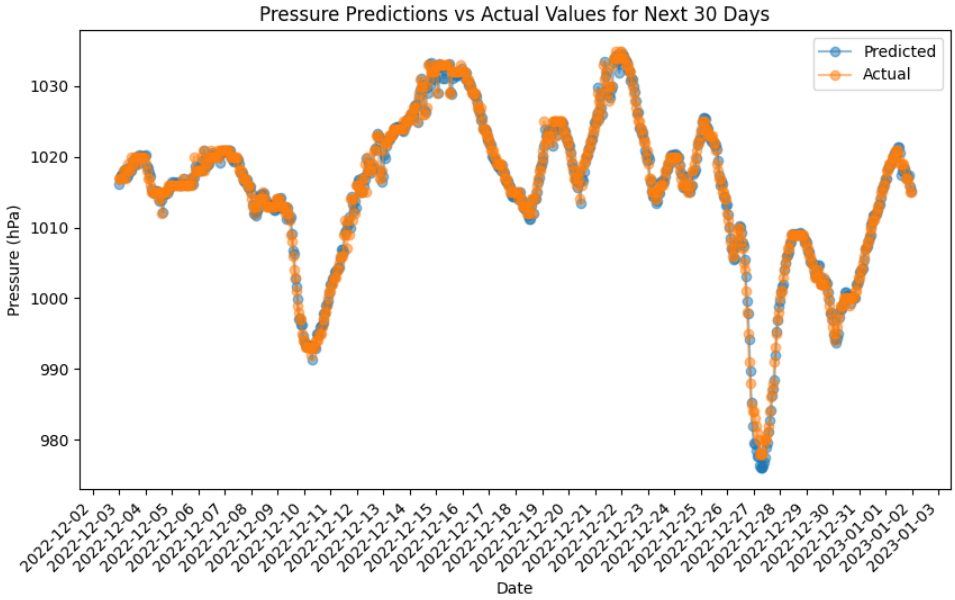


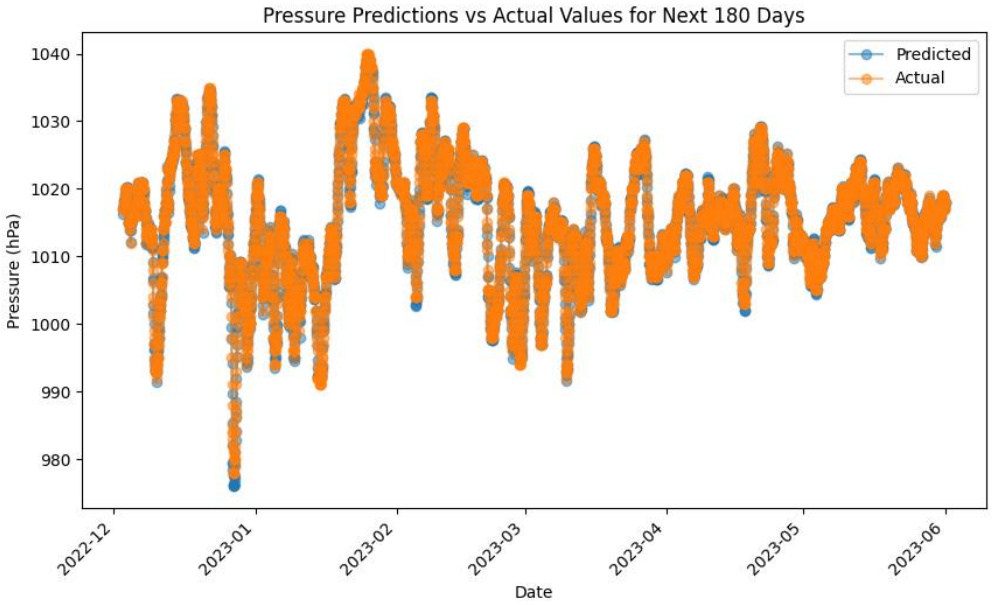

```
Testing results:
Root Mean Squared Error: 2.852894728750666
Predicted Humidity for Next 24 hours: [81.91 82.62 83.24 87.27 86.21 82.52 84.12 82.33 84.09 81.23 83.36 73.33
 72.93 66.64 66.64 64.66 64.64 76.57 85.44 85.86 84.88 80.28 84.26 84.41]
Validating results:
RMSE for Humidity predictions: 3.21
Predicted Humidity for Next 1 Days: [90.49 91.99 91.36 91.19 90.96 91.09 90.27 89.53 89.66 94.52 88.21 87.31
 81.77 78.64 79.97 84.41 85.09 89.82 79.46 85.72 85.49 86.44 85.56 87.22]
```

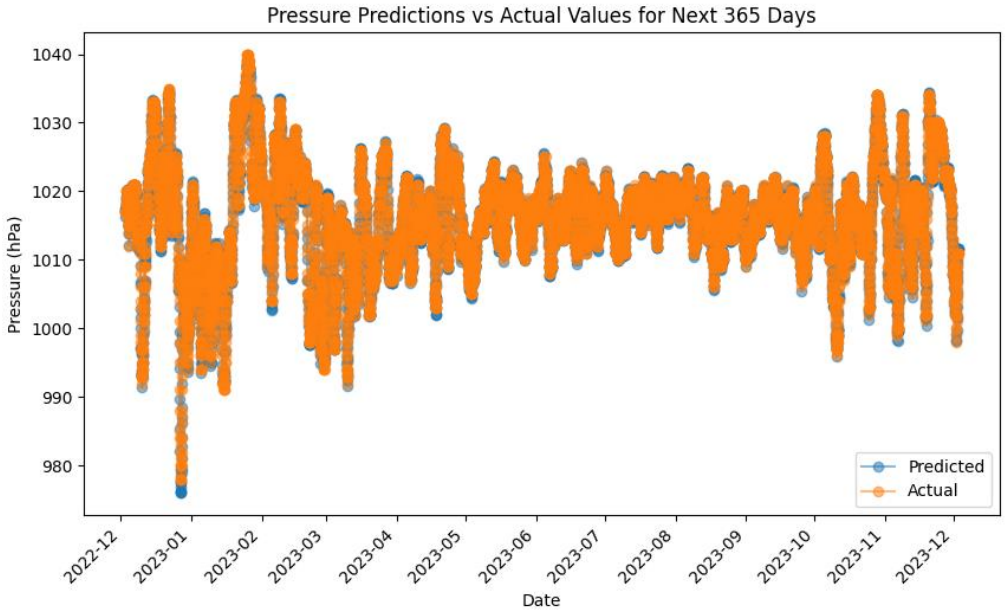
The feature select for the temp model is `["Humidity"]`, and the hyperparameter is `n_estimators=100, min_samples_split = 2`. The testing and validation results indicate a variance of 2.85% and 3.21% in humidity, as measured by the RMSE values of 2.85 and 3.21 respectively. Overall, the variance of the model is low, which means the model has a good performance.

Pressure Model










```
Testing results:
Root Mean Squared Error: 0.7637626158259893
Predicted Pressure for Next 24 hours: [1022.47 1022.42 1022.39 1023.97 1022.69 1021.21 1021.17 1021.28 1021.52
 1021.41 1021.66 1022.99 1021.61 1021.57 1021.08 1019.53 1019.64 1019.64
 1019.74 1020.05 1018.98 1019.02 1019.06 1020.1 ]
Validating results:
RMSE for Pressure predictions: 0.72
Predicted Pressure for Next 1 Days: [1016.09 1017.11 1017.37 1017.48 1017.68 1018.11 1018.1  1017.2  1017.47
 1018.23 1018.12 1018.19 1019.52 1019.06 1018.75 1019.8  1019.92 1019.14
 1020.23 1020.2  1020.01 1020.15 1020.19 1020.13]
```

The feature select for the temp model is `["Pressure"]`, and the hyperparameter is `n_estimators=100, min_samples_split = 2` The testing and validation results indicate a variance of 0.76hPa and 0.72hPa in pressure, as measured by the RMSE values of 0.76 and 0.72 respectively. Overall, the variance of the model is quite low, which means the model has a good performance.

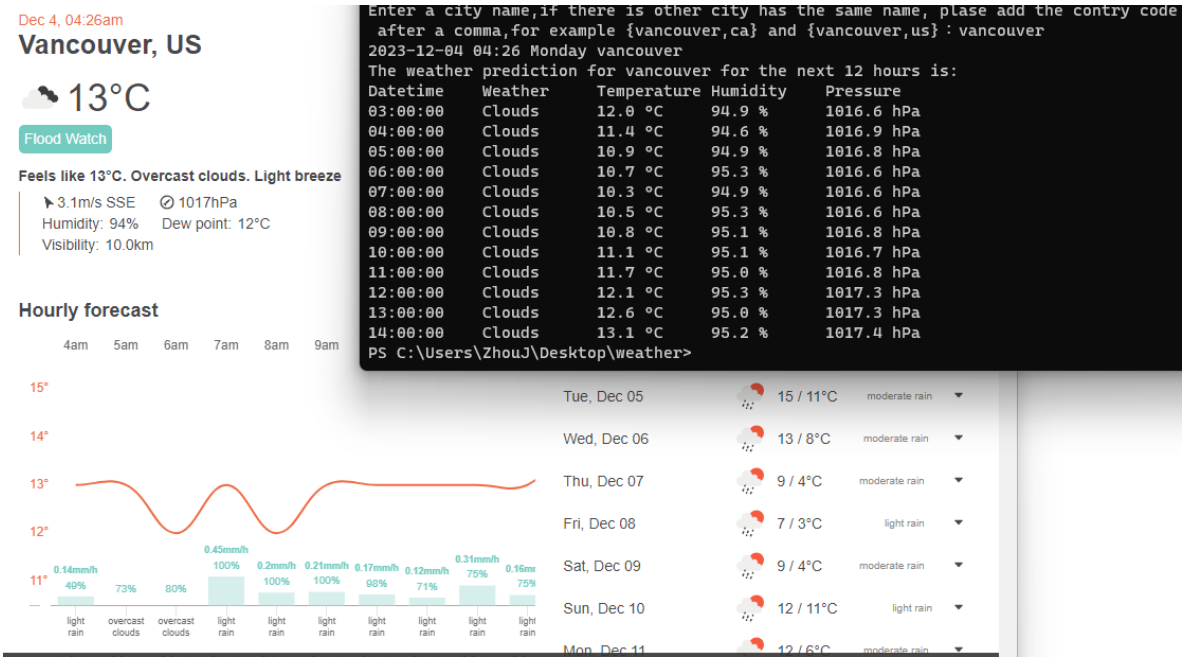
2.6.2 Application Test

Functionality Test

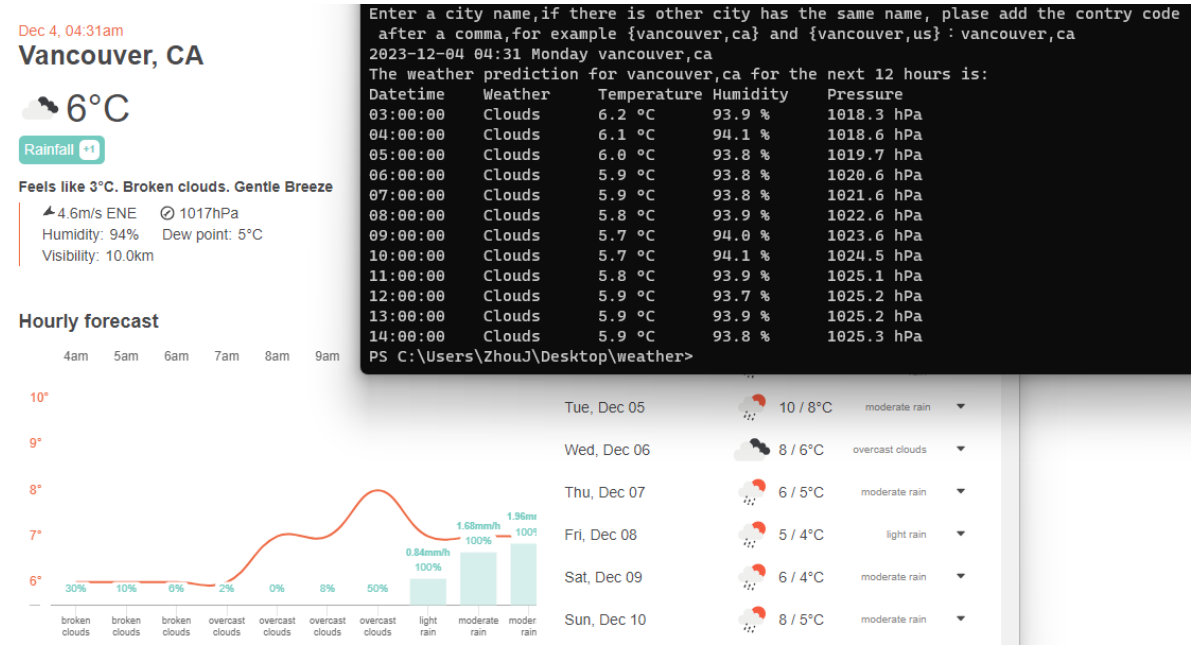
Test No.	Description	Results	Comments
1	Testing input city name {Vancouver}	The application successfully displays the predicted weather information and local datetime of the Vancouver in United State.	The result is acceptable. In cases where there exist several cities sharing the same name, openWeatherMap prioritizes the data of the city located in the United States. Thus, I suggest to add country code after the city name.
2	Testing input city name {Vancouver,ca}	The application successfully displays the predicted weather information and local datetime in Vancouver, Canada.	The result is as expected.
3	Testing input city name {Burnaby}	Returns an error show cannot find the weather data.	Unfortunately, openWeatherMap only offer history weather data for the major cities world wide.
4	Testing input random stuffs	Returns an error show cannot find the weather data.	The result is as expected.
5	Testing input nothing	Returns an error show cannot find the weather data.	The result is as expected.

The screenshots are listed below.

Test case 1



Test case 2



Test case 3

```
Enter a city name,if there is other city has the same name, plase add the contry code
 after a comma,for example {vancouver,ca} and {vancouver,us} : Burnaby
Error: Could not find weather data for Burnaby
```

Test case 4

```
Enter a city name,if there is other city has the same name, plase add the contry code
 after a comma,for example {vancouver,ca} and {vancouver,us} : sddev
Error: Could not find weather data for sddev
```

Test case 5

```
Enter a city name,if there is other city has the same name, plase add the contry code
 after a comma,for example {vancouver,ca} and {vancouver,us} :
Error: Could not find weather data for
```

2.7. Implications of Implementation

The project has three parts: the application, the datasets and the models.

- Datasets: The datasets were obtained from openWeatherMap API's historical data and were processed and saved to a csv file using the pandas library in Python.
- Models: The models were trained with random forest regression model, which is imported from python's sklearn library. The walk forward validation method was hand written. The sliding window was hand written. The models were saved using joblib library.
- Application: The application is written in python. The input data is obtained from the openWeatherMap API. The predictions were made by feeding input data to the loaded models.

2.8. Innovation

The innovation of this project is to improve the speed and accuracy of the current weather forecasting techniques using machine learning. The current weather forecasting apps on the market still use the combination of computer calculation and human expert analysis to predict the weather, but this kind of forecast is not always accurate, and takes hours to calculate. However, with AI forecasting, you can get the result in seconds at a very high accuracy. Let AI model to solve human's problem is the project's main innovation.

Another innovation of AI weather forecasting comes from its ability to make more accurately short-term and long-term predictions. Traditional methods use complex equations and often forecast for only between six hours and two weeks' time, and hardly to forecast for a longer range, but AI can predict if it will rain in two hours' time or after two-six weeks.

Moreover, Random Forest model itself does not account for the temporal aspect. Walk-forward validation is an innovative way to handle this problem by transforming the time series dataset into a supervised learning problem and evaluating the model using a specialized technique that accounts for the temporal nature of the data.

2.9. Complexity

The project is complex because it requires research skills, machine learning knowledges and software design skills. Most diploma students never enrolled in any courses about AI machine

learning, only the machine learning option students learned a bit, they won't know how to select, train, test and validate models, and have no ideas about how to hyperparameter tuning models to improve performance. This project also requires a lot of research, for example: research the time series forecasting algorithm and study for it; look for the datasets and cleaning the data; and search for solutions when face difficulties during the development. The lack of research skills would cause difficulties for diplomas to start the project. Moreover, diploma students don't have enough knowledge about software design architecture and project management. The projects they have done usually have clear instructions and requirements, but for this project you have to define your own requirements. It would be too hard for them to create such a big project from the scratch. Therefore, only those with a bachelor or higher education can complete this project.

2.10. Research in New Technologies

The project has covered several areas and technologies during its design and development. One of the main areas is time series forecasting, which involves predicting future values of a time-dependent variable based on historical data. To achieve this, the project has used several techniques such as Random Forest regression, walk-forward validation, and sliding window representation. Random Forest is a popular and effective ensemble machine learning algorithm that can be used for classification and regression predictive modeling problems with structured data sets. It can also be used for time series forecasting, although it requires that the time series dataset be transformed into a supervised learning problem first. Walk-forward validation is a technique for evaluating time series forecasting models that involves updating the model with new data as it becomes available and making predictions for the next time step. This approach allows the model to adapt to changes in the data over time and provides a more realistic estimate of its performance on new data. Sliding window representation is another technique used to transform time series data into a supervised learning problem by creating a fixed-size window of past observations as input features.

In addition to these techniques, other methods such as LSTM, Neural Prophet, and ARIMA may also be used for time series forecasting. LSTM is a type of recurrent neural network that can capture long-term dependencies in sequential data such as time series. Neural Prophet is an open-source library developed by Facebook that provides an easy-to-use interface for time series forecasting using neural networks. ARIMA is a statistical method for modeling and forecasting time series data that can handle a wide range of patterns in the data.

The development of the project has involved several technical challenges such as transforming the time series dataset into a supervised learning problem, handling unseen labels during prediction, and choosing appropriate hyperparameters for the models. The project has also required extensive research into various techniques and methods for time series forecasting.

Overall, the project has been innovative in its use of Random Forest classification and regression models with walk-forward validation sliding window to handle time series data. This approach provides a powerful and flexible model that can handle complex temporal dependencies in the data and provide accurate predictions for future time steps.

2.11. Future Enhancements

The weather models can be future enhanced by updating the datasets, hyperparameter tuning, or change to other time series models that lists in possible solutions. The application can also be update to a webapp with better GUI design.

2.12. Timeline and Milestones

The following section covers the development schedule and milestones of the AI weather forecasting project. All project code development was completed successfully on October 23th, 2023. Unlike my estimated total hours of 365 hours of effort, the actual hours roughly reached 535 hours. Most of this is time spent on researching and implementing different techniques and re-doing with different tools. During the middle of the development, due to my lack of awareness regarding the temporal order in the time series data, I mistakenly employed a random forest classification model instead of a regression model for training numeric targets. Additionally, I selected an unsuitable validation method, which result in the poor performance of the model. Researching for the correct method and re-training and re-validating models cost me a lot of time.

Timeline	***Milestones***	***Breakdown***	***Total***
Jan 4th to Jan 18th	**Sprint 1 Development environment configuration** ● **Research for all the required materials** ● **Configure Python environment** ● **Install any required packages/ modules** ● **Play around with installed modules and learn the basic usage**	**Planning: 15hrs** **Coding: 5hrs** **Installation: 5hrs** **Testing: 5hrs** **Documentation: 5hrs**	**35hrs**
Jan 18th to Feb 11st	**Sprint 2 Cleaning and dividing Data into sets** ● **Download Datasets** ● **Cleaning Data** ● **Divide Data into Training, Testing and Validation Sets.** ● **Write a python program to load data and display in graphs**	**Planning: 10hrs** **Cleaning: 10hrs** **Coding: 15hrs** **Testing: 10hrs** **Documentation: 5hrs**	**50hrs**
Feb 12th to Mar 13th	**Sprint 3 Training AI models** ● **Design the models** ● **Implement the R.F. algorithm** ● **Train the model using input data**	**Researching: 20hrs** **Coding: 20hrs** **Training: 15hrs** **Testing: 15hrs**	**80hrs**

		Documentation: 10hrs	
Mar 14th to Apr 20th	**Sprint 4 Testing and Validating models** ● **Hyperparameter tuning model to enhance accuracy** ● **Change validation method to enhance model performance** ● **Input testing data to check performance** ● **Input a new set of data for validation**	**Researching: 10hrs** **Coding:20hrs** **Testing:40hrs** **Documentation:10hrs**	**80hrs**
Apr 21st to May 20th	**Sprint 5 Researching for the correct method to process time series data** ● **Researching for the appropriate model choices** ● **Researching for the appropriate validation methods** ● **Understanding why previous method doesn't work**	**Researching: 40hrs**	**40hrs**
May 21st to Jun 20th	**Sprint 6 Re-preparing the data** ● **Download the history data from openWeatherMap API** ● **Process the data and save to csv file using pandas library** ● **Identify the important features in the dataset**	**Researching: 5hrs** **Coding: 5hrs** **Processing data: 10hrs** **Documentation: 5hrs**	**25hrs**
Jun 21st to July 31th	**Sprint 7 Re-train AI models** ● **Researching for the models** ● **Implement the Random Forest Regression model** ● **Implement walk forward validation for split data in to train and test set in respect to temporal order** ● **Implement sliding window to**	**Design:10hrs** **Research: 10hrs** **Coding: 30hrs** **Testing: 10hrs** **Documentation: 5hrs**	**65hrs**
August 1st to Aug 31th	**Sprint 8 Re-Testing and Validating models**	**Research: 10hrs** **Coding: 10hrs**	**55hrs**

	• **Hyperparameter tuning model to enhance accuracy** • **Add dependent features to enhance model performance** • **Input testing data to check performance** • **Input a new set of data for validation** • **Save models to local**	**Testing: 30hrs** **Documentation: 5hrs**	
Sep 1st to Oct 10th	**Sprint 9 Implementing the AI models to the python app and wrapping up** • **Write the basic python app framework** • **load the AI models into the app** • **Testing API requests and Response** • **Make sure the app can make predictions** • **Make sure the weather and date information displayed correctly.** • **Debugging**	**Design:10hrs** **Research: 10hrs** **Coding: 20hrs** **Testing: 20hrs** **Documentation: 5hrs**	**65hrs**
Oct 10th to Oct 29th	**Release** **Final Testing** **Final Report**	**Testing: 20hrs** **Documentation: 20hrs**	**40hrs**

3. Conclusion

Although the project to predict time series weather data using a random forest model with walk-forward validation and sliding window may not seem directly related to network security, there are several similarities between the two fields.

One of the main similarities is that both fields involve working with large datasets of time-dependent variables. In network security, these variables might include network traffic, user behavior, and system logs. In weather forecasting, these variables might include temperature, humidity, and air pressure. Both fields require the ability to analyze large amounts of data and identify patterns and trends over time.

Another similarity is that both fields require the ability to detect and prevent anomalies or security threats. In network security, this might involve detecting and preventing cyber attacks or

unauthorized access to sensitive data. In weather forecasting, this might involve detecting and predicting extreme weather events such as hurricanes or tornadoes.

Finally, both fields require the ability to make accurate predictions based on historical data. In network security, this might involve predicting future trends in cyber attacks or identifying potential vulnerabilities in a system. In weather forecasting, this might involve predicting future weather patterns based on historical data.

Overall, while network security and time series forecasting may seem like very different fields, they share several similarities in terms of the skills and techniques required to be successful. Both fields require a deep understanding of data analysis techniques, machine learning algorithms, and statistical methods for modeling and prediction.

3.1. Lessons Learned

In this project, the student learned how to use random forest model to forecast weather, and how to turn time series data into a supervised learning problem. The student also learned how to walk with large datasets and use walk forward validation and sliding window to process data into a set of input features and make future predictions.

In addition to technical skills, the project will also help students develop critical thinking, problem-solving, and communication skills. They will learn how to analyze complex problems, identify potential solutions, and communicate their findings to technical and non-technical audiences.

3.2. Closing Remarks

The project is very challenging and also entertaining because I learned a lot stuff about machine learning algorithms, data preparation methods and validation methods. The project also enhanced my problem solving and project management skills.

4. Appendix

4.1. Approved Proposal

The approved Proposal is attached to the email.

4.2. Project Supervisor Approvals

The approvals are not yet granted.

5. References

- Tyler Herrington. (September 23, 2019). Why is the weather so hard to predict?

  https://letstalkscience.ca/educational-resources/stem-in-context/why-weather-so-hard-predict

- Javatpoint. (n.d.). Random Forest Algorithm

https://www.javatpoint.com/machine-learning-random-forest-algorithm

- Marwah Sattar Hanoon, Ali Najah Ahmed, Nur'atiah Zaini, Arif Razzaq, Pavitra Kumar, Mohsen Sherif, Ahmed Sefelnasr & Ahmed El-Shafie (September 23, 2021). Developing machine learning algorithms for meteorological temperature and humidity forecasting at Terengganu state in Malaysia

  https://www.nature.com/articles/s41598-021-96872-w

- Geeksforgeeks. Random Forest Hyperparameter Tuning in Python https://www.geeksforgeeks.org/random-forest-hyperparameter-tuning-in-python/
- ChatGPT- https://chat.openai.com/
- Jason Brownlee. (November 1, 2020). Random Forest for Time Series Forecasting https://machinelearningmastery.com/random-forest-for-time-series-forecasting/
- Jason Brownlee. (August 21, 2019). How to Convert a Time Series to a Supervised Learning Problem in Python https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/
- Juan Nathaniel. (May 28, 2021). Introduction to ARIMA for Time Series Forecasting https://towardsdatascience.com/introduction-to-arima-for-time-series-forecasting-ee0bc285807a
- Caner Dabakoglu. (Jun 23, 2019) Time Series Forecasting — ARIMA, LSTM, Prophet with Python https://medium.com/@cdabakoglu/time-series-forecasting-arima-lstm-prophet-with-python-e73a750a9887

6. Change Log

- Oct 29, 2023 - First version of the report completed.
- Nov 23,2023 – Describe the random forest algorithm more detailly in section 2.4 Chosen Solution and indicates whether the alternative algorithms is used by mainstream forecasting solutions in section 2.3 Possible Alternative Solutions. Added section 2.5.1 Feasibility Assessment Report, section 2.5.2 Model API use, and section 2.5.7 Decision trees. Updated graphs to section 2.5.8 Application Design. Provided more explanation and screenshots to section 2.6 Testing Details and Results.
- Dec 3, 2023 - Add line graphs of actual vs predicted value to section 2.6 Testing Details and Results. Add graphs to 2.5.5 Training and Testing, section 2.5.6 Model Validation.