Submission Guidelines

1. Introduction

Please read these submission guidelines in full. We will be using your code to do a multiverse analysis in which all valid combinations of analytical choices made by all participants will be combined. **It is therefore essential that your code is well written and documented.** If you are unclear about any of these guidelines or you feel your analysis cannot easily be accommodated, we will:

- provide templates/example submissions (available shortly);
- run a "drop-in" day in Bristol (and elsewhere if there is interest) where teams can ask any submission related questions;
- be available to assist with any queries (please contact maps-project@bristol.ac.uk).

These guidelines should be followed, however they are not exhaustive as we have no way to anticipate what teams will do. Please contact us if anything is not clear or you have suggestions.

We do not require you to perform your own multiverse analysis but for those of you who are interested, the paper describing this process is here:
https://www.ncbi.nlm.nih.gov/pubmed/27694465

This document is organised as follows:

Section 2. Overview of data analysis and code submission gives an overview of the process from receiving these guidelines to submitting your *Final Analysis Report* and accompanying code.

Section 3. Submission Process gives more detail on the process for submitting your code when you have finished your analysis.

Section 4. Code Submission Format gives specific details on how to format your analysis code for submission.

2. Overview of data analysis and code submission

We recommend you follow the following workflow.

1. Read through all of the *Final Analysis Report* and these guidelines before attempting any data analysis.

2. Do your analysis in any way you please. Use as many scripts, notebooks or other tools as you feel necessary. You will not need to submit these (although we strongly encourage you to post these files on the OSF or similar e.g. Github).
3. Once you have come up with an analysis strategy, re-factor (re-write) your code to conform to the format specified in **Section 4. Code Submission Format** of this document.
4. You will then submit *just the code* to us, via email, to ensure your code can be run on another machine. This will continue until your code runs without problems on our machines.
5. Once this is working fill out the *Final Analysis Report* and submit it, along with your checked code.

Points 4 and 5 of this workflow and described in more detail in **Section 3. Submission Process.**

3. Submission Process

The submission process will be in two stages.

Stage 1 - checking your code.

1. Email your code submission (File 1 and 2 below) to maps-project@bristol.ac.uk
2. We run your code on the synthetic data and email back the results.
3. If our results don't match yours or it fails to run please alter your submission.
4. Steps 1 - 3 will be repeated until your code runs and gives your anticipated results.

Stage 2 - submitting the report.

1. complete the *Final Analysis Report* and submit the checked File 1 and FIle 2.
2. We will run your code on the real data and report back the results.

Note: YOU ARE ALLOWED TO MAKE A SUBMISSION TWICE. This is to accommodate you if you get back the results of your analysis on the real data and want to make additional changes. We will keep both submissions.

4. Code Submission Format

Your code submission should consist of two files in a folder:

1. File 1: this should contain your entire analysis in one of the following type of files.
    a. R (.R)
    b. RMarkdown (.Rmd) (please **do not *knit*** these to HTML/PDF/Word etc.)
    c. Python (.py) files, or
    d. Jupyter notebooks (.ipynb)
2. File 2: a *session information file* as described below.

File 1 should be named:

- final_analysis_YOURTEAMNAME.R/Rmd
- final_analysis_YOURTEAMNAME.py/ipynb

File 2 should be named:

1. session_info_YOURTEAMNAME.txt (R)
2. requirements_YOURTEAMNAME.txt (Python)

Where ‘YOURTEAMNAME’ should be replaced by your actual team name **with no spaces or special characters, only underscores (‘_’)**.

4.1 File 1: the final analysis

The following sections are guidelines on how to format file 1.

4.1.1. Writing functions

We have anticipated a number of “chunks” that may form part of teams analysis. For each of these chunks we would like you to define a function, which is named as described in the table below. If you don’t usually write functions, don’t worry, this is exactly the type of thing that we’re happy to help with and will provide examples of how to do.

Note: ALL CODE MUST BE CONTAINED IN FUNCTIONS

Analysis chunks	**Function name and argument**	**Return variable**
Loads the data from a given path	`load_data(path)`	`data`
Transformations (e.g. scaling, logs, normalising)	`transformation_XXX(data)`	`data`
Outliers	`outliers_XXX(data)`	`data`
Missing values	`missing_XXX(data)`	`data`
Definition of outcome (“depression”)	`depression_XXX(data)`	`data`
Definition of exposure (“computer use”)	`computer_use_XXX(data)`	`data`
Any additional chunk we didn’t foresee	`additional_XXX(data)`	`data`
Specifying and fitting the model	`specify_model(data)`	`data, results`

`path: ‘maps-synthetic-data.csv’`
`data`: a data frame
`results`: a *named list* or *dictionary* containing the following:

- “mod” = model object
- “or_1” = your first odds ratio

- “p_1” = the p-value corresponding to “or_1”
- “ci_1”= the 95% confidence interval corresponding to “or_1”)
- “or_2” = your second odds ratio
- “p_2” = the p-value corresponding to “or_2”
- “ci_2”= the 95% confidence interval corresponding to “or_2”
- “AIC” = the AIC value
- “DIC” = the DIC value

Some rules for writing chunk functions:

1. **Functions should alter the dataframe.** At each stage of the analysis, the dataframe should change in some way.

2. **Use the function names provided.** Where XXX is replaced with a zero-padded number representing how many times that particular type of function has appeared in the model, e.g. `transformation_001()` for the first transformation, `missing_001()` for the first function dealing with missing values.

3. **Make your functions as large as possible.** Functions should represent the biggest chunk of analysis they can.
   For example, if you can do all your transformations at one point in your code, then it only needs to be one `transformation_001()` function. It’s possible for there to be one (or none) functions of each type in an analysis.

4. **Be verbose:** Where appropriate, do operations individually per column.
   For example, if you log scale two columns of your dataset, write two lines of code to do this and comment each one individually. This will allow us to mix and match your analysis more easily with other team’s analysis.

5. **Use indentation:** In Python this is enforced. In R, please use indentation for all functions, loops, conditional (if/else) constructs.

6. **Please include any calls to libraries/packages inside the function.** This is because we will be running the functions independently of each other. This might mean that you need to put `library(tidyverse)` in every function, if you use it consistently, for example.

7. **Function arguments.** Each of these functions should conform to the call and return signatures in the table above.

8. **Functions should be self contained.** Any parameters should be defined inside the function and each one should not call any other functions.

9. **The same function may feature multiple times.** For example, if you would like to do the same transformations before and after some other step.

10. **Arrange your functions in the order you would run them.** They don't have to be in the order of the table above.

11. **Comment your chunks as much as possible.** Place general comments on separate lines. Comments specific to variables should go at the end of the line. There should be approximately equal amounts of code and comments!

12. **Bayesian analysis in R.** If you're using jags/bugs or stan to run a Bayesian analysis, please contact us.

4.1.2. Variable naming conventions

4.1.2.1 Mandatory variables

Some variables should have specific names and be defined in specific places.

- The variable referring to the dataframe containing the data should be called `data.`
- The variable in the dataframe referring to your **depression** variable(s) should be called `depression` (or `depression_1` `depression_2` if you're running a multivariate/multiple output model).
- The variable in the dataframe referring to your **computer use** variable(s) should be called `comp_use` (or `comp_use_1`, `comp_use_2` if you have more than one).
- Please define a variable called TEAM_NAME at the top of your analysis script.
- Please define a variable called VERSION at the top of you analysis script (see below).

TEAM_NAME = your team name as a lower case string - should match your final analysis script file name.
VERSION = the version of R/Python you are using copied **exactly** from the output of the functions below:

In R:

```
version$version.string
```
```
## [1] "R version 3.5.3 (2019-03-11)"
```

In Python:

```
import platform
platform.python_version()
```

4.1.2.2. Optional variables

Optional variables can take the form:

1. New columns in the dataframe.
2. User defined parameters

New columns of data can be named however you wish (except for the depression and computer use variables) however please prepend these new variables with ‘`n_`’. E.g. If you create a new variable for the Body Mass Index please call is `n_bmi`.

Where there are user-defined parameters, these should be:

- Defined in SHOUTING_SNAKE_CASE at the beginning of each function.
- Commented on the same line with a short description.
- The comment should contain a minimum and maximum value this parameter could take. If the parameter can on take on a single value put the min and max to be the same. E.g.

  ```
  MIN_GROUP_SIZE = 12 #Minimum number of people in a group (min: 4,
  max: 15)
  ```

We will be using the definitions in the multiverse analysis so please make them conform to this format. If you need to explain something in more detail please put comments above the variable.

4.1.3. Example function

Here’s an example in Python:

```
def depression_001(data):
"""This function defines depression using 'has_dep_diag' and
'dep_thoughts', weighted towards has_dep_diag.
"""

      #IMPORTS:
      import pandas as pd

      #PARAMETERS:

      DEP_DIAG_MULT = 10  # this number is used to scale the depression
diagnosis by this number because we think it is this many times more
important than number of depressive thoughts (min:1, max:30)

      #DEFINE DEPRESSION:
#na is filled to zero to give a minimum threshold
data['depression'] = DEP_DIAG_MULT*data['has_dep_diag'] + \
data['dep_thoughts'].fillna(0)

return data
```

Here's the equivalent in R:

```r
depression_001 <- function(data){
"""This function defines depression using 'has_dep_diag' and
'dep_thoughts', weighted towards has_dep_diag.
"""
      #IMPORTS:
      library(tidyverse)

      #PARAMETERS:
      DEP_DIAG_MULT <- 10  # this number is used to scale the depression
diagnosis by this number because we think it is this many times more
important than number of depressive thoughts (min:1, max:30)

      #DEFINE DEPRESSION:
      #na is filled to zero to give a minimum threshold
      data$dep_thoughts <- data$dep_thoughts %>% replace_na(0)
      data$depression <- DEP_DIAG_MULT*data$has_dep_diag +
data$dep_thoughts

      data

}
```

4.1.4 Example file 1

This script shows you structure of the file 1 in Python (the structure will be the same for R):

```python
TEAM_NAME="nat_and_rob"
VERSION='3.7.1'

def load_data(path):
      ...code...
      return data

def depression_001(data):
      ...code...
      return data

def computer_use_001(data):
      ...code...
      return data

def transformation_001(data):
      ...code...
      return data

def specify_model(data):
      ...code...
      return data

path = 'maps-synthetic-data.csv'
data = load_data(path)
data = depression_001(data)
data = computer_use_001(data)
data = transformation_001(data)
data, results = specify_model(data)
```

4.2 File 2: session information file

In addition to an analysis file, we also ask for a file describing the R/Python packages you have used. These are easy to create:

1. Restart your R/Python session and run your final analysis script.
2. Make sure it give you the answer you're expecting.
3. Now run the following command:

In R

```
writeLines(capture.output(sessionInfo()),
"session_info_YOURTEAMNAME.txt")
```

In Python (Mac/Linux)

You can create a Python requirements file using https://github.com/Damnever/pigar:

```
$> cd YOURSUBMISSIONDIRECTORY
$> pip install pigar
$> pigar
$> mv requirements.txt requirements_YOURTEAMNAME.txt
```