Advanced Ecology 470 R tutorials; UVic Jan 2015

Created by JPWR, 28-Jan-2015

Data manipulation + statistics - working with data frames

1. Import a dataframe and explore its structure

We're going to work with some wolf and moose population data collected from Isle Royale (Michigan) (http://www.isleroyalewolf.org/data/data/home.html).

EITHER load your data using the full file path name

```
isle_dat<-read.csv(file="/Users/james/Desktop/LearnR/IsleRoyale_Data.csv")
```

OR set your working directory (best practice) and call the file directly. With this approach, any files you save will go straight to the 'learnR' folder.

```
setwd("/Users/james/Desktop/LearnR/")
isle_dat<-read.csv("IsleRoyale_Data.csv")
```

Now isle_dat is loaded into your workspace and we can begin to explore the dataset, make plots, run analyses etc. Let's have a look...

```
names(isle_dat)  ## what are the names of the variables in the data frame?
head(isle_dat) ## what are the first 6 rows of the data frame?
tail(isle_dat)  ## what are the bottom 6 rows of the data frame?
dim(isle_dat)  ## what are the dimensions of the data frame?
str(isle_dat)  ## what's the structure of the data frame?
```

Data frames are used to store data tables. Simply, they are a combination of rows and columns that you might view in an excel spreadsheet. We can select different combinations of rows and columns from a data frame using square brackets.

```
    isle_dat[1,1]   ### print the first row and first column value
    isle_dat[1:10, 1]  ### print the first 10 rows of the first column
```

So we call rows then columns. This is standard mathematic notation, so we would use the same description to work with a matrix object in R.

We can also call whole columns by name using the $ symbol.

```
isle_dat$year
isle_dat$kill_rate
```

2. Basic statistics in R

R has many basic functions that you can call on. Let's try a few..

```
mean(isle_dat$wolf_pop_size)    ### Calculate the mean of wolf population size
sd(isle_dat$wolf_pop_size)      ### Calculate the standard deviation of wolf population size
var(isle_dat$wolf_pop_size) ### Calculate the variance of wolf population size
max(isle_dat$wolf_pop_size)  ### Calculate the maximum value of wolf population size
summary(isle_dat$wolf_pop_size)   ### All the summary statistics at once
```

What if we only want to analyse a subset of the data frame?

```
subset(isle_dat, year==1964)   ### look at all values for year = 1964
```

We use two equal symbols to make a logical statement, and R selects any row in which "year" is equal to 1964.

Let's try year = 1964.

```
subset(isle_dat, year=1964)   ### Error - the full dataset is returned.
```

In R, a single = sign is equivalent to '<-'. This tells R to assign a name to a value, a function, or an object. A double == sign is equivalent to a logical statement - 'Is X the same as Y, TRUE or FALSE?'.

We can also build up multiple logical statements.

```
subset(isle_dat, year > 1964 & year < 1974)  ### Return years 1965 - 1973
```

3. Dealing with NAs

Dataframes often have NA values - these are usually variables that were not recorded over all of the sampling period. R stores these as 'Inf', "NaN", or "NA". You should always check your dataframe for NA values.

```
var(isle_dat$kill_rate)   ### Variance of kill rate is NA.
is.na(isle_dat$kill_rate)   ### Earliest years have no kill rate data.
```

Here we remove the NA values to calculate variance. Note that the NA values remain in the dataframe, but the variance function omits them.

```
var(isle_dat$kill_rate, na.rm=TRUE)  ## Calculate variance without NAs using na.rm = TRUE
var(na.omit(isle_dat$kill_rate))   ## Calculate variance without NAs using na.omit
```

You might want to remove all the NA values from your data - though be careful, because this will throw out useful data points as well. Here, ! is equivalent to "without", and is.na() identifies the NA values with TRUE or FALSE

```
isle_dat_clean<-isle_dat[!is.na(isle_dat$kill_rate),]  ### create new object without NAs.
dim(isle_dat)
dim(isle_dat_clean)  # dropped 12 rows
```

4. Storing new dataframes

We can break up dataframes and save a subset for a separate analysis, or run an analysis and store the results. Remember to name your objects appropriately (a short-hand description of the object is usually best).

```
isle_dat_70s<-subset(isle_dat, year >= 1970 & year < 1980)
```

There are several useful functions that aggregate statistics across variables or logical statements. To get the mean of every column in a dataframe we can use 'apply'.

```
apply(isle_dat, 2, mean)   ## print mean of every column (specified by the 2, rows are specified by a 1)
mean_pop_stats<-apply(isle_dat, 2, mean)
```

5. Exporting results

We used read.csv to bring our data into R. To get data out, we just use write.csv (remember - no spaces in your filenames!)

```
write.csv(mean_pop_stats, file="mean_popvars_isledat.csv")
```

If you used setwd() at the start of your script, a new csv file should have appeared in your working directory. If you didn't use setwd(), the csv file has probably disappeared into your computer hard drive. Use getwd() to check where it ended up...

```
getwd()   ## csv file wasn't created where I thought it would be. Where is it?
```

R can export many different types of file. Check http://www.statmethods.net/input/exportingdata.html for some examples.

6. Linear regression in R

How do we run a linear regression in R?. We need to adopt formula notation to use the command lm().

```
mod1<-lm(wolf_pop_size ~ kill_rate, isle_dat)    # fit a linear model and save to workspace as mod1
summary(mod1)           # model output
coef(mod1)              # model coefficient estimates
```

Remember to check the assumptions of a linear model. You can use normality tests, build residual plots, and extract fitted model values in R. For more information on linear models, look here:

- http://www.r-tutor.com/elementary-statistics/simple-linear-regression
- http://blog.yhathq.com/posts/r-lm-summary.html