R Minicourse Workshop

Presented to the
Washington State Deptment of Ecology
September 2–3, 2014

Dr. Robin Matthews, Institute for Watershed Studies
Dr. Geoffrey Matthews, Computer Science Department
Western Washington University

Preliminary Schedule

September 2

8:30–11:00	Part 1	Basics, including importing/exporting data, writing/running source files, importing/running packages, extracting information, generating univariate descriptive statistics
11:00–12:00		Questions/one-on-one instruction
12:00–1:00		Lunch
1:00–3:30	Part 2	Plotting, including xy-scatterplots, boxplots, overlay plots, dual axis plots, and basic/intermediate formatting
3:30–4:30		Questions/one-on-one instruction

September 3

8:30–11:00	Part 3	Bivariate analysis, including regression, correlation, ANOVA; testing assumptions; transformations, and nonparametric alternatives
11:00–12:00		Questions/one-on-one instruction
12:00–1:00		Lunch
1:00–3:30	Part 4	Ordination, clustering (hierarchical, divisive, pca), association analysis
3:30–4:30		Questions/one-on-one instruction

Quick Introduction to R

A Very Powerful Calculator

```
 3 + 4*(99 - 2)
[1] 391
 c(1,2,3) + c(100, 200, 300)
[1] 101 202 303
 c(1,2,3) * 10
[1] 10 20 30
 1:9 + 3
[1]  4  5  6  7  8  9 10 11 12
 sin(pi)
[1] 1.224606e-16
```

- My input in red, R's response in blue
- c catenates numbers into a *vector*
- Note that floating point arithmetic is always *approximate*
- Can use arrow keys to recall commands

```
a <- rbind(1:3, 4:6)
a
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
b <- cbind(11:13, 14:16)
b
     [,1] [,2]
[1,]   11   14
[2,]   12   15
[3,]   13   16
a %*% b
     [,1] [,2]
[1,]   74   92
[2,]  182  227
```

- We can name mathematical objects with the <- operator
- cbind binds columns
- rbind binds rows

- Without the %*% operator, multiplication is component-wise:

```
1:5
[1] 1 2 3 4 5
5:1
[1] 5 4 3 2 1
1:5 * 5:1
[1] 5 8 9 8 5
```

```
mydata <- data.frame(1:12, gl(3,4), rnorm(12), runif(12))
names(mydata) <- c("ID", "Group", "X", "Y")
mydata
   ID Group           X         Y
1   1     1 -1.14494014 0.4477421
2   2     1  0.78868969 0.1427814
3   3     1  2.06218090 0.5848855
4   4     1 -0.99460233 0.6999962
5   5     2  0.41466492 0.2406326
6   6     2 -0.81099703 0.3214530
7   7     2  0.54504343 0.1111105
8   8     2 -0.65916912 0.3415629
9   9     3 -0.51886246 0.3548058
10 10     3 -0.02180427 0.8345689
11 11     3  0.51281399 0.5886773
12 12     3 -1.58794920 0.8442706
```

More Readable Numbers

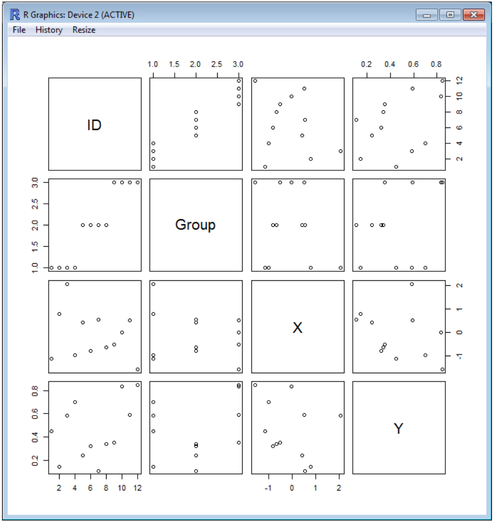
```
options(digits=2)
mydata
   ID Group      X    Y
1   1     1 -1.145 0.45
2   2     1  0.789 0.14
3   3     1  2.062 0.58
4   4     1 -0.995 0.70
5   5     2  0.415 0.24
6   6     2 -0.811 0.32
7   7     2  0.545 0.11
8   8     2 -0.659 0.34
9   9     3 -0.519 0.35
10 10     3 -0.022 0.83
11 11     3  0.513 0.59
12 12     3 -1.588 0.84
```

- R maintains internal accuracy; this is to unclutter the display
- FYI - R uses 32-bit floats with double-precision arithmetic

Quick Introduction to R

Quick Plotting

```
mydata
    ID Group      X    Y
1    1     1 -1.145 0.45
2    2     1  0.789 0.14
3    3     1  2.062 0.58
4    4     1 -0.995 0.70
5    5     2  0.415 0.24
6    6     2 -0.811 0.32
7    7     2  0.545 0.11
8    8     2 -0.659 0.34
9    9     3 -0.519 0.35
10  10     3 -0.022 0.83
11  11     3  0.513 0.59
12  12     3 -1.588 0.84
plot(mydata)
```



Collecting and Entering Data

Quantitative vs. Categorical Data

- Numerical *quantitative* (measured) or *categorical* (grouped)
- Categorical information may be further subdivided:
  - Nominal (groups have no order) . . . red, blue, yellow
  - Ordinal (groups are ordered) . . . low, medium, high
  - Interval (groups are assigned ranges) . . . 0–1, 1–10, 10–100
  - Hierarchical (groups are "nested" in the hierarchy)
- Quantitative data can be used to create categories (e.g., TP $\geq$ 20 μg/L = high; TP $<$ 20 μg/L = low)
- Ordinal and interval data can be ranked for nonparametric tests (e.g., low/med/high = 1/2/3)

Collecting and Entering Data

ASCII Data and Data Set Structure

- Most statistical programs can read **ASCII**† data

  † American Standard Code for Information Interchange

- Many programs can read Excel files if you follow formatting rules
  - Fill all cells with numbers or codes ($\Rightarrow$no empty cells!)
  - Use correct entries for zero (0) and missing data (`R` uses NA)
  - Columns contain variables; rows contain unique samples
  - Do not use spaces in categorical entries or variable names
  - Decide how to handle *out of range* data (see next slide)
  - Enter comments and categorical data in a format that will work for the statistical program, not just Excel
  - Be consistent (NA $\neq$ na $\neq$ n/a $\neq$ N/A; RED $\neq$ Red $\neq$ red)

Collecting and Entering Data

Common Data Set Problems - Out of Range Data

- Out of range data are usually flagged using a nonnumeric code ($<$, $>$, bdl)
- For statistical purposes you need to decide whether to
  - Omit these data
  - Change methods to avoid out of range values
  - Substitute a single value (e.g., one-half the difference between zero and the lowest measurable value)
  - Try to approximate the value using a distribution model
  - Use "uncensored" values (might include negative numbers)
- What you *can't* do is enter "$<$5" with other numeric values because it converts the entire column to a categorical variable

Collecting and Entering Data

Common Data Set Problems - Unbalanced Data Sets

- A balanced data set has the same number of entries for every column (no NAs)
- Multivariate data sets are often unbalanced
  - Samples can be lost, broken, contaminated; some parameters may be measured more frequently than others
- Many statistical programs will not run on unbalanced data unless you specify how to deal with the missing values
- The command `na.rm=T` is used in R to indicate that missing values should be omitted from the calculation[1]

```
x <- c(1,2,3,4,NA)
mean(x)
[1] NA
mean(x, na.rm=T)
[1] 2.5
```

[1] NOTE: Commands will continue in red; responses in blue; > and non-essential output will be omitted

Collecting and Entering Data

Common Data Set Problems - Measurement Units

- Some statistical tests are influenced by measurement units (e.g., from mg/L to μg/L), especially tests based on squared variance
- There is no single solution to this problem, but transformations (log, etc.) and rank-based tests are common approaches

```
a <- c(1000, 2000, 3000, 4000, 5000)
b <- c(4000, 5000, 6000, 7000, 8000)

t.test(a,b)

Welch Two Sample t-test ### edited output
t = -3, df = 8, p-value = 0.01707

kruskal.test(a,b)

Kruskal-Wallis rank sum test ### edited output
Kruskal-Wallis chi-squared = 4, df = 4, p-value = 0.406
```

Exercise #1 - Importing/Exporting Data

Institute for Watershed Studies Northwest Lakes Monitoring Data

Background information: The Institute for Watershed Studies has collected water quality data from more than 65 lakes in northwest Washington since 2007 as a public service and to provide internships for WWU students. A subset of the lakes data are included in lakes.xlsx. The data were edited to exclude rows with missing values and lakes with more than one sampling location. The remaining file contains data from 65 lakes in Whatcom, Skagit, Snohomish, and Island Counties

Summary of variables in lakes.xlsx

site	Short lake names (no spaces!)
month	Month (separate column)
day	Day (separate column)
year	Year (separate column)
chl	Chlorophyll (μg/L) - algae biomass
do	Dissolved oxygen (mg/L) - high oxygen levels indicate active photosynthesis
temp	Water temperature (C) - warm lakes often have more algae
ph	pH - measure of lake acidity; indicates high levels of photosynthesis if >8
cond	Specific conductance (μS/cm) - measure of dissolved salts
alk	Alkalinity (mg/L) - measure of lake buffering capacity (resistance to pH change)
turb	Turbidity (NTU) - measure of lake clarity
nh3	Ammonium (μg-N/L) - algal growth nutrient; Cyanobacteria don't need this
tn	Total persulfate nitrogen (μg-N/L) - inorganic and organic nitrogen
no3	Nitrate/nitrite (μg-N/L) - algal growth nutrient; Cyanobacteria don't need this
tp	Total persulfate phosphorus (μg-P/L) - inorganic and organic phosphorus
srp	Soluble phosphate (μg-P/L) - growth nutrient for all types of algae

Exercise #1, continued

- Open "lakes.xlsx" using Excel and examine the data file
  - The first row contains simple column labels (no spaces)
  - Columns 1-4 contain site and sampling dates (month, day, year)
  - Columns 5-16 contain *uncensored* water quality data
  - Some lakes were sampled only once, while others were sampled multiple years and multiple times within a single year
- Save the file as "lakes.csv" in your R-minicourse working directory
  We will use this file for examples in the next section
- Open an `R` terminal window (double-click on the `R icon`) and change the working directory to your `R-minicourse` directory

Exercise #1, continued

- Read the data into R by typing the following commands

```
lakes <- read.csv("lakes.csv", header=TRUE)

### here is a shorter option:
lakes <- read.csv("lakes.csv", T)

### here is a more versatile option that works with other types of files:
lakes <- read.table("lakes.csv", header=TRUE, sep=",")
```

- Now attach and summarize the data

```
attach(lakes);summary(lakes)  ### note two commands on one line
### "attach" lets you use variable names during the work session
### "summary" lists simple descriptive stats and is a good verification tool
      site          month             day               year              do
REE     : 15   Min.   :1.000   Min.   : 1.00   Min.   :2007   Min.   : 0.500
FAZ     : 12   1st Qu.:6.000   1st Qu.:14.00   1st Qu.:2008   1st Qu.: 8.100
TER     : 12   Median :7.000   Median :20.00   Median :2010   Median : 8.890
BUG     : 11   Mean   :6.741   Mean   :19.17   Mean   :2010   Mean   : 8.903
SQA     : 11   3rd Qu.:8.000   3rd Qu.:26.00   3rd Qu.:2012   3rd Qu.:10.000
SUN     : 11   Max.   :9.000   Max.   :31.00   Max.   :2013   Max.   :21.500
(Other):306
```

etc. for remaining variables; note that the lakes are listed in order of the most frequently sampled, not alphabetically or in the order they are arranged in the data matrix

Extracting Information from a Data Matrix

- One of the goals in statistical analysis is to extract summary information for specific types of measurements
- R has many features to do this ... too many to cover in this class, but we will look at a few examples
- It helps if you understand a few basic features of a typical data matrix
  - A data matrix contains p attributes measured on N samples
  - A matrix denoted $\mathbf{A}_{N \times p}$ has N rows and p columns
  - Individual elements are identified as $a_{i,j}$ where i is the row and j is the column

Simple Data Matrix Example

- This matrix contains four columns, but only the second two columns contain measured data (temperature and oxygen). The first two columns contain categorical information about the sampling location

	column 1	column 2	column 3	column 4
header row	Location	Type	Temperature	Oxygen
row 1	A	stream	10.1	12.5
row 2	A	lake	7.2	4.2
row 3	B	stream	10.5	12.3
row 4	B	lake	6.1	0.1

- The data matrix, therefore, has four rows and two column ($\mathbf{X}_{4\times 2}$)
- Element $x_{1,1}$ is 10.1, element $x_{4,2}$ is 0.1, etc.
- The `lakes.csv` file contains 378 rows and 16 columns, but the first four columns contain site, month, day, year. The water quality data are in columns 5–16

Calculating Simple Descriptive Statistics

Here are some examples for using R to calculate simple descriptive statistics for a single variable (conductivity). Note that the last three examples are slightly more complicated

Function	R syntax	Conductivity example from lakes.csv
Mean	`mean(X)`	`mean(cond)`
Median	`median(X)`	`median(cond)`
Minimum	`min(x)`	`min(cond)`
Maximum	`max(X)`	`max(cond)`
Number of values	`length(X)`	`length(cond)`
Trimmed mean	`mean(X, trim=0.05)`	`mean(cond, trim=0.05)`
Sample variance	`var(X)`	`var(cond)`
Standard deviation	`sd(X)`	`sd(cond)`
Geometric mean	$10^{(\texttt{mean(log10(X))})}$	$10^{(\texttt{mean(log10(cond))})}$
Standard error of the mean	`sqrt(var(X)/length(X))`	`sqrt(var(cond)/length(cond))`
95% conf. interval	`t.test(X)`	`t.test(cond)`

Equivalent Statements in R

- R is a very sophisticated statistics program that includes shortcuts and syntax variations. Each of the following statements will calculate mean conductivity for all sites and dates combined (n=378)
  - `mean(cond)` - This statement works if you attached the variable names (first row) after reading the data
  - `mean(lakes$cond)` - This statement will work with or without attaching the variable names, but you still need `header=TRUE`
  - `mean(lakes[c(1:378), 8])` - This statement works for any data table by specifying the exact rows (1:378) and column (8) that contain the conductivity data

    If there *is* a header line, you need `header=TRUE` because it tells R that the first row doesn't contain data

Selecting Specific Subsets from the Data

- R uses simple arithmetic operators to select specific rows or columns

Selection	R Syntax	Selection	R Syntax
Equal	==	Not equal	!=
Less than	<	Less than or equal	<=
Greater than	>	Greater than or equal	>=
And (all must be true)	&	Or (any can be true)	|

- When using R selection statements, select rows first, then columns, enclosing the selection inside brackets. To select categorical values (e.g., site), enclose the value in quotes

```
mean(cond[year==2013])  ### mean 2013 conductivity, all sites

mean(cond[year==2013 & site !="WIS"])
### mean 2013 conductivity for all sites except Wiser Lake

mean(cond[site=="WIS"])      ### mean 2007-2013 conductivity, Wiser Lake
mean([lakes[c(369:378), 8]) ### mean 2007-2013 conductivity, Wiser Lake
```

Descriptive Statistics for Groups of Data

- You can generate results for groups of variables using `summary(lakes)` (see page 15)
- Another option is to use grouping functions like `tapply`, `lapply`, `sapply`, `by`

```
round(sapply(lakes[c(1:378), c(5:16)], mean), 1)
   do  temp    ph  cond   chl   alk  turb   nh3    tn   no3    tp   srp
  8.9  18.1   7.6 118.0  13.8  35.0   3.9  18.6 684.5 103.5  31.8   7.1

### I selected all rows, but only columns 5-16 ... why?
### here is a shorter way to select all rows: lakes[, c(5:16)]


by(cond, site, summary)
site: ARM
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  53.00   54.50   58.05   59.28   62.82   68.00
------------------------------------------------------------
site: BAGL
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.00   11.00   12.00   12.12   12.60   15.00
------------------------------------------------------------
### (etc for remaining lakes)
```

Writing Simple Functions

- One of the features of R is that is it programmable. You can create simple functions or entire custom packages
- Writing a simple function to calculate Standard Error of the Mean

```
SE <- function(x) sqrt(var(x)/length(x))
SE(cond)
4.611665
round(SE(cond), 1)
4.6
```

- Writing a more complex function to estimate 95% confidence interval

```
ci95 <- function(x)   {
t.value <- qt(0.975, length(x)-1)
standard.error <- SE(x)
ci <- t.value * standard.error
cat("95 CI = ", round(mean(x) -ci, 1), "to ", round(mean(x) +ci, 1), "\n")  }

ci95(cond)
95 CI =  109 to  127.1
```

Exercise #2 - Modifying Simple Functions

- Modify the `SE` function to calculate the standard error of the mean for total phosphorus (tp)
  (Answer = 3.2 μg-P/L)

  Discussion: The IWS data are uncensored, so there are negative values in the total phosphorus results (`min` = -9.5 μg/L). How does this influence variance-based statistics like the standard error?

- Modify the `SE` function to select lakes with total phosphorus $\geq$3.2 μg/L (IWS detection limit)
  Answer = 3.5 μg/L

- *Optional:* Modify `ci95` to calculate the 95% confidence interval for total phosphorus in lakes with conductivities $\geq$100 μS
  Answer: 95 CI = 37.3 to 63.8

Saving Output to a Data Matrix

- We saw that the `tapply` command can be used to repeat ⇒simple functions

```
> round(tapply(cond, site, mean),1)
  ARM  BAGL  BAGU   BAK   BEA  BEAR   BIG   BUG   CAI   CAM   CAV   CAY   CED
 59.3  12.1  12.9  33.6 106.5   5.8  89.2 139.9  59.4 261.3  31.2  19.0  55.0
```

(etc.)

- We can create a temporary object that contains the results:

```
sitemean <- round(tapply(cond, site, mean),1)
```

- Here is how to create additional summary statistics, then "stack" the results into a new data matrix using `cbind`:

```
sitemed  <- round(tapply(cond, site, median),1)
sitemin  <- round(tapply(cond, site, min),1)
sitemax  <- round(tapply(cond, site, max),1)
siteN    <- tapply(cond, site, length)

summary.stats <- cbind(sitemin, sitemean, sitemed, sitemax, siteN)
```

Saving Output to a Data Matrix, continued

- This is a more advanced example, following the approach on page 24
- The objects are stacked into `summary.stats` using `cbind`, then made into a `data.frame` that includes site names (`unique(site)`)
- The `names` function is used to add descriptive names to the final data matrix, which is saved using the `write.table` function

```
lakes <- read.table("lakes.csv", T, sep=",")
attach(lakes)
sitemean <- round(tapply(cond, site, mean),1)
sitemed  <- round(tapply(cond, site, median),1)
sitemin  <- round(tapply(cond, site, min),1)
sitemax  <- round(tapply(cond, site, max),1)
siteN    <- tapply(cond, site, length)
siteID <- unique(site)

summary.stats <- cbind(sitemin, sitemean, sitemed, sitemax, siteN)

cond.stats <- data.frame(siteID, summary.stats)
names(cond.stats) <- c("Site", "Min", "Mean", "Median", "Max", "Count")

write.table(cond.stats, "conductivity.csv",
      quote=F, row.names=F, col.names=T, sep=",")
```

Writing/Running Source Files

- Most R users do *not* type long sequences of commands in the terminal window.
- Instead, we create ASCII text files containing the R commands

  By convention, R source files use the extension .R or .r, but Windows can make it hard to save files with this extension

  Fortunately, R also reads source files with .txt extensions
- To create a simple source file, open Wordpad or any ASCII editor, type the R syntax below, then save the file as simplelake.txt in your minicourse folder

```
lakes <- read.table("lakes.csv", T, sep=",")
attach(lakes)
summary(lakes)
```

Writing/Running Source Files, continued

- Switch to the R terminal window and type

  ```
  source("simplelake.txt")
  ```

  ⇒Make sure you changed the working directory to your minicourse folder and saved the source file to that folder

- If you did everything correctly, you should see nothing but the wedge-shaped start of line symbol (>) because R doesn't automatically print all output

- Re-open `simplelake.txt`, edit the last line to include a `print` statement, save file, then source the file

  ```
  lakes <- read.table("lakes.csv", T, sep=",")
  attach(lakes)
  print(summary(lakes))
  ```

- You should get a statistical summary for `lakes.csv`

Questions/One-on-One Instruction

- This portion of the minicourse is designed for personalized instruction, so can proceed at your own pace
- Novice R-users should review the examples and try to generate simple descriptive statistics for a different water quality variable (e.g., temperature)
- Intermediate and advanced R-users can set up source files and work on their own data sets
- Each section of this minicourse includes an edited source file that can be used to duplicate the more complex examples. The files are posted on the IWS web site (www.wwu.edu/iws)

R Source Files

Parts 1 – 4:	ecology1.r – ecology4.r
White River case study	whiteriver.r
PCA clustering case study	PCAclustering.r

Advanced Topics

R and Dates

- Year, month, and day can be converted with `ISOdate` or `chron` (discussed in Part 2)
- Strings of the form "2012/05/12" can be converted with `strptime`
- If your dates are factors, convert them first with `as.character` or create them with `stringsAsFactors=FALSE`
- Resulting objects can be used to plot on the horizontal axis

```
 lake.dates <- ISOdate(lakes$year, lakes$month, lakes$day)
 lake.dates[1]
[1] "2009-08-31 12:00:00 GMT"
 lake.dates[1] + 1
[1] "2009-08-31 12:00:01 GMT"
 lake.dates[1] + 24*60*60
[1] "2009-09-01 12:00:00 GMT"
 strptime("2012/5/12", format="%Y/%m/%d")
[1] "2012-05-12 PDT"
 strptime("12/5/12", format="%y/%m/%d")
[1] "2012-05-12 PDT"
 strptime("2012/5/12 13:30:02" , format="%Y/%m/%d %H:%M:%S")
[1] "2012-05-12 13:30:02 PDT"
 datestrings <- c("2012/5/1", "2015/2/22", "2011/12/25")
 strptime(datestrings, format="%Y/%m/%d")
[1] "2012-05-01 PDT" "2015-02-22 PST" "2011-12-25 PST"
```

Supplemental References

- Crawley, Michael J. 2013. The R Book. John Wiley & Sons. ISBN 978-0-470-97392-9
- Dalgaard, Peter. 2008. Introductory Statistics with R, 2nd Edition. Springer. ISBN 978-0-387-79053-4
- Lander, Jared P. 2014. R for Everyone, Advanced Analytics and Graphics. Addison Wesley Data & Analytics Series, ISBN 978-0-321-88803-7
- Teetor, Paul. 2011. The R Cookbook. O'Reilly Publishers. ISBN 978-0-596-880915-7