Part 1:

Firstly, I have decided to record the timestamp of when the scripts are run, alongside the name of the script and the arguments used. The arguments used would be within brackets, and all of the data displayed would be separated by hyphens, to be more readable This means I can access previous logs of running the scripts within the same file log. I can access particular runs of certain scripts within the log using the less command, and the timestamp. The formatting (with the new lines) was incorporated to make the logs easy to navigate and understand when parsing through them.

I have decided that whenever I create a new directory, that I would log it in the file, in case that the directory is misspelled, due to the use of an incorrect literal, or for any other reason where a directory may be created. This is done for the initial directory that the user inputs as an argument, as well as the out and data directories that may be created if not already present.
After this, I have decided to output all the URLs that data is downloaded from, both to make sure that the correct URL has been used, as well as to determine the that the formatting of the URL being created is accurate. The reason I decided to use the latter is because I had issues with the URL not being formatted correctly, so I would use the output from the logs to tweak the program slightly in order to result in an accurate URL as required.

For the ProcessData script, I have decided to write down whenever the -t flag is used or not, because it can be used against the final outputs, and I can determine whether the filter script was correctly called or not using the command. As well as this, It outputs all the routes being checked into log file, and also adds the VehicleId when it is added to the VehicleIdDict dictionary.

If there was ever an error with the running of any script (due to incorrect type or number of arguments) I would write this down in the appropriate section of the log file so I would be able to access this and determine why a certain input caused an error, and act upon this in post. At the very end, I would add an extra two empty lines, for sake of readability and to allow a separation between logs from separate runs of tests and scripts.

Part 2:

First, the script checks if the number of parameters input is not 1. If not, it outputs a statement as an error, stating there usage of the shell script was incorrect and that it should be used in the format: shell script name then directory to act on, and then it exits the script.

If the number of parameters is correct, then the line of the command:
“ du -h $1/data/* |sed 's/\.\d*//g' |sed 's/K/000/g'| sort |head -n5 “
is used to produce the required output.

Going individually per command:

The command du -h $1/data/* is used to estimate the file space of each file in the directory $1/data/, where $1 is the name of the directory where the files are to be accessed from. This file size is written in a human readable format, so in kilobytes rather than regular bytes, and this is done for all files. The problem with this is that the file sizes are rounded to the nearest whole number, therefore a file that is 28056 bytes, and one that is 22051 bytes may be assumed to be the same file size, as when converted to kilobytes, they both are shown to be 17KB, which is not what we require.

The output of the previous command is then piped into the command sed 's/\.\d*//g'. This sed command takes all instances of a dot “.” and 0 or more instances of digits and replaces it with nothing (essentially deleting this). This will occur for any matches across the line being converted, thus the use of /g at the end of the regex expression. This can simply be replaced with a delete operation, though this does not achieve the purpose that this was intended for, and deleting the dot means that the correct output of <filename>.csv would not be achievable, so this command should not be used.

The output of the previous command is then piped into the command sed 's/K/000/g'. In this command, all instances where ‘K’ shows up, this will be replaced by 3 zeros. This is most likely done to convert the number of Kilobytes to bytes so it can be used to sort the files in a correct order. However, this can be replaced, and made more accurate by just having the initial du command output the data in bytes. As well as this, due to having “g” at the end of the regex expression, all instances of “K” will be replaced by 000, which is not what is intended, therefore this would have to be removed. An example of this is that the file name of King’s.csv would be changed to 000ing’scsv (the dot is omitted due to the previous command).

The output of the previous command is piped into the command sort, which will sort the files line by line, from smallest file size to highest (ascending order). However, the sizes are ordered on numerical order, rather than the actual value of the number, so 9 would be considered larger than 100, despite 100 being larger than 9 by value of the whole number. This can be avoided using the -g flag, which orders based on the actual numerical value, so 100 is considered larger than 9.

The output of the previous command is piped into the command head -n5, which will return the top 5 lines that are input into this command. The major issue with this is that, following the flow of the commands, the 5 smallest in size files will be returned, rather than the largest 5 files. To get the top highest commands, we need to use the tail -n5 operation, or change the previous sort command to have a -r flag, so it will sort the command from largest file size to smallest (in descending order).