Group Number: 38
Rohan Sharma: 639271
Yash Narwal: 612840
Shuang Gao: 706079

Reflection

Changes we made

We made several changes which are outlined and justified below.

- We changed our prediction model slightly. Instead of our prediction model belonging to a day, we changed it to belong to a weather station. This was done because we found it quite difficult to handle specific cases while implementing our project, for example when someone asks for a prediction 3 hours into the future, and at it is currently 11pm, we would need to get 6 predictions from today, and 12 predictions from tomorrow. By making this change, when a client asked for N predictions we would just retrieve the 'latest N' predictions from the appropriate weather station and return it to the client. This made our solution simpler.
- We added a number of fields and methods to our models. This was done because we hadn't considered all the fields and methods we would require when designing our application. For example, to our prediction model, we added fields such as temperature variance, wind speed variance, wind direction variance and rain variance and several class methods to help us retrieve predictions. These were implemented to increase functional decomposition and make our application more modular. These are outlined in the class diagram submitted with this project.
- Initially, we were going to use Statsample for regression to predict future data based on the current data. However, while implementing the project, we decided against this as it wasn't necessary to use Statsample since we had written classes which perform different types of regression for us in Project 1. After testing these classes on the weather data, we felt they were appropriate for the amount of calculations we were making and gave us no problems with performance.
- We made minor additions/adjustments to the application design, such as using another API to get latitude and longitude for a given post code in Victoria.
- Initially our plan was to make predictions by looking at data from the past 7 days. Suppose we were trying to make a prediction for 4.30pm today. We would

regress on the data for the past 7 days around 4-5pm. Then based on the equation we got from this regression, we would predict what the weather will be like today at 4.30pm (extrapolation). However while implementing and testing our application, we realised that each time we turn off our computer (or do a rake db:drop so the data is gone/old), we will be unable to test the prediction systems without waiting for 7 days to get new data. To work around this, we added an additional case to our prediction system. If there is at-least 4 days of data available, we use the strategy outlined above. If there is less than 4 days of data available, we regress only on the weather data collected today.

Aspects we found challenging

There are several aspects of the prediction system we found challenging, and they are outlined below. Overall, the prediction system was the only thing that gave us trouble.

- Working out the probabilities was very difficult. Having never done any probability, we decided to come up with a model that described how much confidence we have in our predictions. Our model took the following things into consideration: Distance to the closest weather station and how far the prediction is being made into the future. We assumed you can make reasonable predictions when a weather station is less than 25km away from you. Additionally each prediction we make in the future, the probability decreases by some amount. This allowed us to come up with a confidence model to represent as a probability for our predictions.
- Finally, our background scraping script was running way too slow. It was taking over 5 minutes to complete. We managed to fix this by encapsulating the entire background scraping script into a single database transaction.

Our thoughts on designing prior to implementation

Having a design to follow during implementation made us program in a more orderly way than we usually do. We knew what we were doing every step of the way because our implementation could be broken up into several stages as per the design.
We spent our first week implementing the models, and testing whether we got the relationships right. We used our class diagram in implementing these and only changed minor things.

In the next two weeks, we implemented the methods within the models, controllers, and our background scripts. For our background scripts and functions we used our sequence diagrams for implementation. Only minor details had to be changed to get them working.
Through every stage of implementation, the component diagram gave us an overview of our application, allowing us to keep our development inline with our overall design goal.

Additionally, a thorough design enabled us to work better as a team, as we all got together to decide upon the specifics of the design. At the implementation stage, there were no conflicts about how x should implement y.
With the class diagram, we were able to distribute the work evenly, as each person could implement N models.

Overall, it has been an enjoyable experience and we are proud of the application we have been able to develop. We have learnt the importance of design in making software systems and will continue to design before implement in the future!