OUnit Testing vs. Manual Testing

Column and Datatable modules: these modules were tested with OUnit because they are very suitable for automated testing. It is very easy to construct tests because we can easily construct column and datatable structures with the make method, and call functions on that. We constructed extensive tests, testing as many edge cases as possible (3-4 tests per visible functionality)

Ema module (moving averages and normal averages): this was especially easy to auto-test with OUnit because each function takes an input of an int list which is very easy to construct and an output of a list or number (both very easy to construct and compare). Edge cases are needed for this especially, such as empty list testing, and these were extensively tested in our OUnit tests suite.

Linear_regression module: this was manually tested because in general, a lot of outside data is used, and we did not want to manually copy-paste that into the code or mess with the test folder organization. Our manual tests had very good coverage because they included such big data sets though, so we are confident that linear regression functionality works.

Load module: we choose to manually test this module for many of the same reasons as the linear_regression module. As we did not want to copy-paste large CSVs into ml files, we found that using dune utop and loading in large CSVs via the command line as it was much more effective in ensuring the rigor and correctness of our code and more convenient.

Neural_network module: we also choose to manually test this module as it would be extremely tedious to ensure that our neural network was working as typing in hardcoded values for network weights would be very tedious and time consuming. Instead, we created a ml file to use the methods in our neural network module and ensure that our model is training via loss reduction – this served as our confirmation.

Processing module: this module was created to process NBA data. Although this required a lot of data to test, we felt that it was important to OUnit test this due to the importance of this module's functionality in almost all of our user interface. We took data from a CSV file, but it was a little messy because we did not want too much data in our code.

Utils module: this module mostly just incorporates helper functions for processing and printing various objects. Since most functions are prints or just very simple modifications, testing was not possible for some and we decided to just do manual testing for the simple functions.

Test Case Development

As stated above, we tested every module in lib, using OUnit to test Column, Datatable, Processing, and EMA, while just manually testing the rest. We used a combination of black box and glass box tests. We first wrote a few test cases before implementing the functions, testing them after implementation. After implementation and testing, we did a few glass box tests to achieve full coverage and test as many edge cases as we could think of. As for manual testing,

we generated a few random test cases for things like neural networks testing (just for checking if it runs without errors), and for the most part just using the CSV files to load data as testing data.

Correctness of our program

For the modules we tested with OUnit, we are confident of correctness due to a combination of black box testing for functionality and glass box testing for edge cases and coverage. For our other modules, we used a combination of randomized testing and large datasets, which returned values that were very reasonable. Of course, we also used small test cases to verify exact correctness, but our main area of concern was integration with big data.