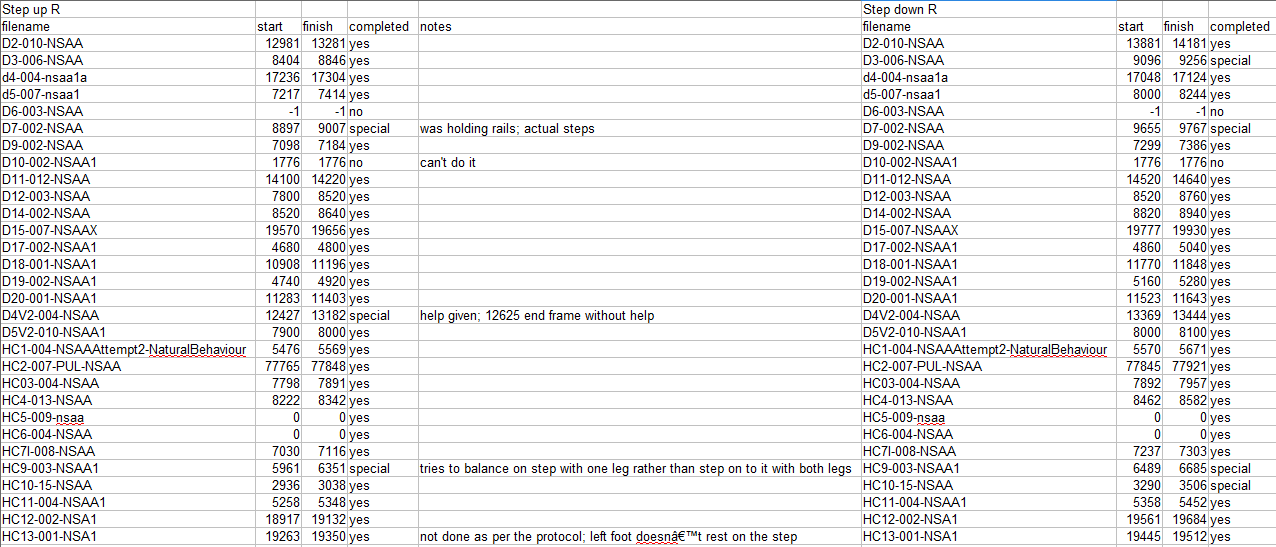
**Reference Documents Explanation**

As we can see from the system overview diagram, as well as having many Python and batch scripts that play integral roles within the system, we also make heavy use of several documents that provide much needed information for training models and places to store results of experiments. These include the Google sheet of annotations, the reference document containing the overall and individual scores of each subject used in the project, the ‘model\_predictions.csv’ file, and the ‘RNN Results.xlsx’ file. Each of these files in turn can be found within the project source directory within the ‘documentation’ directory. In the second below, we will discuss each in turn, from the information they contain to how and where they are used within the system (i.e. what other scripts depend on them and what scripts feed into the documents). The aim here, therefore, is to give a more concrete understanding of what these documents contain prior to seeing them referenced in the results discussion and general system overview sections.

**Google Annotations Sheet (‘nsaa\_17subtasks\_matfiles.csv’)**



As part of the wider research initiative, we collaboratively undertook to analyse and record the times of each activity undertaken by the subjects as part of the initiative. The information that was collected into this document, therefore, was intended to be used as part of several projects that relied on the start and end times of each activity undertaken by the subjects. As a result, the subjects’ corresponding activity videos were divided into three parts and each of us determined the activity times for our given subjects.

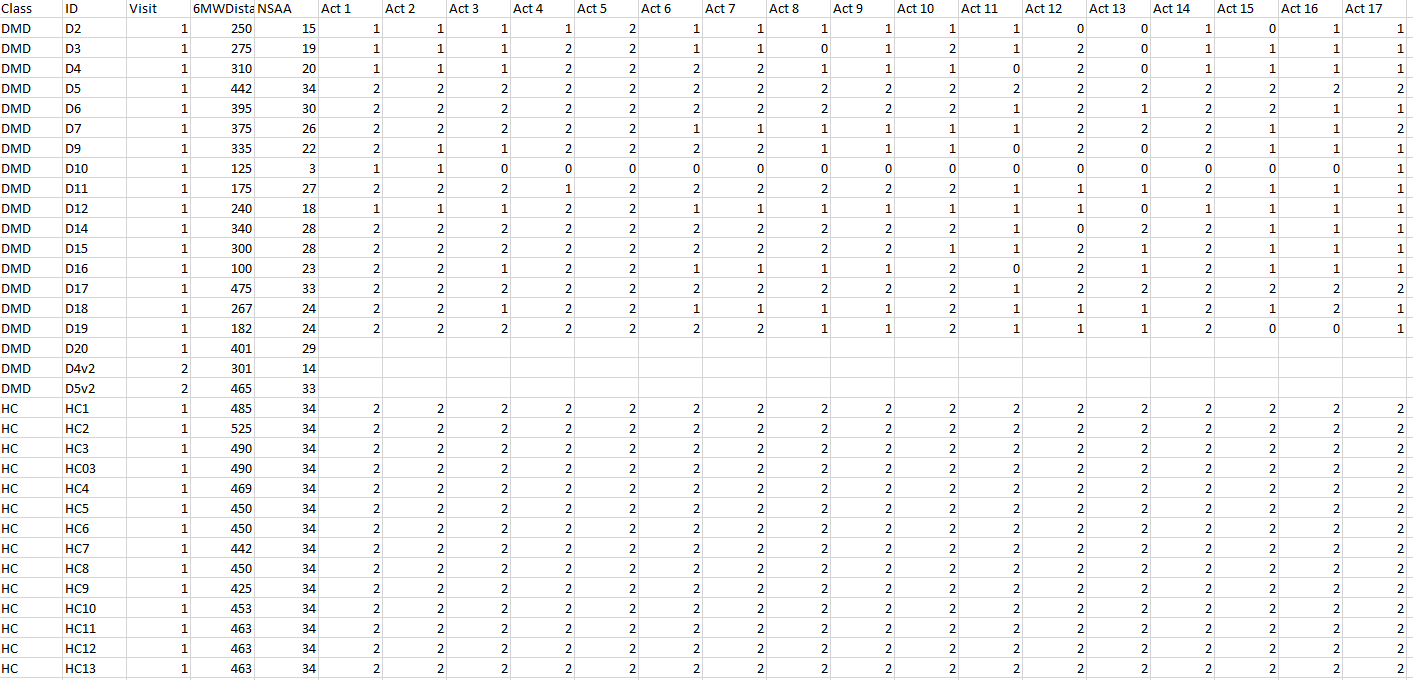
The process to undertake these annotations were as follows:

1. Load either the source video that corresponds to the ‘.mat’ file (which is provided in the data sets as ‘.mov’ files) OR use the corresponding ‘.mat’ file with the ‘dis\_3d\_pos.py’ script to load a basic 3D dynamically updating image through the ‘matplotlib’ library.
2. For each activity of the NSAA assessment set (e.g. ‘raise from floor’, ‘running’, ‘step up using right foot’, etc.), observe the time in seconds (or frames) the activity starts and finishes in the file. Note that we only count the **first completed** activity within the source file and we try to be slightly accommodating of the start time (for example, if the activity started between 3s and 4s in the file, we record it as having started at 3s to ensure we capture the complete activity with a bit of ‘slack’).
3. For the subject in question and for the activity in question, record the start and end times in **frames** in the ‘start’ and ‘end’ columns for the activity (note that if the time was observed in seconds, simply multiply this by 60 as the suit samples at 60Hz), along with recording the name of the file that the activity occurred in (as this will be the same name as the corresponding ‘.mat’ file the suit data for this video will be in, just with a ‘.mat’ file extension instead of ‘.mov’).

In the image above, we can see a snapshot of several activities that have been recorded for some of the subjects, including the file names containing the activity in question for the subjects, along with the start and end times within the respective files. It should be noted that not every activity could be drawn from each of the subjects’ files. This could be due to the subject simply not performing the activity they were told to perform or were otherwise unable to perform the activity. In these cases, the start and end times will be marked with either a ‘-1’ or ‘0’, with a ‘0’ sometimes signifying an incomplete activity annotation done by the annotator.

For our project, the main use of this document is as a tool for the ‘mat\_act\_div.py’ script. For each of the source files that the script wishes to divide up, it will look in the table for the name of the subject in question for that source file, find the row corresponding to that file and, for each activity, get the start and end times for that activity and extract the corresponding frames from source file (making sure it’s name matches that of the activity’s ‘filename’ column entry). So while these ‘single-act’ ‘.mat’ files will be further processed by other scripts (such as ‘comp\_stat\_vals.py’ and ‘ext\_raw\_measures.py’), ‘mat\_act\_div.py’ is the only file that directly relies on the google annotations script, and no file within the system modifies it in any way.

**NSAA Scores and Walk Reference Document (‘nsaa\_6mw\_info.xlsx’)**

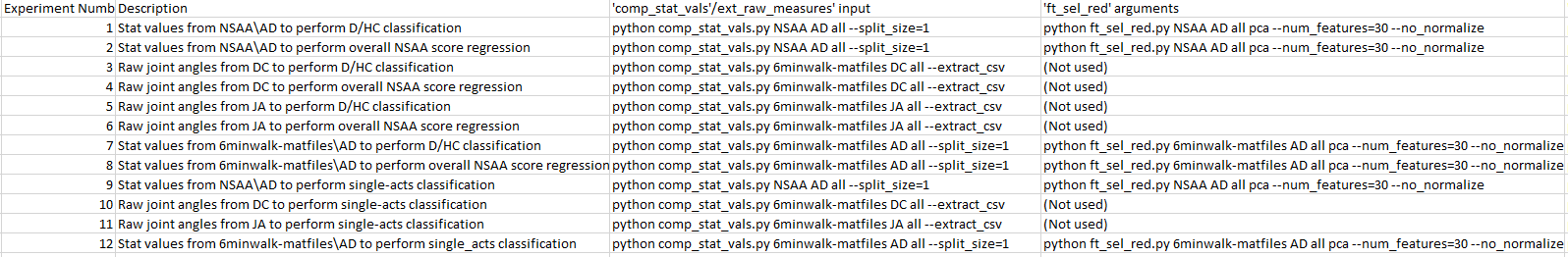


The next script in the system is the reference file we use to create the ‘y’ labels used by every model that is built in the system. As a result of it being the only place that contains the ‘y’ label information for each of the subjects, it’s referenced by several scripts including ‘ft\_sel\_red.py’, ‘rnn.py’, and ‘model\_predictor.py’. The script itself was a collaboration of several other scripts found within the source data directories for ‘NSAA’ and ‘6minwalk-matfiles’, including ‘KineDMD data updates Feb 2019.xlsx’ and ‘nsaa\_matfiles.xlsx’. Some of these scripts contains information about certain subjects that others don’t; hence, rather than having the scripts checking each of the files in turn, it was felt that it would be easier to combine the information from all of them into one file.

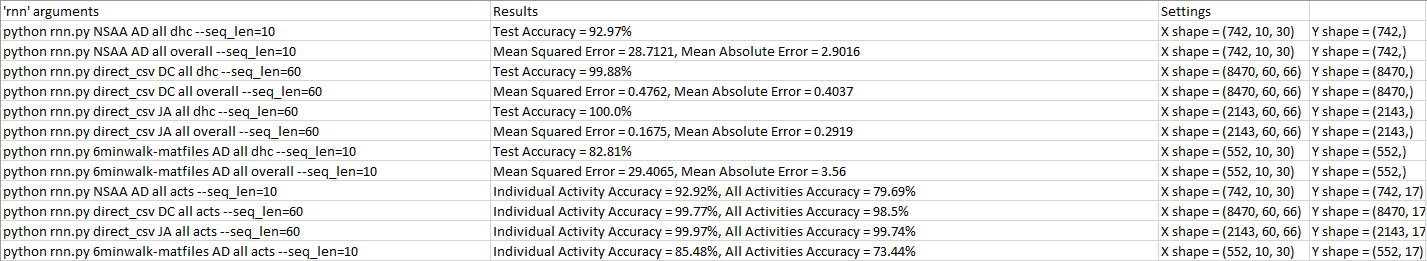
Each of the models that are built by the ‘rnn.py’ script are built to target one output type, which will be either ‘dhc’ (the D/HC label), ‘overall’ (the overall NSAA score), ‘acts’ (the 17 individual activity scores), or ‘indiv’ (the score of the individual activity assuming the input are single-act files) for each of the sequences the model is training and subsequently assessing on. For the ‘dhc’ output type, we don’t need to reference this file, as this can be determined simply by the file name. For example, in preprocessing the data into ‘x’ and ‘y’ components within ‘rnn.py’, if the given data that is currently being preprocessed for a given file is from a file called ‘D4\_position.csv’, then we know that the ‘dhc’ label would be ‘D’ (or 1 when being fed into the network), while if the data came from the ‘HC6\_position.csv’ file the label would be ‘HC’ (or 0).

However, for the other types of ‘y’ labels that we need to train our ‘rnn.py’ for the other output types, it’s not as simple as observing the name of the file. This is where the NSAA score reference file comes in. As we can see in the above image, this contains the information for every subject we have (that was collected by the initial assessors of the subjects) that includes their individual activity scores and their overall NSAA score. Hence, by finding the relevant row within the document, we get all the information we need for the other output types. For example, let’s say the preprocessing function of ‘rnn.py’ is extracting data from the file corresponding to subject ‘D16’. Once the data file has been separated into their sequences with necessary sequence overlap and an optional proportion of the sequence dropped (more on these later), we then need to get the corresponding label for this sequence. If the model is being trained for the ‘overall’ output type, then we check the table for the value in the ‘D16’ subject’s ‘NSAA’ column, which corresponds to ‘23’. This is the ‘y’ label that each sequence extracted from the ‘D16’ source file would get for the ‘overall’ output type. Alternatively, if the model output type was instead ‘acts’ the ‘y’ label would be a list of values ‘[2, 2, 1, 2, 2, 1, 1, 1, 1, 2, 0, 2, 1, 2, 1, 1, 1]’ or, if the source file was a specific single act file for the subject, e.g. ‘D16\_position\_act4.csv’, then it would get the label ‘2’. This process of label extraction is identical across multiple scripts, be it for training models in ‘rnn.py’ or getting the true values of assessing subjects in ‘model\_predictor.py’, and so on.

**Results for Experiment Sets (‘RNN Results.xlsx’)**



(continued)

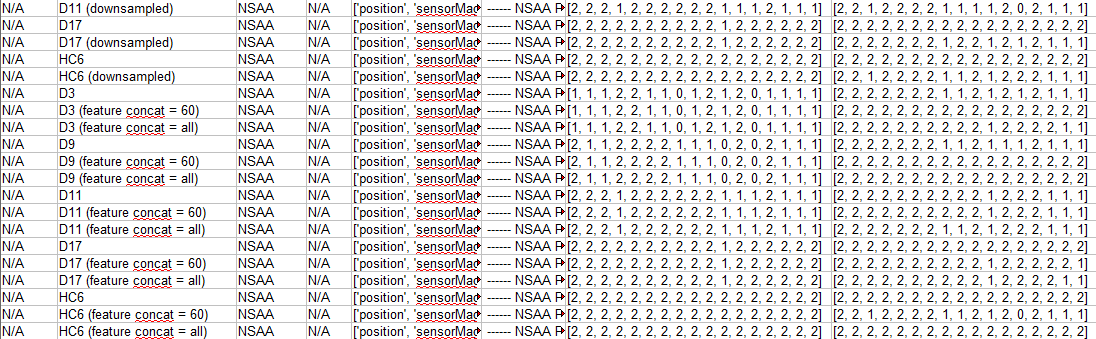


With the two main reference files used by the project covered, we now move onto the first of the two documents containing the results of the various experiment sets and model predictions sets that we run. The first of those, ‘RNN Results.xlsx’, contains the information of building various models within the ‘Experiment Set’s section of the results discussion we cover later on. These include testing different measurements with which to build models, examining different sequence lengths and sequence overlap proportions, and so on. What separates this document from the ‘model\_predictions.csv’ document that we shall discuss shortly is that ‘RNN Results.xlsx’ contains the results obtained from just analysing the testing data sets supplied to the ‘rnn.py’: this is generally 20% of the source data set that is fed into the ‘rnn.py’ script. Hence, the results contained in each cell of the ‘Results’ column for this document covers the performance of various model setups on this testing data. In contrast, for most of the ‘model\_predictions.csv’ sets these are the results of feeding in complete subjects to be assessed on prebuilt models via the ‘model\_predictor.py’ script. Therefore, ‘RNN Results.xlsx’ gets its results from the direct console output of building models in ‘rnn.py’, while ‘model\_predictions.csv’ gets its results from running ‘model\_predictor.py’ on models already built in ‘rnn.py’.

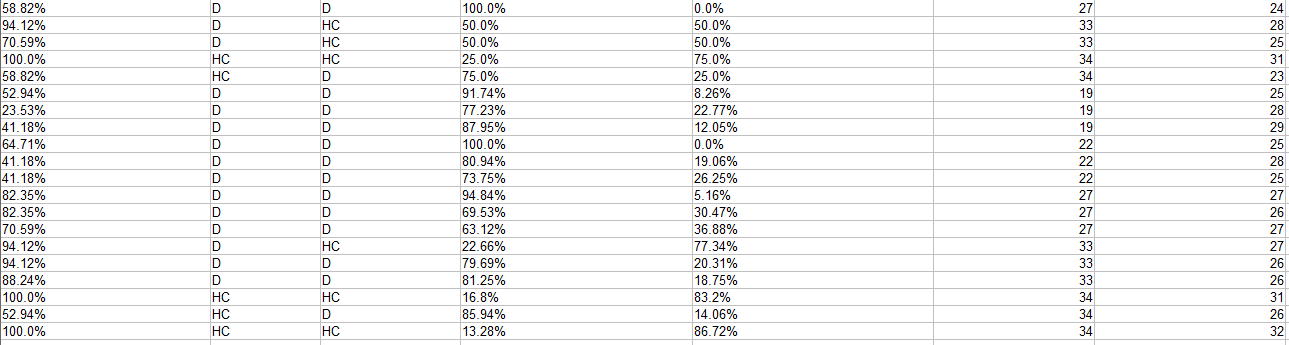
Another difference from ‘model\_predictions.csv’ is that ‘RNN Results.xlsx’ has its information manually inserted into the table, as opposed to automatically done in ‘model\_predictions.csv’ via the ‘DataFrame.writecsv()’ method called in ‘model\_predictor.py’. The reason this is manually done is that, for each model that is created that we wish to write to a row of ‘RNN Results.xlsx’ there is additional information that ‘rnn.py’ has no knowledge about and therefore cannot write to the table. This includes a short description of the model that has been created and what prior scripts were needed to have been run in order to preprocess the data that is required for this model (such as ‘comp\_stat\_vals.py’ or ‘ext\_raw\_measures.py’). Therefore, when we wish to write a row of data to ‘RNN Results.xlsx’, we manually fill in parts of the table ourselves and copy/paste parts of the console output (here, into the ‘Results’ column). Additionally, along with the description, necessary prior script and arguments to have been run, and the results of the model, we also output the general model configuration that includes the shape of the data and several of the more significant hyperparameters. The idea is that anyone observing our results in the results discussion section can refer back to this table fairly easily and see exactly how the models were built.

The way these results from ‘RNN Results.xlsx’ are generally assessed is via the ‘graph\_creator.py’ script. This script reads directly from the file, grabs the requested rows, and plots one or more of the columns against each other depending on the arguments (for example, one configuration of ‘graph\_creator.py’ could plot the results of the MAE of the overall NSAA score against the sequence lengths of the corresponding rows, which we do so in experiment set 8). This is therefore the primary way with which we use the ‘RNN Results.xlsx’ file in this report, though it also serves the purpose of providing results that can be compared to via other users using the system.

**Results for Model Predictions Sets (‘model\_predictions.csv’)**



(continued)



The final document that we make heavy use of in the system is the ‘model\_predictions.csv’ document. This is were any assessments made by the ‘model\_predictor.py’ is stored (and by extension ‘test\_altdirs.py’ which calls ‘model\_predictor.py’ many times) and as such corresponds to the model predictions sets that are discussed later in the experiments and results discussion. As all the information that we wish to write to the file is known by the ‘model\_predictor.py’ at run time, this processed is automated via the script itself and does not require the user to enter information into the table. As the primary differences between the two documents have already been outlined, it just remains to outline the types of information that is recorded in the table along with how we make use of this when assessing various model setups and investigating performance of different variations of subject files.

Each assessment made by ‘model\_predictor.py’ writes one row of information to this table. This contains the name of the subject we are assessing on, along with extra strings that signify different types of models that subject was assessed on or any other preprocessing steps that were taken for the subject file(s) (see the relevant model prediction set descriptions to see to what each of these additional strings correspond). The name of the directory that the subject file(s) are sourced from is also included, along with any other directories that act as alternative directories from which to source files (see model prediction set 1 for an example of this). We also include the different measurements extracted from this subject’s file(s) that were used to act as input to the necessary models (the exact models that are chosen therefore depend on the types of measurement that are extracted for the subject in question). The rest of the columns contain the assessed results of the subject for all the output types we are testing the subject one (which is generally the ‘overall’, ‘acts’, and ‘dhc’ output types). For example, the ‘Predicted ‘D/HC Label’’ reflects the results of the assessing the subject on models built for the ‘dhc’ output type, while the ‘True ‘Overall NSAA Score’’ and ‘Predicted ‘Overall NSAA Score’’ reflects the results of assessing on models built for the ‘overall’ output type. Therefore, we all the information needed from subject assessments on various types of models to create tables comparing different subjects and setups (as shown in many model prediction sets) or to be used via ‘graph\_creator.py’, which can access ‘model\_predictions.csv’ in a similar way to ‘RNN Results.xlsx’ in order to create graphs over various rows of the table, based on the arguments passed to ‘graph\_creator.py’.