Creating a draft Watershed Assessment Report

Carl Schwarz

2017-05-06

1 Creating a draft Watershed Assessment Report

1.1 Introduction

A draft watershed assessment report will be created in MSWord using data extracted from the FWIS database. This draft report can then be edited by tweaking the R code (preferred) or tweaking the MSWord document (not preferred because your changes will be lost when you update the report by rerunning the *R* code).

This draft report is created using *R* and *JAGS* (a program that does Bayesian computatons) to compute summary statistics, compute the probability of belonging to the FSI categories, and make plots (using *ggplot2*). *RMarkdown* is used to mingle text and graphics and is rendered into a MSWord document.

This process follows the principle of Reproducible Research, where a single document contains ALL of the code used to analyze the data and creates a report based on this. It is hoped that when each Watershed Assessment is updated, it will be necessary only to rerun the code and do some minor teaks to the report.

1.2 Getting your computer ready

Your computer will need the following software. Instructions for installing the software can be found at http://people.stat.sfu.ca/~cschwarz/CourseNotes/HowGetSoftware.html.

- *R*. This is the statistical software used create summary table and to create plots. Don't forget to install the packages (add ons for *R*) as listed in item (6) of the installation instructions in the above link. You will also need to install the following packages:
  - *stringr* specialized functions for text processing
  - *R2Jags*, *coda*, *digest* needed to run JAGS
  - *pander* needed to create tables in *RMarkdown*
  - *ggmaps* needed to extract Google maps over the web. You will needs the most recent version which has NOT yet been released to CRAN. You need to install it using
    - devtools::install_github("dkahle/ggmap") on the R console. This may require the *devtools* package to also be installed.
- *RStudio*. An integrated development environment that controls *R* and its output.
- *JAGS* (Just Another Gibbs Sampler). A cross-platform program that performs Bayesian computations. This program will be called by *R*. Don't forget to install the additional *R* packages listed under item (2) of the *JAGS* installation instructions in the above link namely:
  - *R2Jags*, *coda*, and *digest* needed to run JAGS
- *JAVA*. In order to read the FWIS files directly you will have to authorize *JAVA* programs on your computer.

1.3 Getting the FWIS data

Information on each watershed is extracted from the FWIS database. Contact ????? for further details.

The current extraction program creates Excel 95 files. *R* requires workbooks to be in Excel 97 or later format, so you will have to open and save the file as Excel 97 or later format.

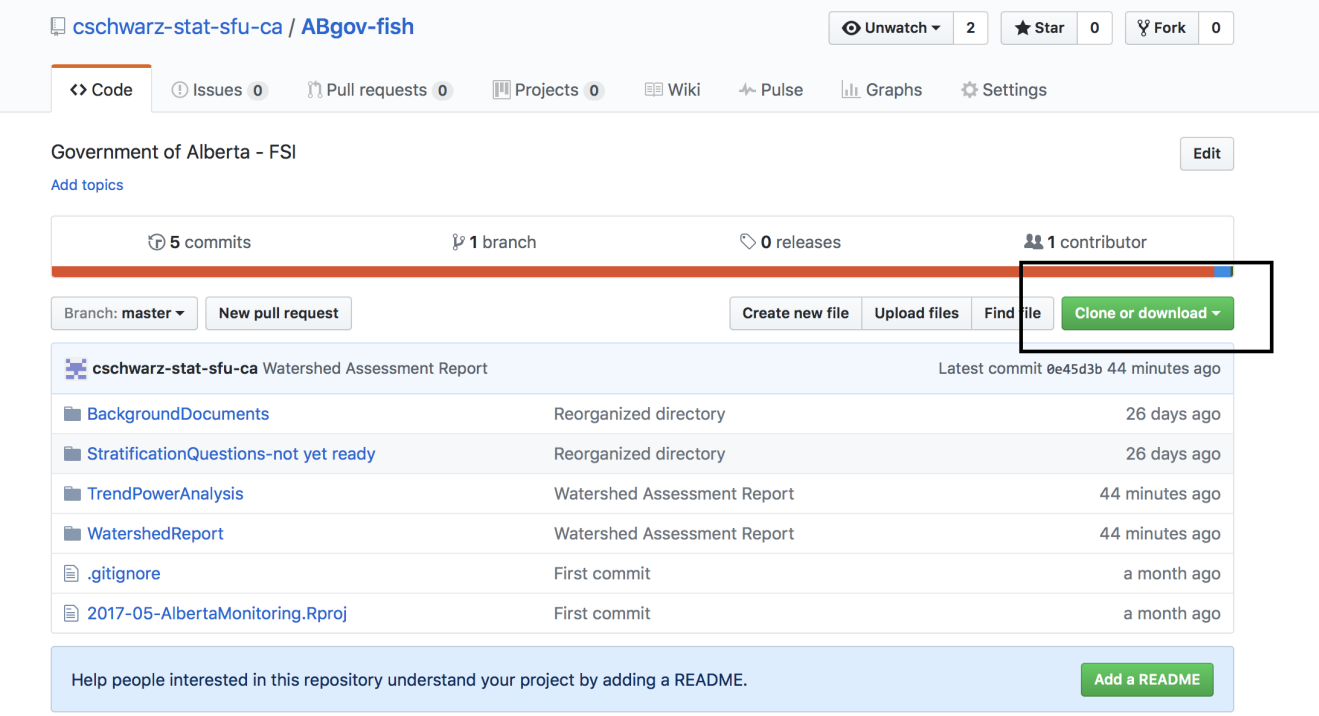
1.4 Create a working directory to hold the data files and the draft template.

Create a working directory which will include the *RMarkdown* files used to create the draft watershed report and any data files, image files, etc. needed for the report.

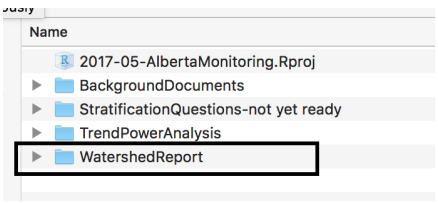
1.4.1 Copy the *RMarkdown* template files to the working directory.

The template files are located on the *GitHub* server and can be retreived as follows.

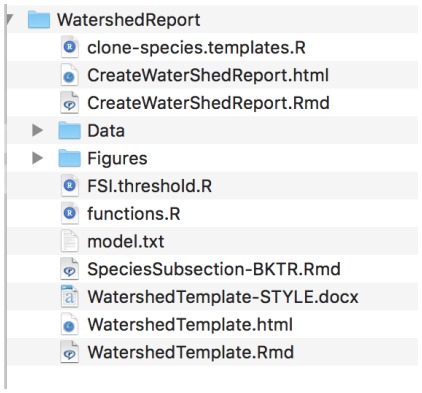
- Go to https://github.com/cschwarz-stat-sfu-ca/ABgov-fish The web page will look similar to:



- Click on the *Clone or Download* button to download a zip file of the entire contents.
- Move the zip file to your working directory.
- Unzip the zip file.
- Discard the zip file.
- Your working directory should now have several subdirectories:

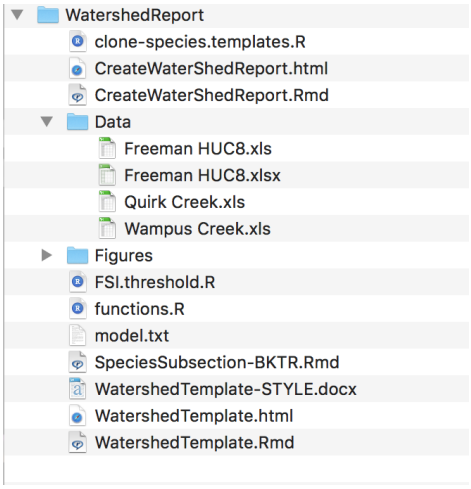


- Open the *WatershedReport* directory



1.4.2 Move the FWIS data file to the *Data* directory.

Move the FWIS data file to the *Data* directory. **The data file MUST be in Excel 97 or later format.** The FWIS extraction system currently creates an Excel 95 file. You will have to open it and re-save it as an Excel 97 or later document (either *.xls* or *.xlsx* is fine).

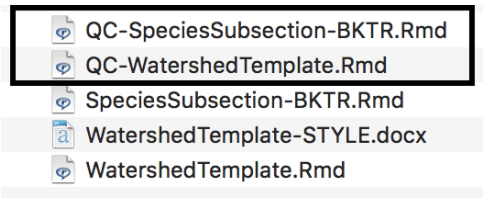


1.4.3 Duplicate and rename the main template file and the species template files.

We plan to analyze the *Quirk Creek* data and so we will rename the main template and species template files by adding a prefix (e.g. QC-) to the *WatershedTemplate.Rmd* and *SpeciesSubsection-BKTR.Rmd* files.

DO NOT MODIFY THE NAME OF THE WatershedTemplate-STYLE.docx file as this is a style template for the *MSWord* reports used by *RMarkdown* when creating the draft watershed report.

You will now have two new files (copied and renamed) of the template files:



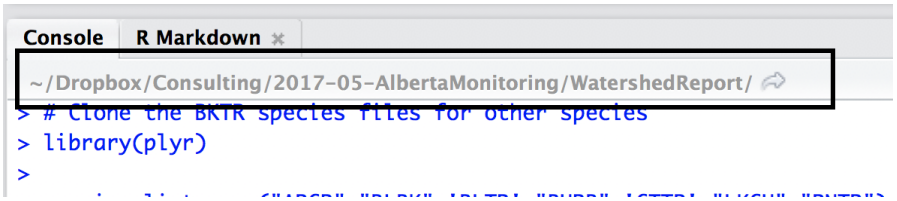
1.4.4 Clone the species template files.

The GitHub site comes with a single species subsection template and you will have to clone this for the other species of interest in the watershed report. A clone is identical to the template for BKTR except all occurances of BKTR are replaced by the species code of interest (e.g. CTTR). This cloning can be done by making duplicates of the BKTR template and changing the name of the file and, then within the file, all occurances of BKTR to the new species code.

I have provided a file, *clone-species.templates.R*, that can do this for you. **Be careful not to overwrite any species template files that you have modified as part of the assessment.**

For example, suppose we want species template files for BKTR (already present), BLBK, BLTR and CTTR.

- Open the *clone-species.templates.R* file using *RStudio* (and NOT R).
- **BE SURE YOU ARE POINTED TO THE CORRECT WORKING DIRECTORY** by examining the name on the console window of *RStudio*. (Your directory name will differ from mine shown below.)

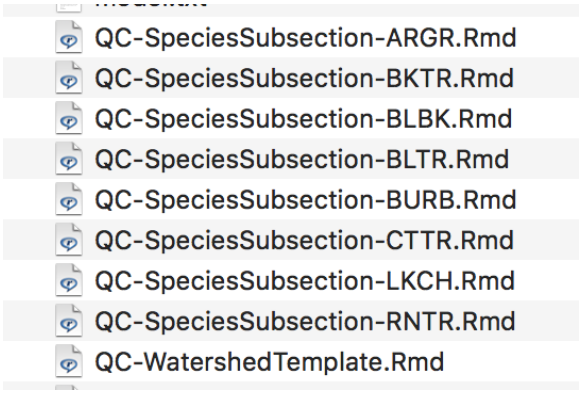


- The Rstudio script pane will have the *R* code from the *clone-species.templates.R* file and will look similar to :

```
# Clone the BKTR species files for other species
library(plyr)
species.list <- c("ARGR","BLBK",'BLTR',"BURB",'CTTR',"LKCH","RNTR")

l_ply(species.list, function (x){
   # read the BKTR SpeciesSubsection, change all BKTR to the species list, and write out new file
   filename <- 'SpeciesSubsection-BKTR.Rmd'
   template <- readLines(filename)
   template <- gsub("BKTR", x, template, fixed=TRUE)
   filename <- gsub("BKTR", x, filename, fixed=TRUE)
   writeLines(template, filename, sep='\r\n' )
})
```

- Modify the list of species codes.
- Change the name of the template file *SpeciesSubsection-BKTR.Rmd* to *QC-SpeciesSubsection-BKTR.Rmd*
- Run the code. Your directory should now be populated with copies of the species template files for this watershed.



1.4.5 Check the FSI category boundaries for the species of interest.

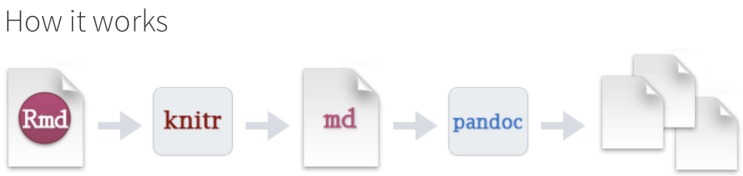
The FSI category boundaries are stored in the *FSI.threshold.R* file. This can be edited directly in *RStudio* or any other text editor.

1.5 Editing the main template file.

1.5.1 Overview of process

Each watershed report will have one main template file (*QC-WatershedTemplate.Rd* in our case) and one or more species template files (*QC-SpeciesSubsection-BKTR.Rmd* etc.) that are *knitted* (in *R* parlance) together. These template files contain both text and *R* code as a living document so that as the data change, the report is updated when the the template are again knitted together.

The templates are written using *RMarkdown* http://rmarkdown.rstudio.com which provides a simple set of commands to knit together *R* code and (formatted) text as shown below:



When you wish to render the document, *R Markdown* feeds the *.Rmd* file to *knitr*, which executes all of the code chunks and creates a new markdown (*.md*) document which includes the code and it's output in a general way.The *pandoc* program (included with *RStudio*) converts the markdown document to the final format (HTML, PDF, MSWord, etc.). While the above seems complicated, most of it is hidden from you and occurs "behind the scenes."

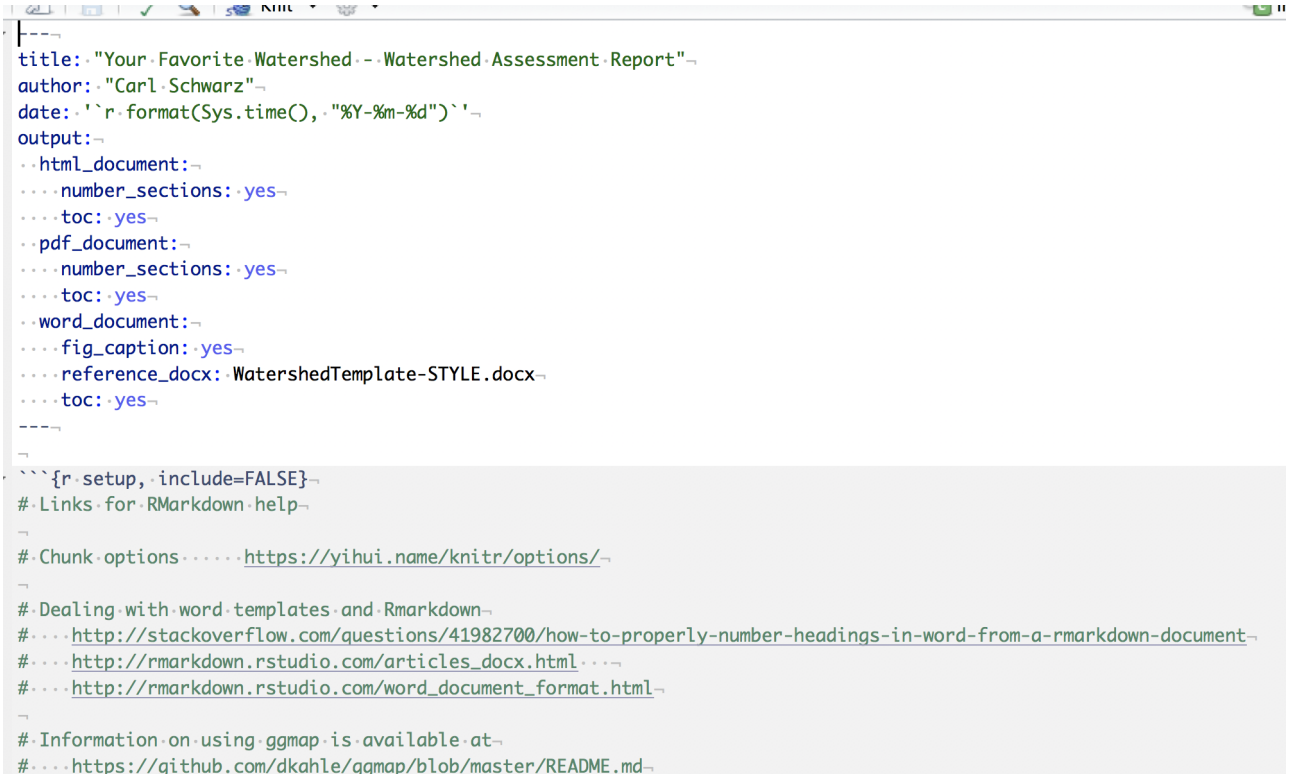
Creating the draft watershed reports will follow a multi-step process.

1. Read in the data from the FWIS system (the workbook you copied to the *Data* directory) and create some prelimnary tables and plots to check the data, remove and outliers, or do any other pre-processing. This preprocessing is 'hidden' from the draft watershed assessment report, but the code used to pre-process the data is stored with the document. Consequently, when you update the assessment in a later year, the decisions made in pre-processing the data are evident.
2. Make some summary tables and plots for the wateshed as a whole. The current template makes some simple tables of counts of fish species by year etc and produces a map showing the sampling locations. You are able to change the default figures and tables by modifying the corresponding *R* code.

3. For each species of interest, the corresponding species template file creates some summary charts and figures. It then calls *JAGS* to estimate the probability that the species is in each FSI category accounting for process and sampling error over time. Summary figures are presented which again can be changed by modifying the *R* code. I would suggest that you analyze one species at a time until all are working to your satisfaction.

4. Finally, you can enter summary text describing the state of this watershed.

Open the main template file (*QC-WatershedTemplate.Rmd* for this example) in *RStudio*. The script pane will look similar to:



The template file consists of 3 types of material:

- The headers (between the —'s) which identifies the watershed assessment name, author, date of creating, and list the options to be applied when the document is rendered. In this case, change the name of the title to a more appropriate name along with the author. The date uses *R* to extract the date the code is run and formats it as YYYY-MM-DD. The options for a MSWord rendering indicate that the *WatershedTemplate-STYLE.docx* file is to be used for styles when creating the document (this style document is where you specify that the sections are to be numbered). The headers only occur once in a template at the top of the primary file.

- Code chunks. These are sections of *R* code of the form delimited by triple BACK quotes with *R* code in between.

  ```
  ```{r chunkname, options}
     R code here
  ```
  ```

  Each chunk has a number of options specifying if the output is to be displayed in the report (e.g. a plot), or computations are proceed with nothing placed in the report. A list of chunk options is available at: https://yihui.name/knitr/options/. There is no limit to the number or types of chunks you can use.

- Text (scroll down the script window to see):

```
173
174 # Background
175 "How are the fish in my river and streams doing?"
176 We need this answer to set appropriate fishing regulations,
177 to understand and correct any problems with fish habitat and
178 to guard against invasive species.
179
180 A healthy fish population and fish community means we
181 can all enjoy the benefits of sustainable fisheries
182 and healthy ecosystems.
183 A standard method of assessing the status of
184 fish populations is necessary to allow comparisons of fish sustainability
```

which is written using *R Markdown* (refer to https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf). There is no limit to the amount of text you can use.

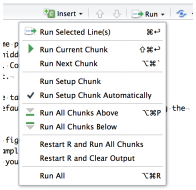
The text and code chunks can be intermixed at will.

1.5.2 Reading in the data and editing the data

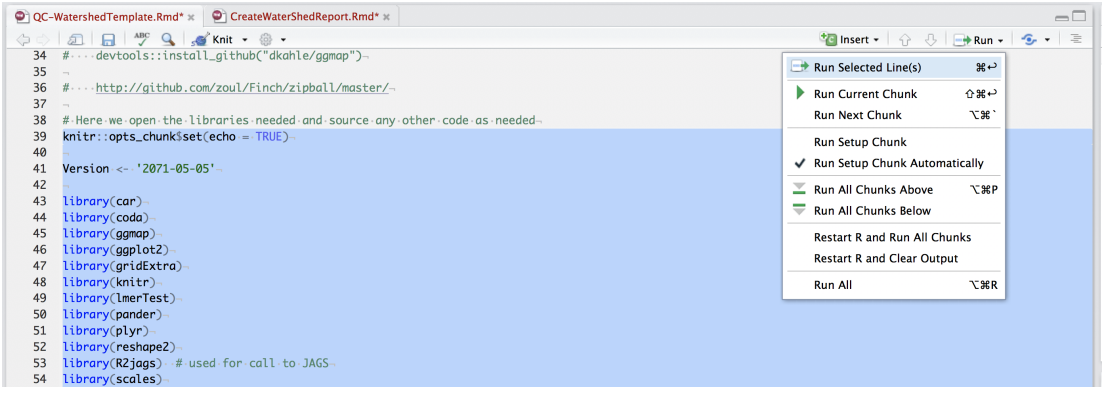
Return to the main template file (*QC-WatershedTemplate.Rmd* for this example) in *RStudio*.

The first chunk (labelled *setup*) is where we set up the *R* code by opening the libraries needed for the analyses, read in the data file, and do any data editing. The option *include=FALSE* implies that any of the output from this chunk will NOT be included in the draft report.

You can execute all of the *R* code in an entire chunk by clicking on the *Run* pop-down menu of *RStudio* and selecting the appropriate option.

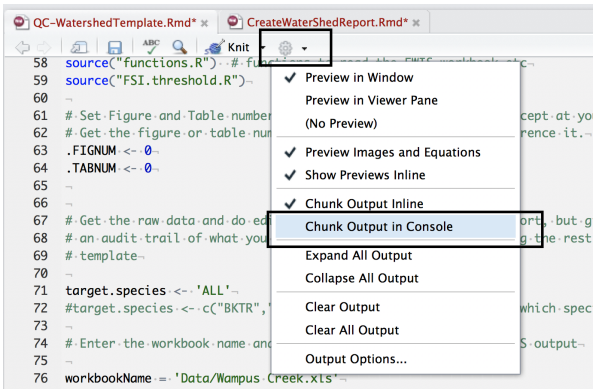


Or you can run one or several lines at a time by selecting, hightlighting, and then using the *Run* pop-down menu:



The keyboard short cuts (Command-Enter for Macintosh; Cntr-R for Windoze) can also be used.

You will find it useful to send all output to the console window during the template setup using:



1.5.2.1 Select which species you want to report on.

About 20 lines down the first chunk, you will see the code fragment:

```
target.species <- 'ALL'
#target.species <- c("BKTR","BLTR","MNWH") # or you can select which species to include
```

If you have particular species in mind, you can select them here by specifying the species codes (in any order) or using *'ALL'* to select all species from the FWIS worksheet. Even if you specify that all species are to be read in from the FWIS workbook, you have the option of selecting which species will have the FSI categorization done later in the code.

In this case, we will leave the defaults as is. Notice the *R* comment indicator (#) in the second line of the code fragment implies that this line is ignored.

1.5.2.2 Specifying the file name of the FWIS workbook.

The next code fragment is where the file name of the FWIS workbook is specified along with the sheetname containing the data.

```
workbookName = file.path('Data','Quirk Creek.xls')
sheetName    = 'F&W'
```

Here the workbook is located in the *Data* subdirectory in the current working directory. The sheetname is "F&W".

1.5.2.3 Read in the data

The next code fragment checks that the file exists and then calls a function to read the FWIS workbook.

```
if(!file.exists(workbookName))stop(paste("File ",workbookName,' not found '))

# read the sheet from the workbook
cpue <- read.FWIS.workbook(
        workbookName  =workbookName,
        sheetName     =sheetName,
        target.species=target.species)
```

Highlight and run all of the code block from the start to the end of the fragment above.

If the read is successful, you should get some information lines in the console:

```
##
##
```

```
##
## *** Starting to read data from  Data/Quirk Creek.xls *****
##     Worksheet  F&W
## total rows read  5381 44
```

1.5.2.4 Check and edit the data

The next sections of code creates summary tables to check the data (e.g. invalid species code, missing species code); create cross-tabulations by years collected, the type of equipment used (e.g., backpack electrofishing) etc. Here is where you can exclude data, for example, from angling surveys that are not amenable to the CPUE computations. Some of the output is shown below:

```
# what years are present in this workbooks?
xtabs(~Year, data=cpue, exclude=NULL, na.action=na.pass)
```

```
## Year
## 1987 1988 1998 1999 2000 2006 2007 2008 2009 2010 2011 2012 2013
##  196    8   36   25   89 1642 1099  743  257  389  446  249  202
```

```
# what species are present in this workbook
xtabs(~Species.Code, data=cpue, exclude=NULL, na.action=na.pass)
```

```
## Species.Code
##      BKTR BLBK BLTR CTTR
##    1 2969  138  294 1979
```

Here we remove all non-electrofishing survey types:

```
xtabs(~Survey.Type + Year,     data=cpue , exclude=NULL, na.action=na.pass)
```

```
##                 Year
## Survey.Type      1987 1988 1998 1999 2000 2006 2007 2008 2009 2010 2011
##   Electrofishing  196    7    0    0   38 1642 1099  743  257  389  446
##   Sample Angling    0    0   36   25   51    0    0    0    0    0    0
##                 Year
## Survey.Type      2012 2013
##   Electrofishing  249  202
##   Sample Angling    0    0
```

```
# Remove all non-electrofishing survey types
dim(cpue)
```

```
## [1] 5380   47
```

```
cpue <- cpue[ cpue$Survey.Type %in% c("Electrofishing"),]
dim(cpue)
```

```
## [1] 5268   47
```

```
xtabs(~Survey.Type + Year,     data=cpue , exclude=NULL, na.action=na.pass)
```

```
##                 Year
## Survey.Type      1987 1988 2000 2006 2007 2008 2009 2010 2011 2012 2013
##   Electrofishing  196    7   38 1642 1099  743  257  389  446  249  202
```

Data editing proceeds untils you are satisfied that all of the data should be included in the subsequent analyses.

1.5.2.5 Modifying the introductory text.

After the first code chunk, some introductory text is given:

> # Background
> "How are the fish in my river and streams doing?"
> We need this answer to set appropriate fishing regulations,
> to understand and correct any problems with fish habitat and
> to guard against invasive species.
>
> A healthy fish population and fish community means we
> can all enjoy the benefits of sustainable fisheries
> and healthy ecosystems.
> A standard method of assessing the status of
> fish populations is necessary to allow comparisons of fish sustainability

All text is written using *RMarkdown* https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf and there is plenty of help available on the web (thanks Google!). Text can be entered free form, i.e. you can break lines anywhere in a paragraph and it will be flowed together, EXCEPT if a line has two or more space characters at the end, in which case it is treated as line break.

A hash mark (#} indicates a level 1 header; two hash marks (##) indicate a level 2 header and so on.

Text is italiczed by enclosing with underscores, e.g.

> Westslope Cutthroat Trout (_Oncorhynchus clarkii lewisi_),
> Bull Trout (_Salvelinus confluentus_),
> Arctic Grayling (_Thymallus arcticus_),
> Athabasca Rainbow Trout (_Oncorhynchus mykiss_),

In some cases, placeholders are present where you need to modify the text as information is not available from the FWIS system. Here the number of potential sampling locations (not the actual number taken) must be entered manually replacing the 99999999 placeholder. The ** indicate boldfaced text to make it standout in the draft report in case you forget to change this.

> Within the study area, **999999999** potential sampling locations were randomly chosen using
> ArcGIS (ESRI, 2013) and R (R Core Team, 2015)

You can also include *R* code expressions inline in the text. See the *RMarkdown* manuals for details.

1.5.2.6 Creating a map of sampling locations

The code chunk labelled *WatershedMap*, creates a map of the sampling locations. It uses the range of latitude and longitude (expanded by a small amount) to set a bounding box before calling Google Maps to return a satellite image. Unfortunately, Google Maps returns a map based on the center of the bounding box and a zoom factor – the *get_map* function tries to compute the appropriate factors but doesn't aways succeed. For example, the default zoom factor gives the following map:

```
# Extract the min/max of lat/long and make slightly larger to try and capture entire map
lat  <- range(cpue$Latitude,  na.rm=TRUE)
long <- range(cpue$Longitude, na.rm=TRUE)
# Exapnd these slighly
lat  <- lat + c(-1,1)* (lat[2]-lat[1])/5
long <- long+ c(-1,1)* (long[2]-long[1])/5

# get the map
boundaries<-c( long[1], lat[1], long[2], lat[2])

# get the map from google. You can fiddle with the zoom to get the right scale
```

```
#cc <- get_map(cc.boundaries, maptype="terrain",  source="google")#zoom=13)
google.map <- get_map(boundaries, maptype="satellite",  source="google") #, zoom=11)
```

```
## Warning: bounding box given to google - spatial extent only approximate.
```

```
## converting bounding box to center/zoom specification. (experimental)
```

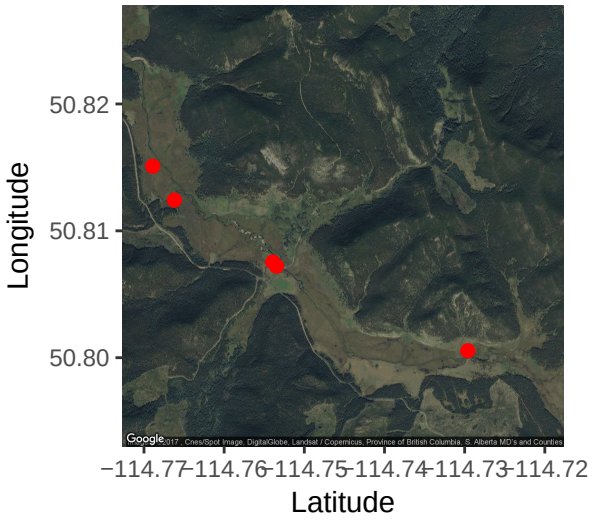
```
## Source : https://maps.googleapis.com/maps/api/staticmap?center=50.810411,-114.745223&zoom=14&size=64
```

```
my.map <- ggmap(google.map)

# Exract the sampling points
stations <- unique( cpue[,c("TTM.Easting","TTM.Northing","Longitude","Latitude")])
plot1 <- my.map +
         ggtitle(paste('Figure ', fig.map,". Location of sampling sites on ", cpue$WatershedName[1],sep=
         theme(plot.title = element_text(size = 8))+  # change text size for title
         geom_point(data=stations, aes(x=Longitude, y=Latitude),size=2, color='red',
                    position=position_jitter(h=.0015, w=.0015) )+
          xlab("Latitude")+ylab("Longitude")
plot1
```

```
## Warning: Removed 3 rows containing missing values (geom_point).
```

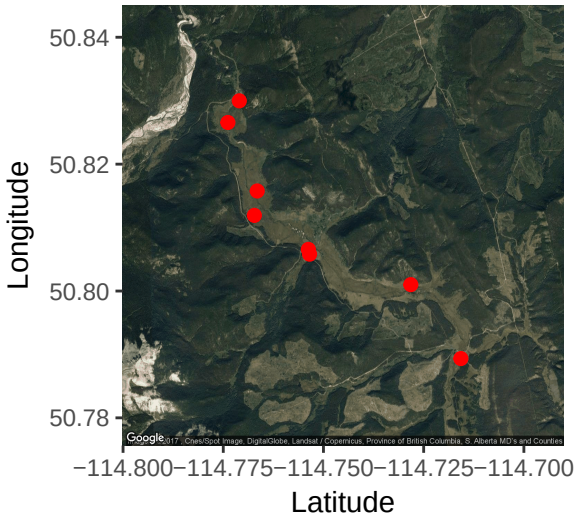
Figure 1. Location of sampling sites on Quirk Creek



Not all of the sampling locations are shown in the returned map. By trial and error, a zoom factor of 13 gives good results:

```
google.map <- get_map(boundaries, maptype="satellite",  source="google", zoom=13)
```

Figure 1. Location of sampling sites on Quirk Creek



Notice how figure and table numbers are generated. You need to execute a code chunk prior to first usage which generates a table or figure number and assigns it to a variable that can be used in the text or in code blocks.

1.5.2.7 Creating a table.

Unfortunately, tables are more difficult to get nicely formatted in *RMarkdown* unless one is using a *LaTeX* rendering engine, in which case you have have very fine control. However, *LaTeX* is likely too complicated for mere mortals.

The *pandoc.table* function does a reasonable job. First the summary statistics are computed in the usual way:

```
# Comput the summary statstics for each species by year
fish.stat <- plyr::ddply(cpue, c("Species.Code"), plyr::summarize,
                  nfish = length(Fork.Length..mm.),
                  fl.mean=round(mean(Fork.Length..mm., na.rm=TRUE)),
                  fl.min= round(min(Fork.Length..mm. , na.rm=TRUE)),
                  fl.max= round(max(Fork.Length..mm. , na.rm=TRUE)))
fish.stat$fl.min[ is.infinite(fish.stat$fl.min)] <- NA
fish.stat$fl.max[ is.infinite(fish.stat$fl.max)] <- NA

# Sort by species
fish.stat <- fish.stat[ order(fish.stat$Species),]
head(fish.stat)
```

```
##   Species.Code nfish fl.mean fl.min fl.max
## 1         BKTR  2912     127     37    327
## 2         BLBK   138     139     82    312
## 3         BLTR   246     188     90    495
## 4         CTTR  1972     132     22    401
```

Then we create the column labels (the \n in the label implies a line break)

```
colnames(fish.stat) <- c(
       "Species\nCode",
       "n",
       "Mean\nfork\nlength\nmm",
       "Min\nfork\nlength\nmm",
```

```
      "Max\nfork\nlength\nmm"
    )
```

and then the *pandoc.table* creates the output to be rendered

```
pandoc.table(fish.stat,
             caption=paste("Table ",tab.fishsummary,
                           ". Summary statistics on species of fish captured in ",
                           cpue$WatershedName[1], sep=""),
             justify='lrrrr',
             split.cells=c(1,1,1,1,1))
```

```
##
## ----------------------------------------
## Species      n     Mean      Min      Max
## Code              fork     fork     fork
##                 length   length   length
##                     mm       mm       mm
## --------- ---- -------- -------- --------
## BKTR      2912      127       37      327
##
## BLBK       138      139       82      312
##
## BLTR       246      188       90      495
##
## CTTR      1972      132       22      401
## ----------------------------------------
##
## Table: Table 1. Summary statistics on species of fish captured in Quirk Creek
```

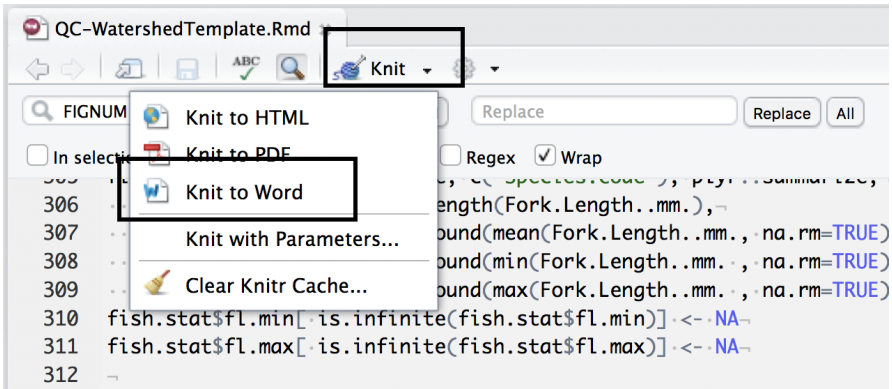
Refer to the help file in *R* for *pandoc.table* for information on the options such as justification (justify='1rrrr' implies the first column is left justified, the second is right etc.), and relative sizes of columns (split.cells=c(1,1,1,1,1) implies that all columns are approximately equal in width).

Notice that the output from *pandoc.table* gets processed into a "nicer" looking table in MSWord, but there is only limited control over formatting tables in basic *RMarkdown*.

1.5.2.8 Run the preliminary report WITHOUT any analysis at the species level.

You are now ready to try to create your first draft assessment, WITHOUT any detailed analyses at the species level.

Use the *Knit* button in *RStudio* to render the document in the format required.



If all goes well, a document in the format requested should be produced. Check the console for errors in either the rendering process or the the *R* code execution process for debugging. It is a bit of an art, but because all chunks are labelled, you can see in which chunk the *R* code failed.

I find it easier to create HTML documents during the debugging phase and leave creating the MSWord creation to the end.

CONGRATULATIONs! You have created your first draft of a watershed assessment!

1.6 Requesting species analysis

Now that you have the main document working, let us work on the species analysis. I would do a single species at a time before rendering the final document.

For species with limited data (e.g. only 3 captured fish), this is a waste of time. Similarly, for species of limited interest, this is also a waste of time.

You need to specify the names of (updated) species template files in the code fragment below. You need to change

```
# Select which species you want assessment for. Comment out those you don't need. Don't forget the comm
#  Don't forget to remove any commas after the final species selected
#child.list <- c(
#        'SpeciesSubsection-ARGR.Rmd',
#        'SpeciesSubsection-BKTR.Rmd',
#        'SpeciesSubsection-BLTR.Rmd',
#        'SpeciesSubsection-BLBK.Rmd',
#        'SpeciesSubsection-BURB.Rmd',
#        'SpeciesSubsection-CTTR.Rmd'
#        'SpeciesSubsection-RNTR.Rmd'
#       )
```

to something like

```
# Select which species you want assessment for. Comment out those you don't need. Don't forget the comm
#  Don't forget to remove any commas after the final species selected
child.list <- c(
   'QC-SpeciesSubsection-BKTR.Rmd'
       )
```

to do one species analysis (in this case for BKTR).

After making this change to the main template document, render it again using the *knit* button in *RStudio*. Note on my Macintosh, the code always ends with an error message about a *segfault*, but the final document has been rendered, so just ignore the error message about an 'In irrecoverable exception occured, R is aborting now.'. The rendered file is in the directory and can be opened in the usual way.

There is really nothing to modify in the species template file (the code is self contained) unless you want to modify some of the plots automatically created or add explanatory text around some of the output. All of the plots are standard *ggplot* objects and so can be edited in the usual way.

The Bayesian analysis returns a list object (*fsi.result*) that you can use to create customized plots or extract information of interest such as the actual probabilities of belonging to each FSI category by year.

Congradulation - almost done. Continue to analyze one species at a time until you are happy with each of the species analyses. Then put all of them together to create the combined report.

1.7 Finish up the summary

Now that all of the species sections are working, update the default summary text and render the final draft document in the format of your choice (MSWord, PDF, HTML, etc.)

1.8 Words of wisdom.

Remember, any changes to the *Rmd* documents will be retained the next time you render the document. Any changes you make to the MSWord document will be lost the next time you render the document. Consequently, try and make the *Rmd* documents as self contained as possible.

Try and include any editing decisions such a removing outliers in the first code chunk of the main template file. In this way, all editing decisions are recorded and will be done the next time the code is run. Remember, you want the template files to be as self-contained and produce reproducible results.

If you get an error message and nothing is rendered, look carefully at the RMarkdown Tab in the console pane of *RStudio* to identify in which code block the error occured. If the error occurs in the Bayesian analysis, you will need to give me a call unless you are a glutton for punishment as debugging JAGS will make you hair turn grey quickly.

It is possible to adust the MSWord Style file – give me a call to discuss what kind of changes you want to the style file. It is not for the faint of heart. Here are some links on editing the style file - Google is your friend.

- http://stackoverflow.com/questions/41982700/how-to-properly-number-headings-in-word-from-a-rmarkdown-document
- http://rmarkdown.rstudio.com/articles_docx.html
- http://rmarkdown.rstudio.com/word_document_format.html

1.9 P.S.

This document was created using -RMarkdown_.