# BLADE code documentation

The accompanying code is intended to create longitudinal datasets out of BLADE components with proper R object formatting. The code parses required BLADE variables and imputes missing values with XGBoost. This version of the code was designed for the 2017-18 release of BLADE and may require tweaking for newer releases. All code is subject to the attached licence file (MIT).

The core components of the code are:

* BLADE parser functions (contained in the R folder)
* Initial BLADE parse and missing value imputation model matrix creation
* Missing value imputation model estimation
* Application of model to BLADE

The enclosed files were originally part of a research project into wages and investment but has been repurposed to serve as a starting point for other Commission researchers using BLADE. Researchers using this code should adjust required variables as needed and tweak model parameters to suit their needs.

## Parsing functions

This part requires a bit of understanding of the BLADE structure. The data item list is quite helpful:

["V:\BLADE1617\_CORE\BLADE Core Data Item List\_September\_2019.xlsx"](file:///V:\BLADE1617_CORE\BLADE%20Core%20Data%20Item%20List_September_2019.xlsx)

There are four datasets with each year of the dataset stored in different CSV files. The datasets are BAS (Business Activity Statement), BIT (Business Income Tax), PAYG (Pay as You Go income tax), and Frame (extracts from the ABS business register). For the most part, reading in these datasets is a straightforward task:

1. Parse the CSV files
2. Add a date column
3. Bind rows
4. Recode integer categorical values to strings (as factor)

As you’ll see, there are functions for each dataset in the /R/ directory which does exactly this. The exception to this process is BIT data. BIT data comes in more difficult format where variables have up to four columns each depending on what form a firm filled out. Each variable will have one of four prefix:

* C\_ = Company form
* P\_ = Partnership form
* T\_ = Trust form
* I\_ = Individual form

For most firms, they will only have data values for one form (a company only completes the company form and a trust only completes the trust form). However, some firms fill in multiple. To fix this, the parse\_bit function has a concept of “collapsing”. Depending on the argument chosen, this will take the max or sum of each firm’s value for a particular variable across all of the variables C\_, P\_, T\_, and I\_ entries. BIT also has flags for type of form filled with dates in the variable name, e.g. “BIT\_Comp\_0102” which means that a firm filled a company BIT form in the 2001-02 financial year. The parser functions remove these flags but use them to extract date information.

The R folder also contains the parse\_blade function. This is a helper function that automatic regex searches a vector of names of all BLADE datasets and constructs a merged panel dataset based on the inputted variable names. It matches names to both their literal interpretation and an interpretation of BIT variable names without the C\_ P\_ T\_ I\_ prefixes. Merging BLADE datasets is memory intensive. To help fit it in memory, parse\_blade() contains a function called get\_merge\_rm(). The idea behind this function is that R cannot hold every BLADE dataset in memory and merge a complete BLADE at the same time. This function removes a dataset after it is merged to the complete BLADE dataset to conserve memory.

## Cleaning and missing value imputation

Most of the cleaning is done in the parser functions but any cleaning done in files 1, 2 and 3 are all reasonably straight forward (i.e. filtering out 0 values). Missing value imputation was another problem of memory capacity. The code uses the XGBoost algorithm which is efficient but it has limits. A core problem is that it only accepts matrices, not data frames. This means researchers have to binary code all categorical variables, making a long dataset wide too.

To complete the task, in script 1 chunks the data into 2 matrices per dependant variable – test and train, then saves them to disk. Researchers may notice that the code uses FST files in this analysis because it is the only storage format fast enough for this task. Each imputed variable has a formula, not because the imputation needs it, but because model.matrix is the easiest way to binary code factors and it uses formula syntax.

In script 2, the code:

1. Reads in each an imputed variables test and train matrices,
2. Estimates a model
3. Saves the model, fitted Y values for test and actual Y values for test
4. Purges everything from memory
5. Moves on to the next imputed variable.

Script 3 is mostly a copy of script 1. The only difference is at the end, the code estimates a saved model against data which is missing a Y value instead of simply saving a matrix.

## Contact

For further information on this code or using BLADE with R, please contact Henry McMillan at henry.mcmillan@pc.gov.au