Note: R code is in courier font

Open a word file and paste your answers to the following questions

In R, ALWAYS

1. use the text editor in R to run / edit code; this can be saved as a .r text document
2. cut and paste your code, output , and figures into a the problem set word file to upload to the dropbox

Use the up arrow key to bring back previous commands; you can edit them – try it!

1. **Find & download data IF YOU ARE INEXPERIENCED WITH R PLEASE FOLLOW THESE DIRECTIONS EXACTLY**

# Our data files are located on D2L

# save worms.csv onto your desktop; THIS MAY NOT BE EASY IN WINDOWS

# if you mistakenly open it in excel, you absolutely must save it exactly as worms.csv, with the “comma-delimited” formatting and it needs to be found on your desktop.

# if the file name gets changed, you can change the filename in the code below so that it matches.

1. **Open an example of Crawley’s data: worms.csv  
   (this is probably the hardest thing to do in R)**

# to open data, first change the working directory by using the toolbar:

# on a mac: click on *misc/change working directory*

# on a pc: click on *file/change dir*

# then select the folder where you put the data – the desktop; it might be tricky to   
# find the desktop in windows; it’s usually under users/yourname/desktop

# In R, use the following to open worms.csv (if it works, R won’t complain or praise you – it will look like nothing happened)

worm.data = read.csv("worms.csv", header=T, row.names=1)

#!! note that if you type in word like this (“worms.csv”), it will fail because ms-word likes “smart quotes”, but R uses only standard quotes -- " " – see the difference?

# translation of the R code into regular English:

# “read the comma-delimited text in the file, “worms.csv” into a dataframe called worm.data,   
it is True that the first row contains a header stating the names of the variables,   
for the names of the rows, use the first column”

# now print the data to the screen:

worm.data

# print the first few lines of the dataframe; useful if the data set is large

head(worm.data)

# Attach the names of the variables to the actual vectors (lists) of numbers, and then print the names of variables:

attach(worm.data)

names(worm.data)

# here is some more info on file opening: have a look, but don’t use the code right now:

# read.table works on tab-delimted text files

# read.csv works on comma-delimited csv files – these will be used a lot

# header = T means that the column names will be read as column names

# row.names = 1 means that the words in column 1 will be used as row   
# names. This option is often omitted.

# you can open comma-delimited text files (.csv) with:

# read.csv("filename.csv", header = T)

2.1 paste the worms data frame here ;

# it won’t look very good, but this is your answer

Area Slope Vegetation Soil.pH Damp Worm.density

Ashurst 2.1 0 Arable 4.8 FALSE 4

Cheapside 2.2 8 Scrub 4.7 TRUE 4

Church.Field 3.5 3 Grassland 4.2 FALSE 3

Farm.Wood 0.8 10 Scrub 5.1 TRUE 3

Garden.Wood 2.9 10 Scrub 5.2 FALSE 8

Gravel.Pit 2.9 1 Grassland 3.5 FALSE 1

Gunness.Thicket 3.8 0 Scrub 4.2 FALSE 6

Nashs.Field 3.6 11 Grassland 4.1 FALSE 4

North.Gravel 3.3 1 Grassland 4.1 FALSE 1

Nursery.Field 2.8 3 Grassland 4.3 FALSE 2

Oak.Mead 3.1 2 Grassland 3.9 FALSE 2

Observatory.Ridge 1.8 6 Grassland 3.8 FALSE 0

Pond.Field 4.1 0 Meadow 5.0 TRUE 6

Pound.Hill 4.4 2 Arable 4.5 FALSE 5

Rookery.Slope 1.5 4 Grassland 5.0 TRUE 7

Rush.Meadow 2.4 5 Meadow 4.9 TRUE 5

Silwood.Bottom 5.1 2 Arable 5.2 FALSE 7

South.Gravel 3.7 2 Grassland 4.0 FALSE 2

The.Orchard 1.9 0 Orchard 5.7 FALSE 9

Water.Meadow 3.9 0 Meadow 4.9 TRUE 8

1. **get the moments of Worm.density etc.**

# to make a histogram, use:

hist(Worm.density)

# 3.1 paste the histogram below (it will open itself in another window; click on it, then paste in into word; if it’s too large, double click on it in word and change it’s size)

does this look like a normal distribution? Why / Why not?



The data is right-skewed

# 3.2 use the following functions to get some measures of central tendency and spread:

mean(Worm.density)

median() # yes, you have to put Worm.density into the function

range()

var()

sd()

> mean(Worm.density)

[1] 4.35

> median(Worm.density)

[1] 4

> range(Worm.density)

[1] 0 9

> var(Worm.density)

[1] 6.871053

> sd(Worm.density)

[1] 2.621269

# paste the output here:

# 3.3 Use the following code to create a skew function and then run it on worm.density, run it all together as a “chunk”

############## start chunk ##########

skew = function(x) {

m3 = sum((x-mean(x))^3)/length(x)

s3=sqrt(var(x))^3

m3/s3 }

skew(Worm.density)

############## end chunk ##########

# what is the skewness? Is the median = to the mean

# how is the skewness value interpreted?

The skewness is [1] 0.1195527 which is close to zero which means the data is approximately symmetric

# 3.4 Do the same for kurtosis.

############## start chunk ##########

kurtosis<-function(x){

m4<-sum((x-mean(x))^4)/length(x)

s4<-var(x)^2

m4/s4-3}

kurtosis(Worm.density)

############## end chunk ##########

# what is the kurtosis? How peaky the data is

# how is the kurtosis value interpreted?

[1] -1.258412 which means it is flat. Higher number = more peaky

# 3.5 Make a “qq plot” to visually assess whether worm density is strongly skewed or kurtotic; the data points are expected to follow along a straight line (the line is the expected values from a normal distribution, the dots are the observed data points)  
# a short optional youtube video: <https://www.youtube.com/watch?v=Erze9pNIX8A>

# next week we’ll try some formal tests of non-normality

qqnorm(Worm.density)

qqline(Worm.density)

# part 4 has been removed

**5. Use tapply to get means etc.**

# 5.1 the function tapply allows you to calculate a summary statistic across more than one group, for example the mean Worm.density among Vegetation types:

?tapply

# the help window shows:

*# tapply(X, Index, FUN = Null)*

# X is the variable of interest (Worm.density) #note the capitalization!

# Index is the grouping variable (Vegetation)

# FUN is the function. Functions include mean (mean), median (median), variance (var),

# standard deviation (sd), and length (length), among others.

#So now calculate the mean worm density among vegetation types, be careful here – R is case-sensitive so make sure the capital letters are:

tapply(Worm.density, Vegetation, mean)

#paste the answer here:

Arable Grassland Meadow Orchard Scrub

5.333333 2.444444 6.333333 9.000000 5.250000

# 5.2 you can store the answers into a variable, noting that you have put the right code inside the parentheses:

mean.worm.density = tapply(Worm.density, Vegetation, mean)

# do this to get the mean, median, variance, standard deviation, and length of worm.density across Vegetation types (i.e., change mean.worm.density to median.worm.density etc.), copy and paste the previous code and edit it:

median.worm.density = tapply(Worm.density, Vegetation, median)

var.worm.density … now it’s up to you to copy, paste, and edit the code!

sd.worm.density …

length.worm.density …

# organize these into a data frame and print it to the screen (run all of this part at once by highlighting all of it). copy/paste the output below as the answer to part 4.2.

summary.worm.density = data.frame(mean.worm.density, median.worm.density, var.worm.density, sd.worm.density, length.worm.density)

summary.worm.density

mean.worm.density median.worm.density var.worm.density

Arable 5.333333 5 2.333333

Grassland 2.444444 2 4.277778

Meadow 6.333333 6 2.333333

Orchard 9.000000 9 NA

Scrub 5.250000 5 4.916667

sd.worm.density length.worm.density

Arable 1.527525 3

Grassland 2.068279 9

Meadow 1.527525 3

Orchard NA 1

Scrub 2.217356 4

# Now have a closer look, and answer these questions:  
# 5.3 Are the means and medians always similar? no

# 5.4 If not, for which vegetation type(s) do the means and medians differ by more than a small amount?

# 5.5 For each, what are some reasons why this might occur?

# 5.6 There are some NA values – why would R not be able to calculate some things?

There is no variance within the data

1. **Output the new data.frame to a comma-delimited text file and open it in excel.**

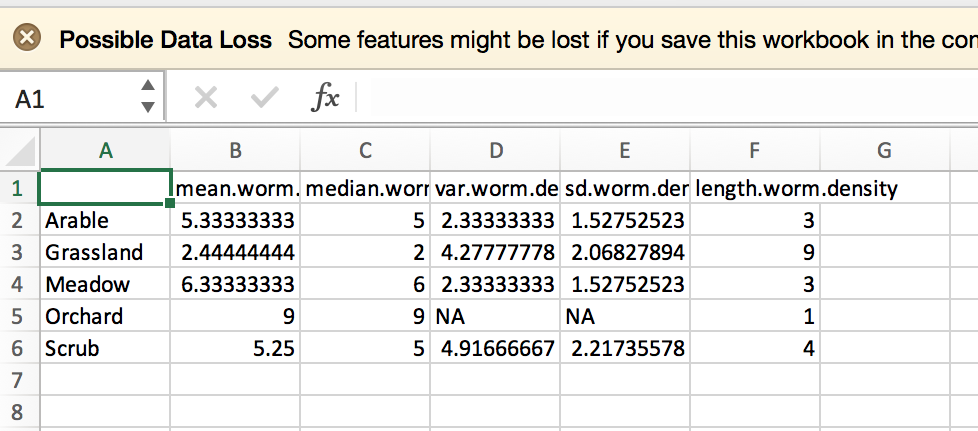
write.csv(summary.worm.density, file="summary of worm density.csv", row.names=T, quote=F)

# this will create a .csv file in your working directory that is comma delimited

# Translation: create a comma-delimited file named “summary of worm density.csv” and put the dataframe called summary.worm.density into it, include row names and column names, don’t put quotation marks around the entries.

# open excel and import the file.

# Copy and paste an image from excel here:



1. **Make some basic figures 2 – boxplot**

# Make a boxplot of worm.density: the format of the function looks like this:

# boxplot(y ~ x)

# translation “make a boxplot of y as a function of x”

# y is the variable of interest (Worm.density), aka the dependent variable

# x is the grouping variable (Vegetation), aka the independent variable

# 7.1 write the code for making the boxplot, run it, and paste the box plot here (if you can’t see all the words, change the shape of the quartz window).



# 7.2 Make some observations – what does the boxplot tell you about the data? Reconsider your answers to Question 4. Means are all over the place, shows quartiles and outliers.

# Here’s some help on interpreting the boxplot:  
http://flowingdata.com/2008/02/15/how-to-read-and-use-a-box-and-whisker-plot/

# **optional** 7.3 Here’s some code to make a minimalist boxplot in the style of Edward Tufte (copy and the paste the whole thing – it uses made -up data):

############## start chunk ##########

tufte.boxplot <- function(x, g) {

k <- nlevels(g)

crit.val <- tapply(x, g, median)

plot(1:k, crit.val, ylim=c(min(x)\*1.1, max(x)\*1.1), pch=19,

xlab=deparse(substitute(g)), ylab="worm density", axes=F)

#axis(side=1, labels=levels(Vegetation))

for (i in 1:k) {

tmp <- boxplot.stats(x[as.numeric(g)==i])

segments(i, tmp$stats[1], i, tmp$stats[2])

segments(i, tmp$stats[4], i, tmp$stats[5])

points(rep(i, length(tmp$out)), tmp$out, cex=.8)

}

}

dd <- data.frame(x=rnorm(50\*100), g=gl(50,100))

with(dd, tufte.boxplot(x, g))

########################## end

# this chunk makes the worm boxplot Tufte-style

with(worm.data, tufte.boxplot(Worm.density, Vegetation))

axis(side=1, at= c(1:length(levels(Vegetation))), labels=levels(Vegetation))

axis(side= 2)

############## end chunk ##########

1. **Optional: do some simple calculations in R**; when finished, copy and paste all your work here.

# 8.1 calculate the cube root of 2007, type the following in R and press <enter>  
2007^(1/3)

#8.2 get the absolute value of 5.7 minus 6.8 divided by .58; put the numbers into the following equation:  
abs(A-B)/C

#8.3 create a list of integers from 1 to 12 and call it “a”:

# note that “c” is a **function** ; it takes a group of numbers and concatenates it into a **vector (http://www.r-tutor.com/r-introduction/vector)**  
a = c(1:12)  
a #(this will print a –or whatever- to the screen, so you can paste it into word; do this each time)

#8.4 create a sequence of numbers from 10 to -1, stepping down by one each step. Note that seq is a function. Functions are like enzymes: they usually take one or more **arguments** (which are like the substrates for an enzyme) and then you put the output of the function into a variable, here it’s called “b” (b is the product of the enzyme)  
b = seq(10,-1, -1)  
b

#8.5 take the natural log (ln) of a: (note that this done to the entire vector called “a”)  
ln.a = log(a)  
ln.a

#8.6 take the log base 10 of a (the builders of R are nerds; they default the log function to base e because that’s how mathematicians think and because log base 10 is arbitrarily based the number of fingers most people have):  
log.a = log(a, 10)  
log.a

#8.7 inverse ln should return the list again:  
a.1 = exp(ln.a)  
a.1

#8.8 Similarly for log base 10:  
a.2 = 10^(log.a)  
a.2

#8.9 let’s check for mistakes:  
errors.a = abs(a - a.1) + abs(a - a.2)  
sum(errors.a)

#8.10 do you think there are errors? explain.

#8.11 create a “data frame” to hold all these and then print them out:  
all.a = data.frame(a, a.1, a.2, ln.a, log.a, b)  
all.a