THE UNIVERSITY OF NEW SOUTH WALES

Short Intermediate Statistics Course

Stats Central

School of Mathematics and Statistics

Eco-Stats Research Group

School of Biological, Earth and Environmental Sciences

13-17 February 2017

Contents

References and Readings

Session 1 – Experimental Design

Keough, M. J., & Quinn, G. P. (2002). *Experimental Design and Data Analysis for Biologists, seventh edition*. Cambridge University Press.

Underwood, A. J., & Chapman, M. G. (2013). *Design and analysis in benthic surveys in environmental sampling. Methods for the Study of Marine Benthos, Fourth Edition*. Wiley Online Library

Session 2 – Intro Stats Revision

Dalgaard, P. (2008). *Introductory Statistics with R, second edition*. Springer, New York.

Moore, D. S., McCabe, G. P. & Craig, B. A. (2012). *Introduction to the Practice of Statistics, seventh edition*. WH Freeman.

Warton, D. I. & Hui, F. K. C. (2011). The arcsine is asinine: the analysis of proportions in ecology. *Ecology*, **92**, 3–10.

Session 3-5 – Linear Regression and (Weirder) Linear models

Faraway, J. J. (2005a). *Linear models with R*. CRC Press.

Quinn, G. G. P. & Keough, M. J. (2002). *Experimental design and data analysis for biologists*. Cambridge University Press.

Weisberg, S. (2013). *Applied Linear Regression*. John Wiley & Sons.

Session 6 – Model selection

Burnham, K. P., & Anderson, D. R. (2002). *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*. Springer-Verlag New York.

Hastie, T., Tibshirani, R. & Friedman, J. (2009). *The elements of statistical learning, second edition*. Springer-Verlag New York.

Shao, J. (1993). *Linear model selection by cross-validation*. *Journal of the American Statistical Association*, **88**, 486–494.

Shao, J. (1997). *An asymptotic theory for linear model selection*. *Statistica Sinica*, **7**, 221–262.

Session 7 – Generalised linear models

Dunn, P. & Smyth, G. (1996). Randomized quantile residuals. *Journal of Computational and Graphical Statistics*, **5**, 236–244.

Faraway, J. J. (2005b). *Extending the linear model with R: generalized linear, mixed effects and nonparametric regression models*. CRC press.

Warton, D. .I. (2005). Many zeros does not mean zero inflation: comparing the goodness-of-fit of parametric models to multivariate abundance data *Envirometrics*, **2005**, 275–289.

Warton, D. I. & Hui, F. K. C. (2011). The arcsine is asinine: the analysis of proportions in ecology. *Ecology*, **92**, 3–10.

Welsh, A. H., Cunningham, R. B., Donnelly, C. & Lindenmayer, D. B. (1996). Modelling the abundance of rare species: statistical models for counts with extra zeros. *Ecological Modelling*, **88**, 297–308.

Wright, D. B. & London, K. (2009). *Modern regression techniques using R: a practical guide*. Sage.

Session 8-9 – Mixed effects models

Bates, D. (2013). Linear mixed model implementation in lme4. *University of Wisconsin, http://lme4.r-forge.r-project.org/lMMwR/lrgprt.pdf*.

Bolker, B. M., Brooks, M. E., Clark, C. J., Geange, S. W., Poulsen, J. R., Stevens, M. H. H. & White, J. S. (2009). Generalized linear mixed models: a practical guide for ecology and evolution. *Trends in Ecology & Evolution*, **2009**, 127–135.

Zuur, A. F. (2009). *Mixed effects models and extensions in ecology with R*. Springer.

Datasets

Moles, A. T., Warton, D. I., Warman, L., Swenson, N. G., Laffan, S. W., Zanne, A. E., Pitman, A., Hemmings, F. A. & Leishman, M. R. (2009). Global patterns in plant height. *Journal of Ecology*, **97**, 923–932.

Poore, A. G., Hill, N. A. & Sotka, E. E. (2008). Phylogenetic and geographic variation in host breadth and composition by herbivorous amphipods in the family Ampithoidae. *Evolution*, **62**, 21–38.

Roberts, D. A. & Poore, A. G. (2006). Habitat configuration affects colonisation of epifauna in a marine algal bed. *Biological Conservation*, **127**, 18–26.

Wright, I. J., Reich, P. B., Westoby, M., Ackerly, D. D., Baruch, Z., Bongers, F., Cavender-Bares, J., Chapin, T., Cornelissen, J. H., Diemer, M. *et al.* (2004). The worldwide leaf economics spectrum. *Nature*, **428**, 821–827.

Exercises

Session 1 – Experimental Design

Key messages:

- Experimental design starts with asking good questions, the more specific your question, the better your planning can be to answer it.
- Experiments are designed to avoid confounding. To do this we need controls, replication and randomisation.
- Stratification and blocking can reduce unexplained variation and increase the power of your experiment.
- Pilot studies can save time and money by helping you better plan your main study.

1. Consider the following example.

   > *Tanya is interested in how changing water tables from mining will affect the biomass of plants in affected swamps. She has conducted a pilot study, where she extracted 10 soil sods, and exposed them to two water table levels in a greenhouse, one (the control) simulating current conditions, and the other (treatment) with a reduced water level. She measured the biomass of each sod after 6 weeks. In her pilot study, mean biomass in the control group was 92 gm while in the treatment group the mean was 85 gm. The standard deviation within treatments is 45. Tanya is interested in detecting a 15 gm difference between treatments. Tanya wants to know an appropriate sample size for her main study.*

   (a) What is the research question? Descriptive statistics or inference? Do we have a specific hypothesis to test or are we just estimating some parameter?

   (b) What are the main properties of the data? One variable or two? Categorical or quantitative?

   (c) What is the standard deviation within treatments?

   (d) What is the relevant effect size?

(e) Use the `pwr.t.test` function in the `pwr` package to find the sample size required to have 80% chance of detecting the desired effect size.

(f) Use the `sample` function in R to allocate the N sods randomly to treatments.

Session 2 – Introductory Stats Revision

Key messages:

- When thinking about how to analyse data, think: (1) what is the research question? (2) what are the main properties of the data?
- When doing inferential statistics e.g., hypothesis tests, confidence intervals, there are assumptions: what are they? Are they reasonable for my study?
- Be careful using and interpreting hypothesis tests and P-values
- Do you know how to do a two-sample t-test on `R`, and interpret the output?

Feel free to refer to the How-To Manual any time you want help on `R`. (Or Google the problem!)

1. Consider the guinea pigs (smoking) example:

   Is there an association between # of errors and treatment group (Control and Nicotine)?

   (a) Load the `smokePregnant` data into `R`.

   (b) What is the research question? Descriptive statistics or inference? Do we have a specific hypothesis to test or are we just estimating some parameter?

   (c) What are the main properties of the data? One variable or two? Categorical or quantitative?

   (d) Produce a graph relevant to the original research question..

   (e) Left-skewed, right-skewed, symmetric?

   (f) Use a two-sample t-test to see if there is evidence the # of errors (transformed if required) is associated with treatment group.

   (g) What is a plausible range of values for the mean difference between treatments?

   (h) What did you assume in the above? Check any assumptions you can, and comment on anything that could have been done in the study design to ensure assumptions were reasonable.

   (i) I assume you have been keeping a record of all your `R` commands in a script file? Save this script file, it will come in handy later.
   (You should get in the habit of *always* doing this.)

2. Consider the bats example:

   Kerry goes bat counting. She finds 65 female bats and 44 male bats in a colony. She would like to know if there is evidence of gender bias.

(a) What is the research question? Descriptive statistics or inference? Do we have a specific hypothesis to test or are we just estimating some parameter?

(b) What are the main properties of the data? One variable or two? Categorical or quantitative?

(c) Produce a graph relevant to the original research question..

(d) Test if there is evidence of gender bias in the colony (consider `prop.test`)

(e) What is a plausible range of values for the sex ratio in this colony?

(f) What did you assume in the above? What could have been done to ensure assumptions were reasonable?

Session 3 – Linear Regression

Key messages:

- Simple linear regression is about looking for an association between a response (y-variable) and a predictor (x-variable).
- How do I test assumptions in linear regression?
- Two-sample t-test is just a special case of simple linear regression!

Feel free to refer to the How-To Manual any time you want help on `R`. (Or Google the problem!)

1. Consider the guinea pigs (smoking) example again:

   Is there an association between # of errors and treatment group (Control and Nicotine)?

   (a) Load the `smokePregnant` data into `R`.

   (b) The data could be organised differently.

   Instead of looking like this...

```
      Treatment Control
1            38      11
2            26      19
3            33      15
4            89      47
[etc]
```

   Can you can make it look like this?

```
   errors treatment
1      38  Nicotine
2      26  Nicotine
3      33  Nicotine
4      89  Nicotine
[etc]
11     11   Control
12     19   Control
13     15   Control
14     47   Control
[etc]
```

   (c) Left-skewed, right-skewed, symmetric? If you think `errors` needs transforming...

   (d) In the session 1 exercises, we used a two-sample t-test. Let's now use linear regression. To do this:

      i. Construct a new variable `numericTreatment`, which takes the value 1 for `Nicotine` and 0 for `Control`.

      ii. Fit a linear regression for `errors` (transformed if required) against `numericTreatment`.

      iii. Compare the output to that obtained previously for the two-sample t-test.

   (e) What did you assume in the linear regression model above? What could have been done to ensure assumptions were reasonable?

2. Consider the water quality example:

   How is water quality in creeks associated with/related to the size of the catchment area?

   (a) Load the `waterQual` data into `R`.

   (b) What is the research question? Descriptive statistics or inference? Do we have a specific hypothesis to test or are we just estimating some parameter?

   (c) What are the main properties of the data? One variable or two? Categorical or quantitative?

   (d) Produce a graph relevant to the original research question.

   (e) What model do we want to use here? What parameter in this model describes how water quality is associated with catchment area?

   (f) What is a plausible range of values for this key parameter? Hint: Use `confint`

   (g) What did you assume in the above? What could have been done to ensure assumptions were reasonable?

Session 4 – Linear models

Key messages:

- Multiple regression is an extension of simple linear regression, except: interpret effects conditionally, partial residual plots, can use `anova` to test multiple parameters, and consider multi-collinearity.
- ANOVA is just a special case of multiple linear regression (linear models)!

Feel free to refer to the How-To Manual any time you want help on `R`. (Or Google the problem!)

1. Consider the global plant height example:

   Can latitudinal variation in plant height be explained by rainfall?

   (a) Load the `plantHeightSingleSpp` data into `R`.

   (b) What is the research question? Descriptive statistics or inference? Do we have a specific hypothesis to test or are we just estimating some parameter?

   (c) What are the main properties of the data? One variable or two? Is the response variable categorical or quantitative?

   (d) Produce a graph relevant to the original research question.

   (e) What model do we want to use here?

   (f) What did you assume in the linear regression model above? What could have been done to ensure assumptions were reasonable? Transformations?

   (g) Check for multi-collinearity.

   (h) What parameter in your model is of particular interest?

   (i) What is a plausible range of values for this key parameter?

   (j) Now reanalyse data to answer the question:

   Can latitudinal variation in plant height be explained by climate (defined as precipitation and mean temperature)?

2. Consider Alistair's habitat configuration study:

   Does epifauna density change with distance of isolation?

   (a) Load the `HabitatConfig` data into `R`.

   (b) Sub-sample the data so that we are only considering small patches left for 10 weeks before measurement (Hint: `dat$Size=="SMALL"&dat$Time==10`).

   (c) What is the research question? Descriptive statistics or inference? Do we have a specific hypothesis to test or are we just estimating some parameter?

   (d) What are the main properties of the data? One variable or two? Is the response variable categorical or quantitative?

   (e) Produce a graph relevant to the original research question.

   (f) What model do we want to use here?

   (g) What did you assume in the linear regression model above? What could have been done to ensure assumptions were reasonable? Transformations?

   (h) After transformation (if required), test if there is evidence that density is related to distance of isolation.

   (i) Do multiple comparisons to find where the pairwise treatment differences are (if there are any).

Session 5 – Weirder linear models

Key messages:

- Paired and blocked designs can be used to control for known sources of variation that are not of primary interest.
- Analysis of covariance (ANCOVA) is a blocked design where the blocking factor (or "covariate") is quantitative
- Order is important: To test for an effect of B after controlling for A, formula must have the form `y~A+B`.
- Factorial designs allow testing of the additivity assumption ("interactions"). These interactions can answer cool questions but they can be complicated to interpret.
- Blocked designs, ANCOVA, factorial ANOVA... it's all just linear models!
- You can test for interactions in regression as well – but note that interactions between quantitative variables are quadratic terms.

Feel free to refer to the How-To Manual any time you want help on `R`. (Or Google the problem!)

1. Consider the habitat configuration example:

   Does invertebrate density change with distance of isolation? Does this relationship vary with sampling time or patch size?

   (a) Load the `HabitatConfig` data into `R`.

   (b) What is the research question? Descriptive statistics or inference? Do we have a specific hypothesis to test or are we just estimating some parameter?

   (c) What are the main properties of the data? One variable or two? Is the response variable categorical or quantitative?

   (d) Produce a graph relevant to the original research question.

   (e) What model do we want to use here?

   (f) What did you assume in the linear model above? What could have been done to ensure assumptions were reasonable? Transformations?

   (g) Well... does invertebrate density change with distance of isolation? Does this relationship vary with sampling time or patch size?

   (h) Where are the pairwise treatment differences? (Hints: make sure all relevant variables are factors, plotting results might help.)

   (i) Wet mass (`Wmass`) of seaweed patches is an important predictor of density – the more seaweed there is, the more invertebrates one might expect to find living on it. Adding this variable to the model might control for some of the unaccounted for variation between replicates.

i. Produce a useful graph for looking at whether `Wmass` is an important predictor of density.

ii. Does adding habitat wet mass to the model change the previous results in any way?

iii. Is the effect of habitat wet mass additive or does the effect change with distance of isolation?

Session 6 – Model selection

Key messages:

- It's hard. Work on shortlisting your x variables first to get better results – there is a limit to what the data can tell you, any help you can give it is much appreciated.
- Useful tools for model selection are cross-validation and information criteria
- If using cross-validation, repeat, to account for randomness in training and test splits
- Useful methods for finding variable subsets are stepwise regression, all-subsets, and penalised estimation (*e.g.* LASSO)
- Penalised estimation methods, like the LASSO, has good predictive properties while simplifying the problem of subset selection.

Feel free to refer to the How-To Manual any time you want help on `R`. (Or Google the problem!)

1. Consider the global plant height example:

   > *Angela collects data on how tall plants are at lots of different places around the globe. She also has data on 8 different precipitation variables. She is interested in how plant height relates to precipitation, and to which precipitation variables height relates to most closely.*

   (a) Load the `plantHeightSingleSpp` data into `R`.

   (b) What is the research question? Descriptive statistics or inference? Do we have a specific hypothesis to test or are we just estimating some parameter?

   (c) What are the main properties of the data? One variable or two? Categorical or quantitative?

   (d) What model do we want to use here?

   (e) Fit a model predicting height from a bunch of rainfall variables.

   (f) What did you assume in the model above? What could have been done to ensure assumptions were reasonable? Transformations?

   (g) Find a subset of precipitation variables that optimally predicts plant height. Try a couple of different methods.

   (h) Any issues with multi-collinearity amongst the precipitation variables? Try to address any multi-collinearity by culling one or two of the main culprits. Does this affect your previous model selection results?

Session 7 – Mixed effects models

Key messages:

- Use random effects in your model if you have a factor with a large number of levels, you only sampled some of these levels and did so at random, and you want to generalise across all levels.
- Do this using a linear mixed effects model, which can be fitted using the `lme4` package.
- Accurate inference is more difficult for mixed effects models. `summary` and `anova` are "quick-and-dirty" – can be very approximate, don't take too seriously.

Feel free to refer to the How-To Manual any time you want help on `R`. (Or Google the problem!)

1. Plant height data again:

   How does plant height change as latitude changes?

   This time we will use multiple data from each site – height data from up to 5 randomly chosen plant species.

   (a) Load the `plantHeight5Spp` data into `R`.

   (b) What is the research question? Descriptive statistics or inference? Do we have a specific hypothesis to test or are we just estimating some parameter?

   (c) What are the main properties of the data? One variable or two? Is the response variable categorical or quantitative? Fixed or random factors?

   (d) What type of model do we want to use here?

   (e) What did you assume in the model above? What could have been done to ensure assumptions were reasonable? Transformations?

   (f) Use `summary` to get a quick-and-dirty answer to Angela's key question.

   As always, make sure you save your analysis script. We will come back to this dataset later.

2. Consider Graeme's estuary data. For each Estuary, Graeme actually took 4-7 measurements at each of "Inner" and "Outer" zones. He wants to know:

   > Is there an effect of modification on species richness (`Richness`)? Is this effect different in different estuary zones?

   (a) Load the `estuaries` data into `R`.

   (b) What is the research question? Descriptive statistics or inference? Do we have a specific hypothesis to test or are we just estimating some parameter?

   (c) What are the main properties of the data? One variable or two? Is the response variable categorical or quantitative? Fixed or random factors?

   (d) What type of model do we want to use here?

   (e) What did you assume in the model above? What could have been done to ensure assumptions were reasonable? Transformations?

   (f) Use `anova` to get a quick-and-dirty test of Graeme's key questions.

   As always, make sure you save your analysis script. We will come back to this dataset later.

Session 8 – Mixed effects models (continued)

Key messages:

- Recall that accurate inference is more difficult for mixed effects models. `summary` and `anova` are "quick-and-dirty" – can be very approximate and shouldn't be taken too seriously.
- If you need accurate inference from mixed models about fixed effects you should consider using resampling.
- If you want to make inferences about a random effect in your model then you almost always consider resampling.

Feel free to refer to the How-To Manual any time you want help on `R`. (Or Google the problem!)

1. Plant height data again:

   How does plant height change as latitude changes?

   This time we will use multiple data from each site – height data from up to 5 randomly chosen plant species.

   (a) Load the `plantHeight5Spp` data into `R`, and your saved analysis script.

   (b) This would all have been a bit easier if there wasn't a random effect in the model... do you think we really need `Site` in there?

2. Consider Graeme's estuary data. For each Estuary, Graeme actually took 4-7 measurements at each of "Inner" and "Outer" zones. He wants to know:

   Is there an effect of modification on species richness (`Richness`)? Is this effect different in different estuary zones?

   (a) Load the `estuaries` data into `R`, and your saved analysis script.

   (b) Use the parametric bootstrap to get a formal test for a `zone:mod` interaction.

   (c) How do results compare to those from when you were using the `anova` function?

   (d) This would all have been so much easier if there wasn't a random effect in the model... do we need `Estuary` in there?

Session 9 – Wiggly models

Key messages:

- Spline smoothers or generalised additive models allow you to fit curves under a linear modelling framework
- Don't be fooled by the term "additive" – they can handle interactions too. But if you want to keep it simple you can manually add a quadratic interaction.
- Smoothers are useful additions to residual plots, to help you look for a mean trend.
- If you have "circular data" (time of day, aspect, ...) make sure you `cos` and `sin` your variable before using in modelling.
- You can fit more flexible periodic functions by adding `cos` and `sin` functions with half the period.

Feel free to refer to the How-To Manual any time you want help on `R`. (Or google the problem!)

1. Plant height data again:

   Angela is interested in how plant height relates to latitude, and whether the relationship is non-linear.

   (a) Load the `plantHeightSingleSpp` data into `R`.

   (b) What is the research question? Descriptive statistics or inference? Do we have a specific hypothesis to test or are we just estimating some parameter?

   (c) What are the main properties of the data? One variable or two? Categorical or quantitative?

   (d) What type of model do we want to use here?

   (e) What did you assume in the model above? What could have been done to ensure assumptions were reasonable? Transformations?

   (f) Plot the fit. Does it look non-linear?

   (g) Which model is better for predicting height, linear or non-linear? Answer this question using the model selection approach of your choice.

   (h) But wait, there's more:

   Angela is interested in how plant height relates to climate, and whether the relationship is non-linear.

   i. Any evidence of interaction?

   ii. Is a quadratic interaction term sufficient or does this need to be non-linear?

Session 10 – Design-based inference

Key messages:

- Design-based inference (as opposed to model-based inference) leverages from independent units from the study design to make valid inferences even when some aspects of the fitted model are wrong.
- Permutation tests and bootstrapping are key tools for design-based inference (especially hypothesis testing), and easy to use in the `mvabund` package.
- These methods are exact (or close to) when the null hypothesis is no effect, and can still be used in other settings by resampling residuals.
- Check assumptions! Resampling is not a get-out-of-jail card. If your model is very wrong your results can be very wrong even with design-based inference.

1. Poore et al (2008, *Evolution*) did a meta-analysis of host breadth across four genera of crustacean.

   Are some crustacean genera more specialist than others?

   For each of 104 species across the four genera, they recorded the number of different hosts it was found on. They also recorded the number of published studies for each species. This was considered important because a species that is better studied tends to found on more types of host plant. We will use the `HostSpecialisation` dataset.

   (a) What is the research question? Descriptive statistics or inference? Do we have a specific hypothesis to test or are we just estimating some parameter?

   (b) What are the main properties of the data? One variable or two? Is the response variable categorical or quantitative?

   (c) Produce a graph relevant to the original research question.

   (d) What model do we want to use here?

   (e) What did you assume in the linear regression model above? What could have been done to ensure assumptions were reasonable? Transformations?

   (f) Do some genera tend to be found on more hosts than others? Use design mased and model nased inference to answer this question. Do your answers differ?

2. Go back through your Session 2-4 analyses, pick a dataset, and a hypothesis test. Repeat the test(s) using resampling – did you get the same answer?

Session 11 – Generalised linear models

Key messages:

- If you have discrete data (with small counts or zeros) you cannot transform your way out of trouble – you need a GLM
- Key GLM decisions – what mean-variance relationship, what link function. (But just choosing the distribution usually sorts both out at once.)
- Default residual plots are not great, but re-fitting as a `manyglm` from the `mvabund` package solves the problem. Make sure there isn't a big fan-shape.
- Inference for GLMs is approximate but good in most instances. If you have small samples and small counts you can use the `mvabund` package for design-based inference (bootstrapping).

Feel free to refer to the How-To Manual any time you want help on `R`. (Or Google the problem!)

1. Consider Anthony's bush regeneration survey. He wants to know:

   Is abundance of Blattodea *different between revegetated and control sites?*

   (a) Load the `reveg` data into `R`.

   (b) What is the research question? Descriptive statistics or inference? Do we have a specific hypothesis to test or are we just estimating some parameter?

   (c) What are the main properties of the data? One variable or two? Is the response variable categorical or quantitative? (Discrete? Continuous?)

   (d) Produce a graph relevant to the original research question.

   (e) What model do we want to use here?

   (f) What did you assume in the model above? What could have been done to ensure assumptions were reasonable? Transformations?

   (g) Poisson or negative binomial? As well as doing residual plots, Compare `AIC` for the two models.

   (h) Does abundance of *Blattodea* differ between control and revegetated sites?

   (i) How much does it vary by? Construct a confidence interval for key parameter(s) of interest.

   (j) This is a pretty small dataset – might be safer to use design-based inference. Repeat your hypothesis test using a bootstrap test for exact inference about the effect of revegetation.

2. Consider Graeme's data – For each of seven Estuaries (some `Pristine`, some `Modified`), Graeme took 4-7 measurements at each of "Inner" and "Outer" zones. He wants to know:

   Is there an effect of modification on presence/absence of `Amphipod.tubes`?
   Is this effect different in different estuary zones?

   (a) Load the `estuaries` data into `R`.

   (b) What is the research question? Descriptive statistics or inference? Do we have a specific hypothesis to test or are we just estimating some parameter?

   (c) What are the main properties of the data? One variable or two? Categorical or quantitative?

   (d) What type of model do we want to use here?

   (e) What did you assume in the model above? What could have been done to ensure assumptions were reasonable? Transformations?

   (f) Use `anova` to get a quick-and-dirty test of Graeme's key questions.

   (g) Use the parametric bootstrap to get a formal test for a `mod` main effect.

   (h) This would all have been so much easier if there wasn't a random effect in the model... do we need `Estuary` in there?

Session 12 – Multivariate analysis

Key messages:

- Do you really need to go multivariate?
- If you do, there aren't many variables, and you can use linear models, the `lm` function still works – just make sure your response variable is stored as a matrix.
- Make sure you plot data. Plotting each variable separately is a good idea, scatterplot matrices or ordinations can help – but use ordinations cautiously and always ground-truth findings against plots of the raw data.

Feel free to refer to the How-To Manual any time you want help on `R`. (Or google the problem!)

1. Coral cover was measured along ten transects in the Tikus Islands (Indonesia) at different times – we will focus on 1981 and 1983 data, before and after an El Niño event, respectively. Warwick et al (1990) used this dataset and MDS ordinations to argue that stress increases dispersion in coral communities.

   (a) Load the data from the `mvabund` package using:

   ```
   > library(mvabund)
   > data(tikus)
   ```

   (b) The first 20 observations of `tikus$abund` contain the 1981 and 1983 observations – extract these 20 rows and store them as `dat`.

   (c) Construct a factor called `time` which takes the value 1981 for the first 10 observations, and 1983 for the second ten observations.

   (d) Do an MDS plot of the data, and colour-code symbols by year of sampling. Does this plot agree with Warwick et al's interpretation?

   (e) Use the `plot.mvabund` function to plot each coral response variable as a function of time. What is the main pattern that you see?

2. Consider Ian's data. We want to know:

   > Is there evidence that leaves vary (in their "economics" traits) across sites with different levels of rainfall and soil nutrients?

   (a) Load the `leaflife` dataset from the `smatr` package.

   (b) Consider the residual plots on the lecture slides – any problems? Any suggestions?

   (c) Reanalyse the data using any necessary changes.

   (d) In which variable(s) is there an effect? Fit univariate linear models to dig a little deeper.

Session 13 – Multivariate abundance data

Key messages:

- Do you really need to go multivariate?
- Treat "conventional" tools for multivariate analysis in ecology with some scepticism – especially distance-based approaches (PRIMER) and canonical correspondence analysis (CANOCO), which are also available on `vegan`.
- Make sure you plot data. But watch out for ordinations, they can do some weird things if used inappropriately, need to ground-truth with plots of raw data.
- For multivariate abundance data you need to account for the mean-variance relationship in anything you do – it can swamp everything else. Using GLMs via the `manyglm` package might do the trick.
- Post-hoc testing – first break things down by response variable. The `p.uni` argument on `mvabund` can help with this.

Feel free to refer to the How-To Manual any time you want help on `R`. (Or google the problem!)

1. Consider Alistair's habitat patches.

   Is there evidence of a difference in invertebrate communities settling on patches with different distance of isolation? Does this effect change over time?

   (a) Load the `habitatConfig` data into `R`.

   (b) Columns 6-21 are the response variables – extract these and save as a `mvabund` object.

   (c) Produce a graph relevant to the original research question.

   (d) For simplicity we will analyse **presence/absence data only**. Convert the response variable to presence/absence using:

   ```
   > pres=I(abund>0)
   ```

   (e) What is the research question? Descriptive statistics or inference? Do we have a specific hypothesis to test or are we just estimating some parameter?

   (f) What are the main properties of the data? One variable or two? Is the response variable categorical or quantitative? (Discrete? Continuous?)

   (g) What model do we want to use here?

   (h) What did you assume in the model above? What could have been done to ensure assumptions were reasonable? Transformations?

(i) So... is there a difference in invertebrate communities settling on patches with different distance of isolation? And does this effect change over time? (Use `show.time=T` to see how long it will take)

(j) Come up with a "top 3" taxa that most strongly express a `Dist:Time` interaction. How much of the overall change in deviance is explained by these three taxa?

2. Reanalyse the epifauna data from Alistair's habitat patches, this time analysing the count data directly (as in `HabitatCounts`). In particular, think about:

   (a) What distribution (`family`) do you think might work here?

   (b) Are there any offset terms that should be included in the model?

   (c) How do results compare to the presence/absence analysis?

The RStudio How-To Manual

This chapter introduces you to the statistics program RStudio, which is 'jazzed up' version of the basic but powerful R statistical programming language. In fact RStudio is the statistics package that was used to construct most of the graphs in the slides! RStudio is FREE to download and use on your own computer, and simple instructions to install it on your own computer are given at the start of this manual.

The below sets of instructions provide you with the skills you require in order to complete the exercises given previously.

General introduction to RStudio

How to install RStudio on your home computer

If you want to also install it on your own computer so that you can work at home, then it is easy to do: Either watch the short youtube video http://www.youtube.com/watch?v=9m3Q-gIJh4M for visual instructions, or follow the steps below

1. Go to CRAN http://cran.r-project.org/, the official R website.
2. In the section at the top of the page, titled "Download and install R", select the link corresponding to the type of computer you want to install R on.
3. Mac users: simply download the appropriate .pkg file and open it to start installing.

   Windows users: click on "base" and download the file with a name of the form R-...exe. This is the installation file, and opening it will start installing.

   Linux users: you will need to know what operating system you are running. Navigate to the directory describing your operating system, and download an appropriate R-base file. (or use your appropriate package manager commands to locate and install R).
4. Once you open the file you just need to follow the installation instructions.
5. Go to http://www.rstudio.com/products/rstudio/download/ to choose your version of RStudio appropriate for your computer, download and follow the installation instructions.

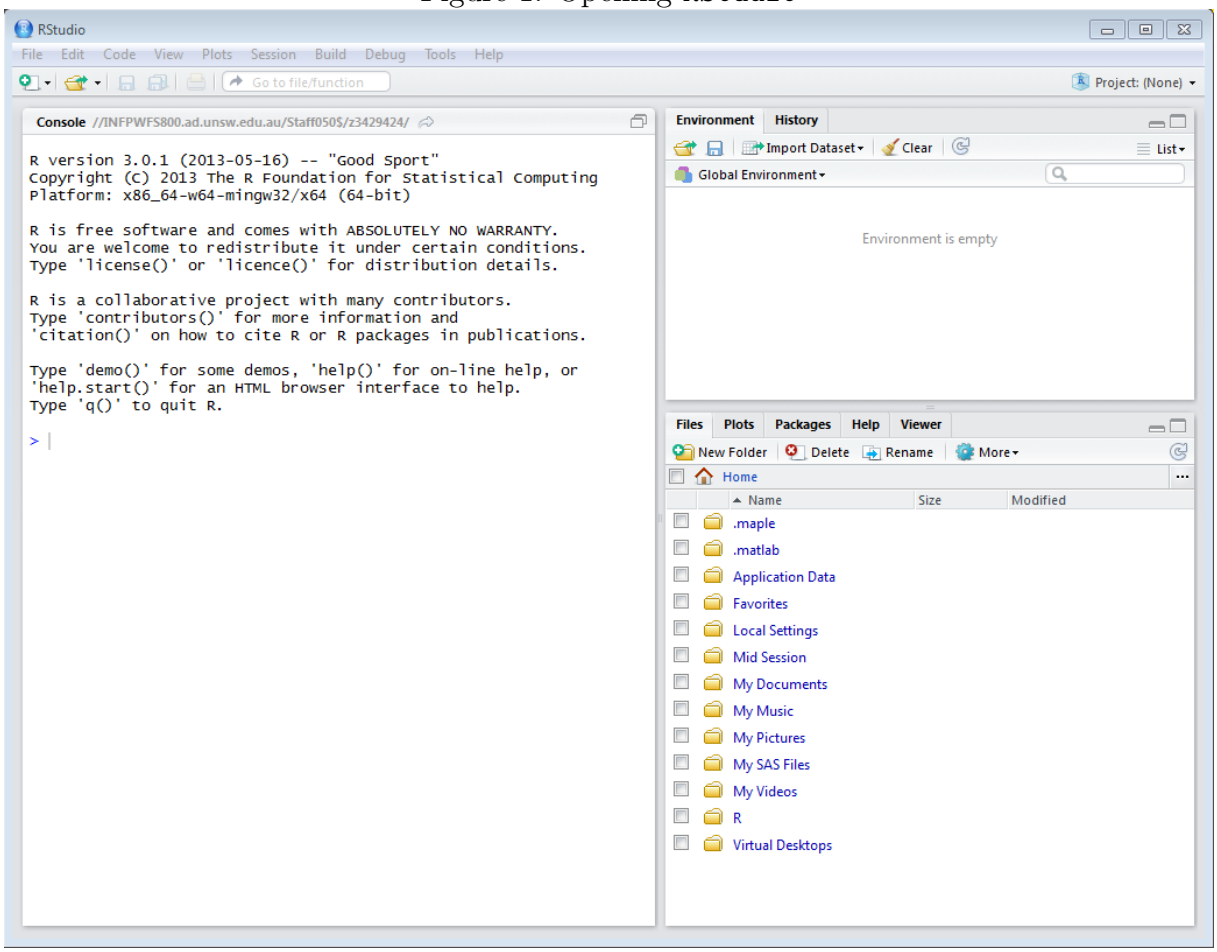
Handy tip: If you don't have your own computer and instead you often use one of several different computers, a useful trick is to install `R` and `RStudio` onto a USB stick (instead of installing it onto the computer's hard drive). A standard `R`plus `RStudio` installation takes up less than 100M of space, so it will fit easily onto most USB sticks. Then you would be able to use `RStudio` on pretty much any computer that reads your USB stick! For more information, check out https://support.rstudio.com/hc/en-us/articles/200534467-Creating-a-Portable-Version-of-RStudio-for-a-USB-Drive.

How to open `RStudio`

Navigate to the `RStudio` program in the start menu and launch the program. You can either look for the `RStudio` folder itself which contains the shortcut to the program, or you can type 'RStudio' in the run/search bar to look for it directly.

Once `RStudio` opens up (it make take a little while), you will end up up something like in Figure 1

Figure 1: Opening `RStudio`



Like most Windows programs, there are several commands at the top such as *File*, *Edit* and so on. Below this, you should see three panels in `RStudio`:

- The big left panel is called the *console*. Whenever `RStudio` is opened, there are

always some introductory message printed, informing you of the version of R you are using, how to cite R if you use in as part of a journal article and so on. At the bottom of all of this, you can see a read prompt symbol > (it may be a different color to red), followed by a blinking 'I'. Whenever it is blinking, that means the console is active, and so if you type in anything now it will be read into the console. However, if you click on either of two right panels, then the 'I' will stop blinking and become a slightly faded color. This means your console is now inactive, and anything you type won't be read in.

- The top right panel will contains several tabs. In the case of Figure 1, you can see three tabs in *Workspace*, *History*, *Help*. We'll explain what *Workspace* is shortly. *History* is generally not too useful, because you will learn a better way to save your work and history. *Help* is an a very helpful tab: Anytime you want to search what a particular command in RStudio does, click on *Help*, type what you want in the search bar at the top right hand corner (with the magnifying glass) and press 'Enter'.

- The bottom right panel also contains several tabs. In the case of Figure 1, you can see three tabs in *Files*, *Plots*, *Packages*. *Files* is a bit like Windows Explorer: it shows everything in your "working directory" (we will discuss what this term means later). *Plots* is another useful tab, which is automatically 'clicked on' whenever you construct a plot (we shall see this later). The *Packages* tab will not be used in this course, but comes in handy when you want to download add-ons or packages to do more advanced statistical analysis.

Note: Depending the computer different tabs maybe located in different places e.g., the *Help* tab may be on the bottom right panel instead of the top right. In ALL of the notes below, we shall go by the tab positions seen in Figure 1, but it should not be too hard to located these tabs in your own computer (hopefully).

How to type data into RStudio

If your dataset is small, you can enter it in manually. For example, consider the dataset:

2.3 2.5 6.5 7.4 1.1 3.9

To enter it into RStudio and to store it as the object dat, type at the prompt

```
dat=c(2.3, 2.5, 6.5, 7.4, 1.1, 3.9)
```

then press 'Enter'. You can think of the 'c' as meaning to "combine" these bunch of numbers together.

You can now type dat and press 'Enter' to check that RStudio has recorded your data correctly – it should return to you the values you entered. In addition, if you look at the *Workspace* tab in the top right panel, you'll see that there is dat object has been created. You can now guess that the *Workspace* tab basically keeps a record of every object that you have created since opening RStudio.

Handy tip: Note that if you created a new object which has the same name as an object that already exists in the *Workspace*, then it overwrite the old object. For instance, if now type `dat=c(1, 2, 6, 4)` and press 'Enter', and then press check out what `dat` is, you'll see the old dataset has been replaced by your new one.

Now that `RStudio` has your data stored in it, you can do some calculations to summarise the variable `dat`, such as finding the mean, median, and so on. We will do this a little later. The object `dat` will remain in the `RStudio` memory i.e., in *Workspace* until you either overwrite it using another `dat=...` command, or until you close `RStudio`. This is true of all objects that are created and stored in `RStudio`.

Handy tip: If at any time you are typing into console and the prompt changes into a +, then it means `RStudio` believes you have not finished typing. In this case you have two options: I) press the escape key `Esc` or press `Ctrl+C` you will be back in business; II) finish your typing! For instance, if you type `2+`, then you'll see + come up on the next line. That is because `RStudio` reckons you haven't finished, and rightly so...what do you want to add to 2? If you now type `3`, then you'll see the answer `5` pop up, followed by >, and you are back to business.

How to import a text file into `RStudio`

When entering data into a computer, people typically use a spreadsheet such as Excel, or some other program. We can take data from these other programs and import it into `RStudio` for analysis. This section explains how to do this for a dataset that has been stored in tab-delimited format.

Download and open the file `survey.txt`. This file contains past student responses to the survey administered to UNSW first year statistics students at the start of semester. It is a tab-delimited file, which means that the 'Tab' key has been used to distinguish between columns of the dataset. There are many columns in the dataset, and each has a label in the first row identifying which variable it is *e.g.* travel.time. Notice that the column labels have no spaces in them – this is important because `RStudio` may have problems reading the dataset if there are spaces in any of the column labels.

To import the dataset in `survey.txt` into `RStudio`:

- First save `survey.txt` to your computer. It is HIGHLY recommended that you save any text file downloaded in to your target directory.

- Now change `RStudio`'s "working directory" to the the same directory, assuming you followed the step above and hence have `survey.txt` contained there. This can be done by clicking on **Session...Set Working Directory...Choose Directory...** and navigating.

  So what's this thing called "working directory"? As the name suggests, it's the directory where `RStudio` works from i.e., the place where `RStudio` can find datasets and script files (we'll get to later).

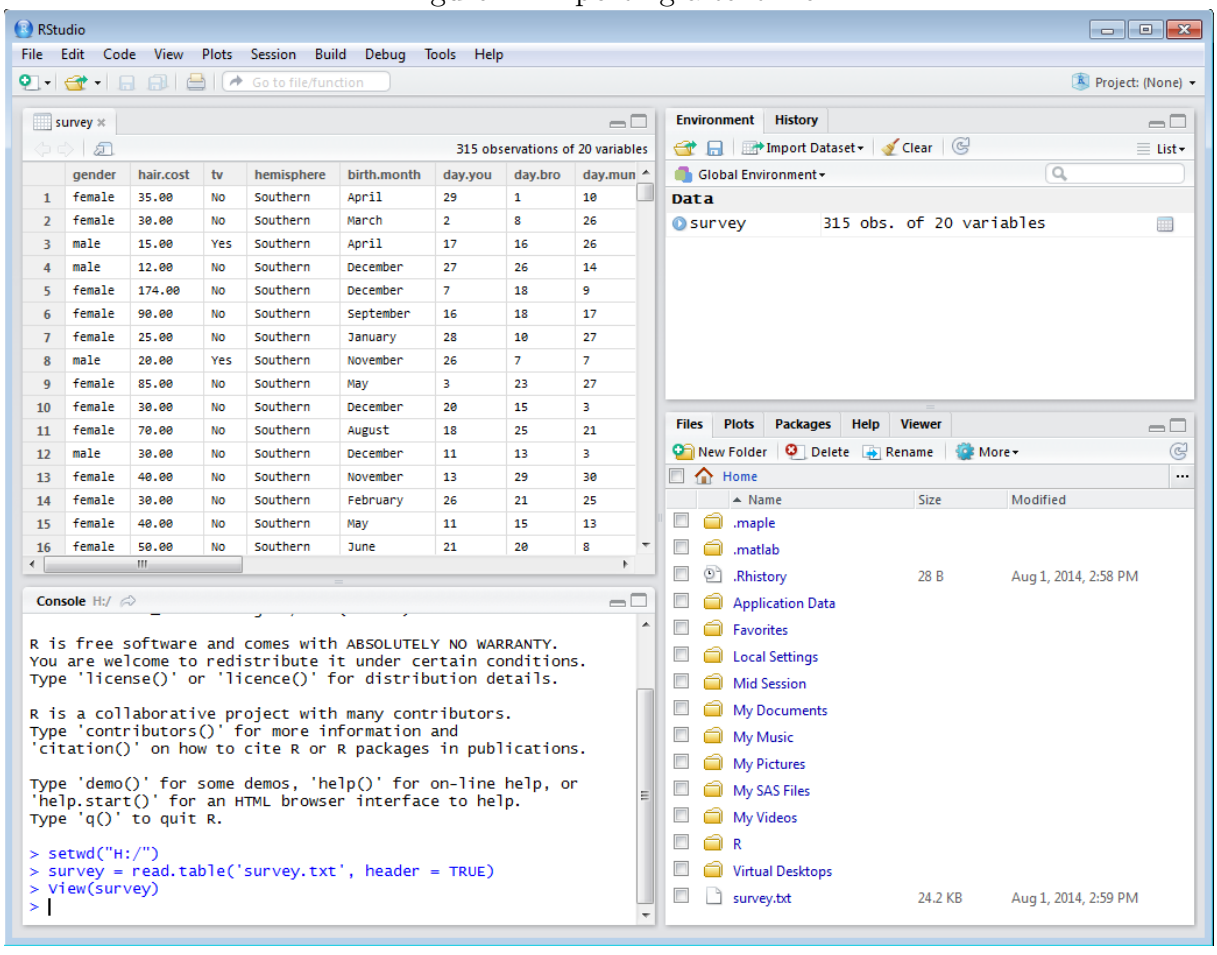
- To load the file `survey.txt` and to store it under the name `survey`, type at the prompt

  ```
  survey=read.table("survey.txt",header=T)
  ```

  `read.table` is the `RStudio` command which reads data in from a text file, and `header=T` tells `RStudio` that the first row of the file is a "header row" which contains the names for each column. Note that we called the dataset `survey`, but you are free to name it anything you want!!! Just because the text file is called `survey.txt` doesn't mean it could has to called `survey` in `RStudio`...call it `myfirstrdataset` or `iaminlovewithr` if you felt like it, although see then handy tip below.

- You'll notice in the *Workspace* tab that there is now a new object called `survey` (or whatever you decided to name it). This is seen in Figure 2 where in the top right panel, `RStudio` informs us that `survey` contains 310 observations or rows of data, and 20 variables or columns of data. What's more, if you actually click on this `survey` in the *Workspace* tab, you'll notice that a preview of the survey dataset as read in by `RStudio` pops up in the top left panel (in turn shrinking the console to the bottom right panel). Again, we this is Figure 2. If you want to close the preview of the dataset, just click the '×' next to the newly opened `survey` view.

Figure 2: Importing a text file



Handy tip: It is good practice to give objects in `RStudio` informative names. For example in the above, we called the dataset we read in `survey` because it was a survey of

responses. Generically names like `dat` are not just uninformative, but you run the risk of constantly overwriting objects if you give everything vanilla names like `dat`!

To check out the data (as well as to just check out that `RStudio` read things in correctly), there are some useful commands you can use. First, if you just want to view the entire file, then you can type:

```
survey
```

Although clicking on the object in the *Workspace* tab is probably more useful. If you just want the first 10 rows of the dataset, then type:

```
survey[1:10,]
```

The

```
1:10
```

is a shorthand in `RStudio` for writing `c(1,2,3,4,5,6,7,8,9,10)`. Alternatively, if you want the first second and fourth columns:

```
survey[,c(2,4)]
```

If you want a more general glimpse of what the beginning or 'head' of the dataset looks like:

```
head(survey)
```

Handy tip: We always recommend that one of first, if not THE FIRST, thing you do once opening `RStudio` is to set your working directory wherever your data are located. Start in a place where everything is located, and life's a breeze.

Note: In this course, we shall be working almost exclusively with CSV files instead of text files. However, the method to import CSV files into `RStudio`are more or less the same as the above, except instead of `read.table`, we use `read.csv`!

How to import data from Excel into `RStudio`

We will import the `smokePregnant.xls` dataset (which an Excel spreadsheet) into `RStudio`, by first saving it as a CSV file, then importing this file into `RStudio`. First, download `smokePregnant.xls` and save it to your target directory.

To save `smokePregnant.xls` as a CSV text file:

- Open `smokePregnant.xls` in `Excel`. Then highlight, right click and select delete to remove the top five 'useless' rows of the dataset, so that row 1 contains the variable names, and the rows immediately below contain the data.
- Use **File...Save as...** to save the data as a CSV (*.csv) file titled `smokePregnant.csv` in your target directory. This document is in a form that `RStudio` can read.
- Now open `RStudio` set your working directory to the target directory, and then type:
  ```
  smoke.preg=read.csv("smokePregnant.txt", header=T)
  ```

This command stores the dataset from `smokePregnant.csv` as the object `smoke.preg`, and uses the first row of the dataset to label each variable.

To check that `RStudio` has stored the dataset correctly, type
`smoke.preg`

or click on the `smoke.preg` object in the *Workspace tab*.

How to save your work

Because `RStudio` is a command-line language, you don't usually need to save every object that has been created. Instead, it is usually sufficient to just save the commands you typed into `RStudio`. We'll show you two ways to do this. The first is simple but not very practical. The second requires a little bit more fiddling but is AWESOME.

Saving your commands

To keep a history of every command you have typed during since opening `RStudio`, click on the *History* tab in the top left panel. You will now see a record of all the commands that you types in. Now click on the disk symbol, and save this record in your target directory, giving it any name you want followed by the '.txt' format.

A text file will be created, which will list every single command you have typed in your current `RStudio` session. In principle this is really cool, because you can then rerun your analyses at a later date by simply opening this text file, and copying and pasting the commands into `RStudio`. In other words, if you ever need to go "back to the drawing board" to check out some commands or do some revision (for a lab test maybe!), then it is easy to repeat your analysis on `RStudio` – just copy and paste all commands from this history file.

However, constantly flicking back and forth copying and pasting is very tedious. Also, you'll see that the text file has all the text in black. Can you imagine how hard it would be if you had hundreds of commands in your history file and you are just looking for one of them? We now look a much better way of doing things, which is virtually what all practicing statisticians using `RStudio` do.

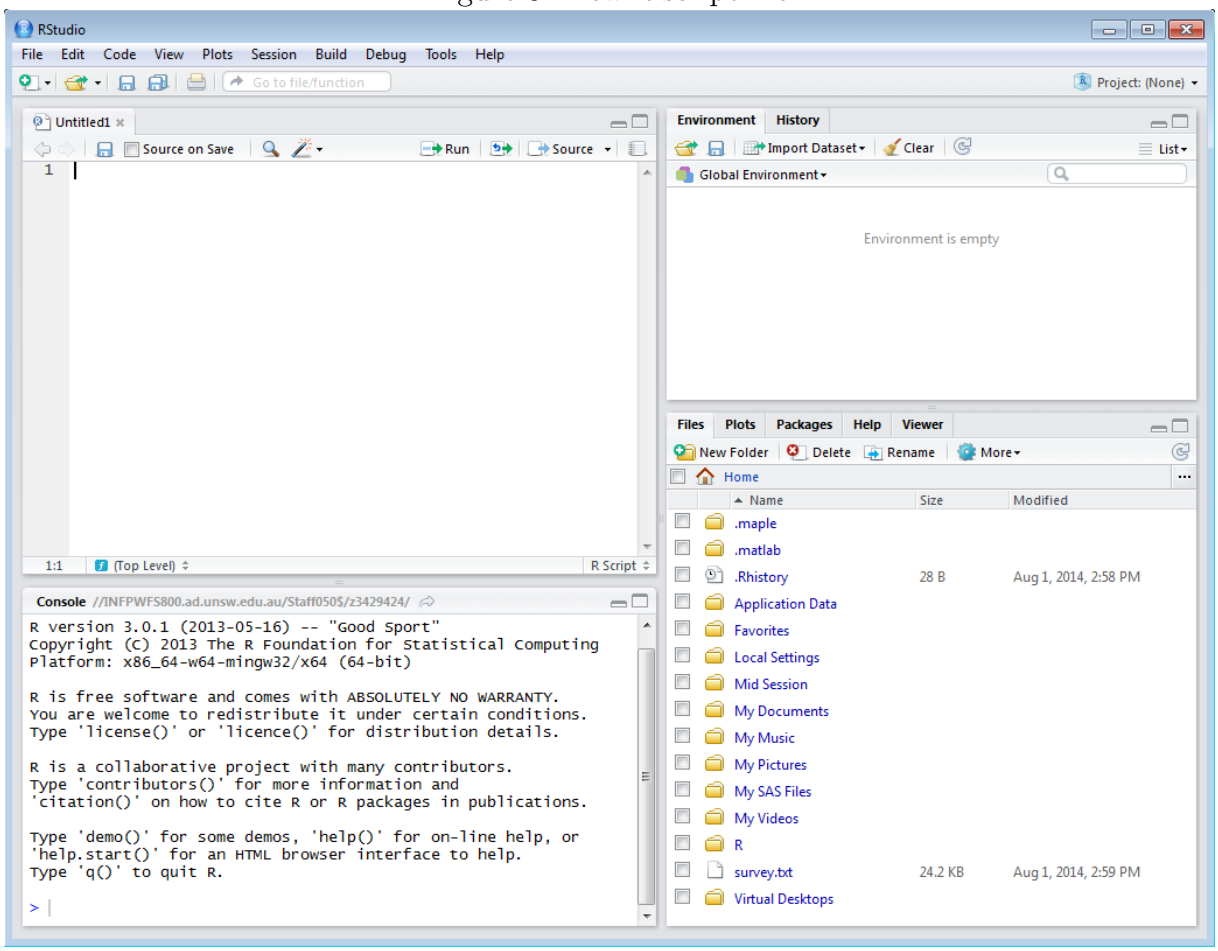
Using and Saving `R` script files

Rather than saving typing all your commands in the console (left pane) and then saving a record before you exit, a much better way of doing things is to write all your commands from the beginning in a `R` script file. This approach offers a number of advantages, which you shall witness below.

First, once you have opened `RStudio` (and set your working directory to target directory, of course), click on the **File...New...R Script**. At this point, what you'll see happen is that the console on the left pane will shrink, and a `R` script file will open up as a new, top

left panel names with the name `Untitled 1`; see Figure 3. Note that previously the top left panel was also where datasets were previewed; see Figure 2.

Figure 3: New R script file



This is now your R script file, where you can write whatever you need. The key thing however is with this approach, rather than copying and pasting commands into the console on the panel below, you can type the commands in the R script file and directly execute from there! For example, in Figure 4, you can see that a line of data called marks has been typed into the R script file. Note this has NOT been run yet – it's simply sitting in the script file.

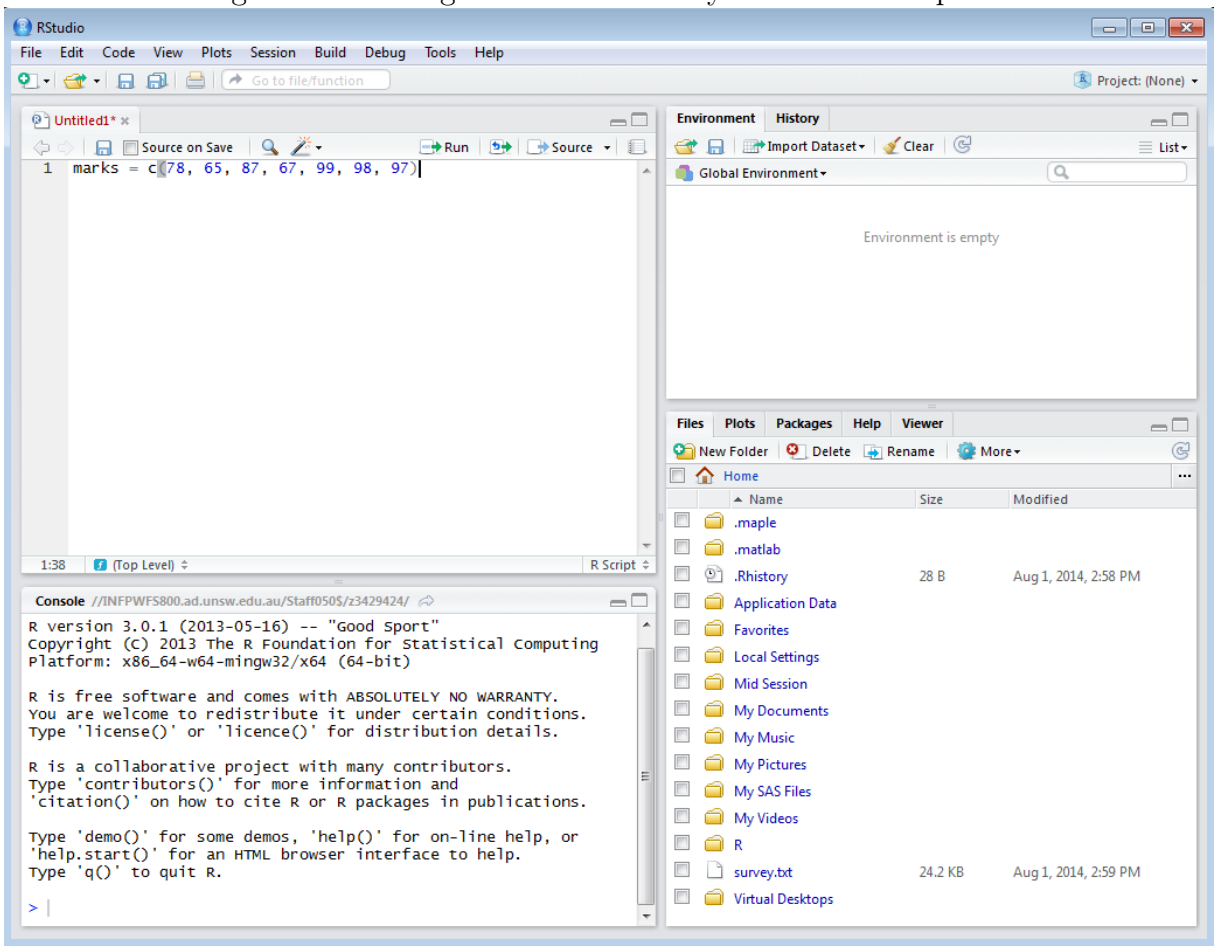
To run this command, ensure that the blinking 'I' is somewhere on that line (anywhere on that line will do). Then move your cursor over to the 'Run' button on the right and press it. Once you've pressed you, you'll notice that the command has been run in the console on the bottom left panel. That's the beauty of an R script file!

If there's a whole bunch of commands you want to run at once in your R script file, then what you can do in highlight them all (you'll notice the highlighted lines will be in highlighted blue), and then press the 'Run' button.

Handy tip: If you prefer to use the keyboard, then instead of pushing the 'Run' button, you can press 'Ctrl + Enter' while the blinking 'I' is somewhere on that line.

Aside from running commands directly off it, another major advantage of an R script file

Figure 4: Running commands directly from the R script file



is syntax assistance:

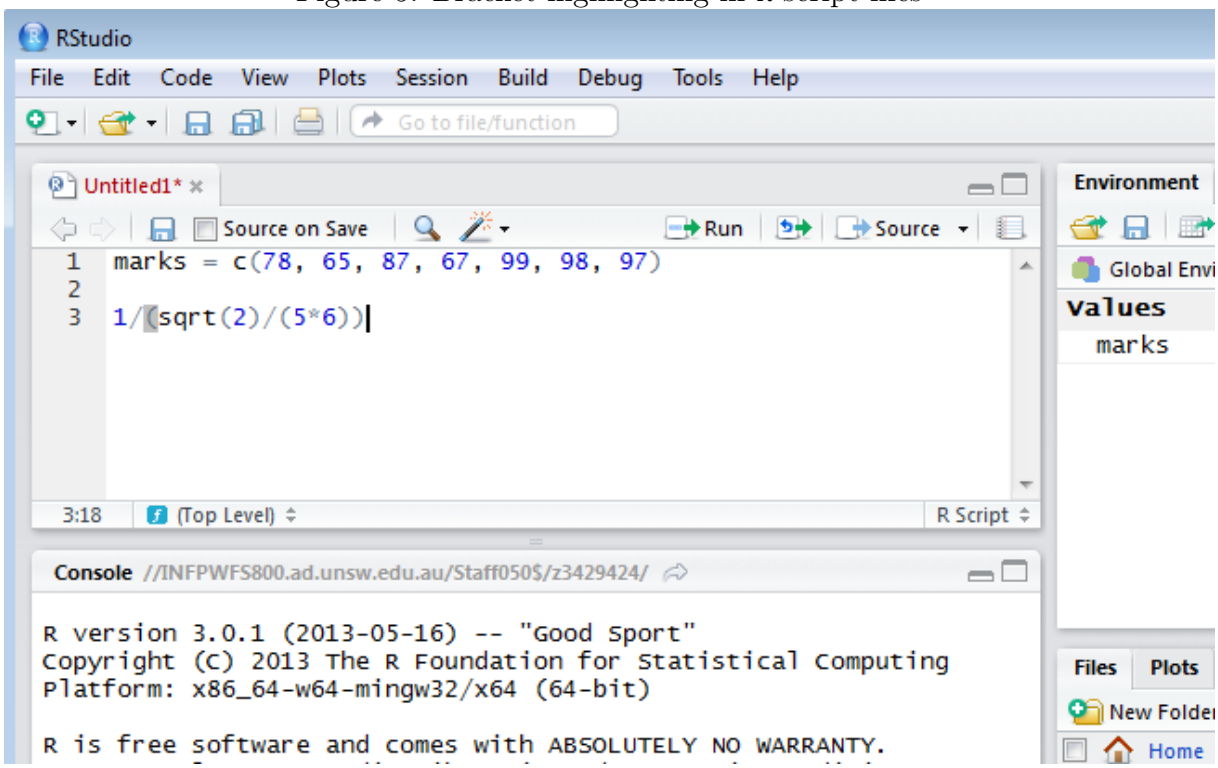
- RStudio will automatically highlight corresponding brackets. This is seen in Figure 5, where there command is designed to calculate $1/(\sqrt{(2)}/(5 \times 6))$. You'll see that my 'I' is at to the right of the final ')' bracket, and what RStudio has done is highlight the corresponding '(' bracket.

- Bracket completion. This is kind of related to the above. If you type an (is an R script file, then RStudio will automatically put a ')' to complete bracket AND move the 'I' blinker to in between those brackets.

- Basic syntax coloring. R scripts files come with some basic coloring to help differentiate things. This is illustrated in Figure 6 (don't worry about the commands for now...in time you'll learn what they are doing!). Most text is colored in black, all symbols are colored in purple, and anything inside quotation marks are colored in blue. There's also some green text, which we'll talk about now.

- You may also want to write titles for different sections and comments in the R script file as you go along, to remind yourself what you are doing and what the results mean. This is really useful if you plan to return to these script files later on (maybe for revision!). To put comments in, use the special comment hash character #.

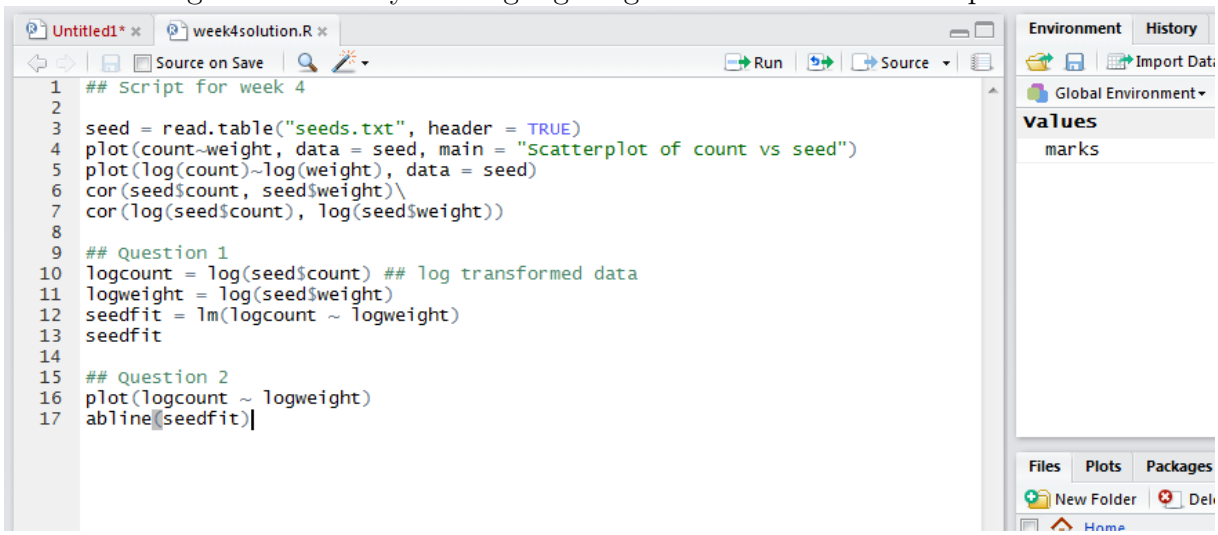
Figure 5: Bracket highlighting in R script files



Anything you write after # and which is before a new line in the R script file, will appear in green and be a comments. This is seen in Figure 6, where a title 'Script for Week 4' has been given to the file, and some comments and sub-headings here.

Note that a double hash ## has been used in Figure 6 – there's reason for this, just force of habit!

Figure 6: Basic syntax highlighting and comments in R script files



Once you have types all your commands, comments, and run some stuff off your R script file, it's time to save it (if you want to, of course). To do this, make sure your there is blinking 'I' somewhere in the R script file – this ensures that it is the script file which is

active and not the console or the other panels. Then you can either go to **File...Save** or click the disk symbol below `Untitled 1`. Then navigate to your target directory, give a name to your `R` script file and click 'Save'. The only thing you have to remember is that instead of a '.txt' format, please ensure you use a '.R' ending. This ensure the saved file is an official `R` script file. Once your file is saved, you'll notice that `Untitled 1` is replaced by your filename e.g., `week4solution.R` in Figure 6.

Finally, note that you can have multiple `R` script files opened at once. In Figure 6 for instance, there is an `Untitled 1` as well as an `week4solution.R` script file opened. This is analogous to having multiple tabs in Firefox, and comes in handy if you are working on several `R` script files at once (which we won't be doing in this course).

Handy tip: Another awesome benefit of using `R` script files is that `RStudio` is automatically linked to it. That is, you can go to Windows Explorer, navigate to your target directory, right click on an `R` script file which you have saved previously, then choose to open with `RStudio` directly! This saves you from having to open `RStudio` and then look for the script file. The other major advantage of opening `R` script files this way is that `RStudio` will AUTOMATICALLY set the working directory to the directory where you opened the script file from i.e., your target directory.

Copying numerical output

Often, when preparing a document summarizing the results of an analysis, it is most convenient to present results using a Word processor such as Microsoft Word. There are two types of `RStudio` output that you might want to copy from `RStudio` across to Word when writing up a report or assignment, as below.

To do this, we will use standard copy-and-paste techniques. That is, we will highlight what we want to copy, then go to **Edit...Copy**, then we will move where we want to paste the output and go **Edit...Paste**.

- First, you will need some `RStudio` output that you want to copy across to Word. For example, calculate a five-number summary for the travel time variable form the `survey` dataset as described in Section : Numerical summaries of a quantitative variable.

- To tell the computer what you want to copy from `RStudio`, highlight the output by starting at the top left corner of the output, then holding the mouse button down while dragging the mouse across the output, such that the whole of the output to be copied is highlighted. Then go to **Edit...Copy**.

- Open a (new) word document. To save the five-number summary from section into an empty document, simply go to **Edit...Paste**.

Constructing and Saving Graphs

RStudio makes it very easy to save any plots you have constructed. In the below, you will see how to save plots in PDFs format. Once you have done this, then you can open the plot in Acrobat Reader (for instance) and copy and paste sections it into Microsoft Word, or print them, or whatever you want.

- First, create a graph. For example, work through section : How to graph one quantitative variable using RStudio.
- When relevant command to create a graph is run (from your R script file), you notice the bottom right panel jumps to the *Plots* tab, and your plot is there is all its glory!
- To save the graph as a PDF, click on **Export...Save as PDF**. A new window will not open up. You can change the size and orientation plot, although usually the default is good enough. See Figure 7.
- Importantly though, click on **Directory** and navigate to your target directory if you're already there (the default directory to save graphs is your working directory).
- Change the filename to something useful or just leave as it is as Rplot. Note however you do NOT need to put a .pdf format at the end of the filename – RStudio already knows you are exporting as pdf!
- Click 'Save' and your plot is saved!

Handy tip: Once you've constructed your plot, if you want to look at it in more detail, click the **Zoom** button and a bigger image will pop up. Close it when finished.

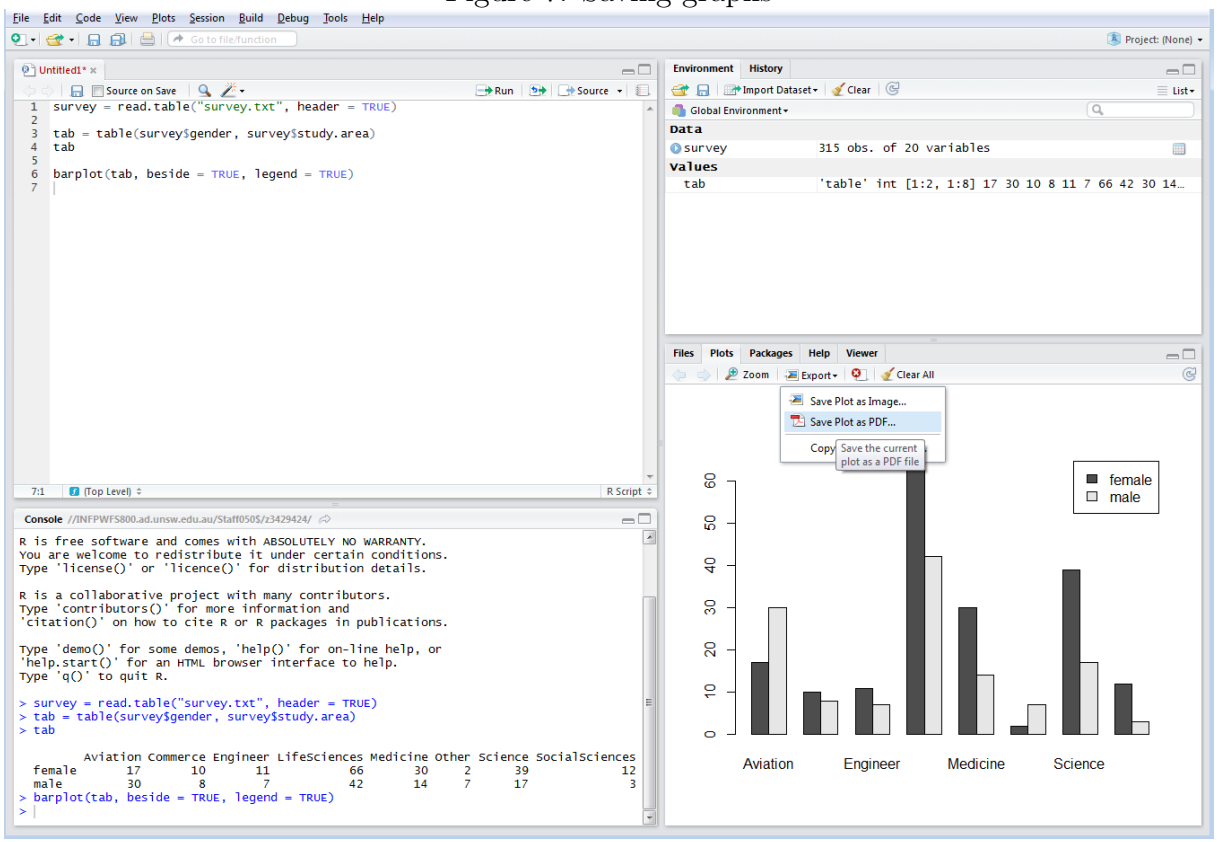
Handy tip: RStudio is smart enough to keep a records of all the plots you've made in a session. You can check previous plots by clicking on the left and right arrows.

Getting help on RStudio

There are a few different ways to find help on RStudio, if you want more information on anything:

- On the *Help* tab in the top right panel, search for the command of interest in the search bar with the magnifying glass
- To search for help on a particular technique, use the `help.search` command, *e.g.* to find out what functions are available for constructing histograms, type
  `help.search('histogram')`
  in your R script file and run it.
- If you know about a particular function enter a question mark '?' followed by the command *e.g.* to find out more about the `hist` command, type
  `?hist`

Figure 7: Saving graphs



- Google is your friend! There are also awesome brief introduction manuals out there e.g., 'The R Guide', which you can download from http://cran.r-project.org/doc/contrib/Owen-TheRGuide.pdf

Exiting RStudio

At the end of any RStudio session, after you have saved your R script file, quit RStudio by going to **File... Exit**, or pressing the '×' at the top right hand corner. Note that you'll then be asked whether you want to save your workspace i.e., all the objects located in the *Workspace* tab; see Figure 8. You can safely press DON'T SAVE, because you have already got a history of your commands in your R script file – any objects can always be recreated when by running commands off the script file.

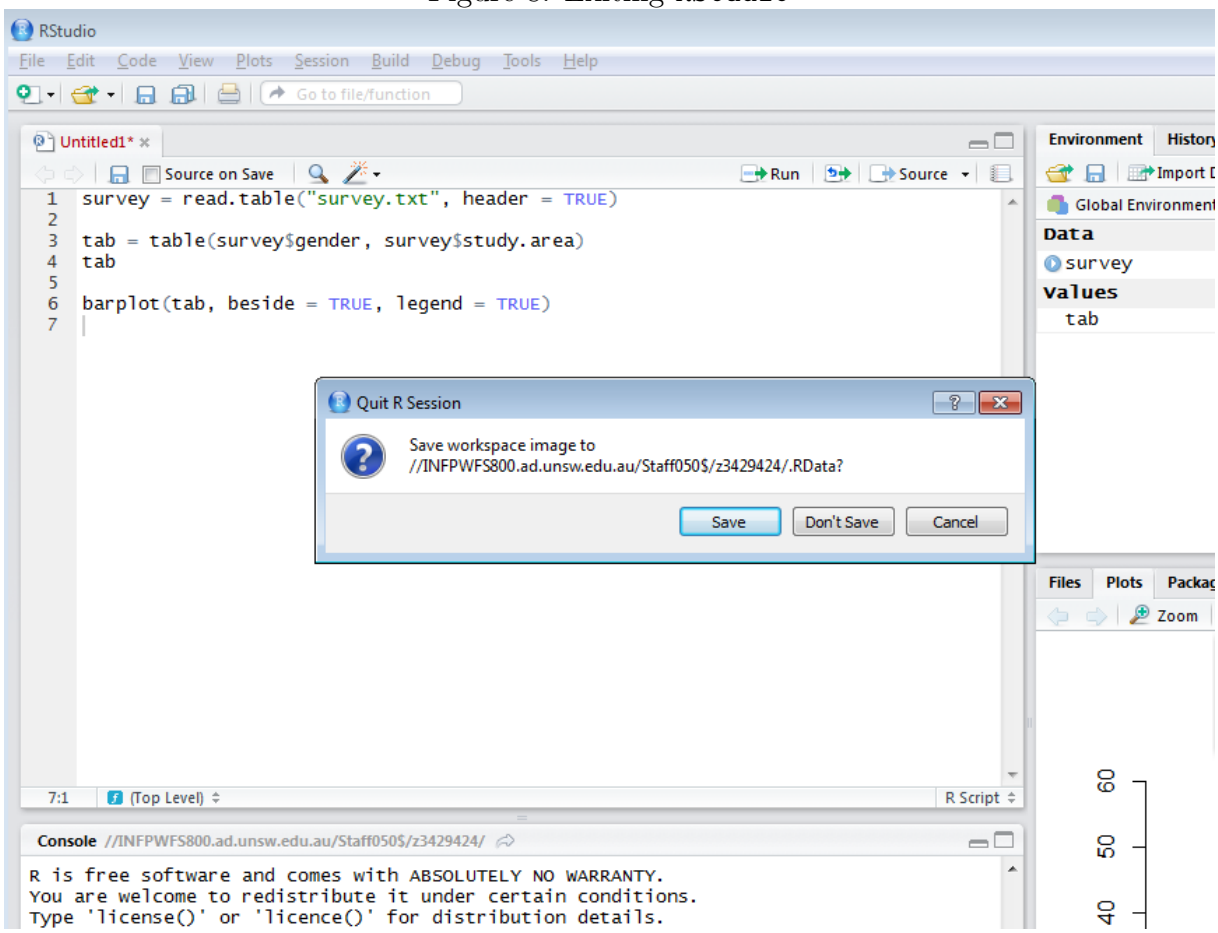
Quick fire steps

So here is a list of quick fire steps you are recommended to run

1. Open RStudio, or go to an R script file and open with RStudio on this.

2. Set working directory via **Session...Set Working Directory...Choose Directory...**. If you opened an R script file in Step 1 then you can skip this step.

Figure 8: Exiting RStudio



3. Open a new R script file via **File...New...R Script**. If you opened an R script file in Step 1 and you want to work on this now, then you can skip this step. Remember you can have multiple R script files open at once, much like multiple tabs in Firefox.

4. Type whatever commands you want in the R script files, and run them by pressing 'Run' or pressing 'Ctrl + Enter'.

5. Save your R script files by going to **File...Save**

6. When you're finished, go to **File... Exit**, and click 'No' to saving the workspace.

Summarising data graphically using RStudio

R and RStudio is a powerful graphical tool – you can create all sorts of different graphs, usually pretty easily too! In particular, it is easy to construct boxplots and normal quantile plots using RStudio, whereas this is very difficult when using Excel. The one drawback compared to Excel is that to make graphs "picture perfect" with titles and axis labels in the right place etc...is rather tricky (more like an 'art form'!). In this course though, we want to have appropriately labelled graphs, but not a 'perfect work of art'.

Note also that for all of the below plots, you can change the labels of the X-axis, the

Y-axis or the main title. To do this, use the `xlab`, `ylab` and `main` options respectively. This is illustrated below for bar charts.

How to graph one categorical variable using RStudio

To construct a bar chart of the `gender` variable from the `survey` dataset, use the command
`plot(survey$gender)`

The syntax `A$B` sign means "in dataset A, find a column called B". If column B can be found, then the command will run without problems. However if there is no column B in dataset A, then you'll errors popping up in the console on the bottom right panel.

To change the X-axis title to "Male or female?" and to change the main title to "A bar chart of the gender of statistics students":
`plot(survey$gender, xlab="Male or female?", main="A bar chart of the gender of statistics students")`

Could you modify the above so that the y-axis has the label "Frequency"?

How to graph one quantitative variable using RStudio

To graph the `travel.time` variable from the `survey` dataset as a boxplot, type
`boxplot(survey$travel.time)`

For a histogram:
`hist(survey$travel.time)`

How to Graph one Categorical Variable Against Another

A clustered bar chart is an effective tool for displaying the relationship between two categorical variables. RStudio can construct a clustered bar chart from a table of frequencies. To construct a table of frequencies of the relationship between gender and study area for the `survey` data, and store it as the object `tab`:
`tab=table(survey$gender, survey$study.area)`

Have a look what the created object, `tab`, looks like, by typing `tab` and running it.

Now to construct a clustered bar chart based on this table, including a legend:
`barplot(tab,beside=T,legend=T)`

The `T` is shorthand of `TRUE`. In this case, this command is telling RStudio to put the bars corresponding to 'male' and 'female' next to each other, and to make a legend

Alternatively, if you have already summarized your data in a table and you want to enter your table of frequencies directly into RStudio and store them in a `table`, you can do this as described in Section .

How to graph a quantitative versus a categorical variable using RStudio

To construct comparative boxplots for the hair cut costs of different genders, type any **one** of the following:

```
plot(survey$hair.cost~survey$gender)
boxplot(survey$hair.cost~survey$gender)
plot(hair.cost~gender, data=survey)
```

Note that most functions have an optional `data=` argument so you don't have to use the "`survey$`" form every time you want to use a variable from within the dataset `survey`.

The "~" stands for "against", that is, `plot(hair.cost~gender, data=survey)` means "plot `hair.cost` against `gender`, for the dataset `survey`." In other words, for each category of gender, make a boxplot of hair.cost.

How to graph one quantitative variable versus another quantitative variable using RStudio

In this part, use the `concept` dataset. This dataset lists the grade point averages (GPAs) and IQs of students at various ages. Download the text file version named `concept.csv` and save it to your target directory. Once you have set the working directory correctly, read the data into `RStudio` using the command

```
concept=read.csv("concept.csv",header=T)
```

To construct a scatterplot to compare GPAs and IQs, use the command

```
plot(concept$GPA~concept$IQ)
```

or equivalently,

```
plot(GPA~IQ, data=concept)
```

Note the use of the "~" again. This time is means "plot `GPA` against `IQ`" In other words, GPA goes on the y-axis and IQ on the x-axis.

Adding a trend line to a scatterplot

Continuing from the previous section:

```
reg=lm(GPA~$IQ, data=concept)
abline(reg)
```

You'll learn what the `lm` and command does latter on in the course. The `abline` command is a general function for adding straight lines to a scatterplot.

How to produce a normal quantile plot using RStudio

To produce a normal quantile plot of hair cost:

```
qqnorm(survey$hair.cost)
```

Have a look at the normal quantile plot. How well do you think the hair cost data approximate normality? If you think the data are skewed, are they left-skewed or right-skewed?

Summarising data numerically using RStudio

The below methods will be illustrated using the dat dataset that we entered manually in Section : How to type data into RStudio, and the survey dataset, which we imported from the survey.txt file in Section : How to import a text file into RStudio.

Numerical summaries of a categorical variable

The key function for numerically summarizing a categorical variable is the table function. We will illustrate the use of this function on the gender variable from the survey dataset.

After importing the survey.txt dataset and storing it as survey, type:
survey$gender

This command displays the raw data stored in the gender column of the survey dataset. Note that entries are either Male or Female, hence this variable is categorical. Note also that there are hundreds of entries, so we would like to create a numerical summary to make sense of them all!

Use the table function to numerically summarise the gender variable as follows:
table(survey$gender)

This returns a frequency table of the counts of the number of responses in each category.

It is also possible to use the summary function to obtain similar output.

Numerical summaries of a quantitative variable

Calculating numerical summaries of quantitative variables is pretty easy! With the dat dataset for instance,

- To calculate the mean, type mean(dat)
- To calculate the standard deviation, type sd(dat)
- To calculate the median, type median(dat)
- To calculate the 25th percentile (otherwise known as the first quartile), type quantile(dat,0.25)
- To calculate a five number summary, type summary(dat). Note that this actually returns a six number summary – it throws in the mean as well for good measure. It also calculates the quartiles slightly differently to how we did it in class, so if you do the calculations by hand you won't get exactly the same answers as RStudio.

All of these commands will also work for any quantitative variable in the survey, although you need specify which column or variable from the dataset you want to access. Recall that you did this via the '$' symbol e.g., summary(survey$travel.time).

How to numerically summarise the relationship between two categorical variables using RStudio

To numerically summarise the relationship between categorical variables, use the table function. Remember we also used this command for summarizing one categorical variable.

For example, we might wish to construct a table of frequencies to summarise the relationship between the gender and study.area variables, to look at how study area varies between males and females. To do this, type:

```
table(survey$gender, survey$study.area)
```

Handy tip: Remember we actually ran this command earlier in Section : How to Graph one Categorical Variable Against Another. The difference there was that we labeled the resulting table tab, because we wanted to construct a barplot from this table.

How to type in a table of frequencies

Sometimes, you may have a table of frequencies that has been calculated in another software package or given to you on paper, which you want to use for analyses in RStudio. For instance, you may want to construct a clustered bar chart as shown in Section : How to Graph one Categorical Variable Against Another.

Consider the following example:

	Virgin Blue	QANTAS
0-3 min late	22	11
3-15 min late	6	21
$>$ 15 min late	0	2

To enter the data into RStudio and store in a table called tab. Try running the set of commands:

```
Virgin=c(22, 6, 0)
QANTAS=c(11, 21, 2)
tab=as.table( cbind(Virgin, QANTAS) )
row.names(tab)=c("0-3", "3-15", ">15")
```

In brief, the strategy is to 1) type the data in on a per column basis, so here we have two columns of data, 2) use as.table(cbind(...)). The command cbind() tells RStudio to combine my columns together, while putting as.table() outside of this informs RStudio to treat this 'thing' as a table, 3) row.names(tab) =... is not actually required, but it almost always helps to label for each row in your table is talking about, in this case the # of minutes late.

Handy tip: Remember with R script files, you can run a whole bunch of commands that are different lines simultaneously, by highlighting them all and then clicking 'Run' or pressing 'Ctrl + Enter'.

After running all of the above commands, you can type `tab` and then run this to have a look at your pretty table! Afterwards, you could then use the `barplot` and `summary` commands on `tab` to construct a clustered bar chart and so on.

How to numerically summarise the relationship between a quantitative and a categorical variable using RStudio

Consider the case where we want to create a numerical summary of the relationship between a quantitative variable and a categorical variable. For example, for the `survey` dataset, we might want to look at the relationship between the variables `hair.cost` and `gender`, to see how male and female students differ in how much they spend at the hairdressers.

What we want to do here is to repeat the function for numerically summarizing a quantitative variable (as in Section : Numerical summaries of a quantitative variable) at each level of our categorical variable `gender`. This can be done using the function `by`. For example, to create a five-number summary of `hair.cost` for each level of `gender`:
`by(survey$travel.time, survey$gender, summary)`

To calculate the standard deviation of `hair.cost` for each `gender`:
`by(survey$travel.time, survey$gender, sd)`

How to numerically summarise the relationship between two quantitative variables using RStudio

We will illustrate how to numerically summarise the relationship between two quantitative variables using the `survey` dataset. So before attempting the below, import the survey data into RStudio as described in Section : How to import a text file into RStudio.

How to calculate a correlation coefficient

To find the correlation between travel time and hair cut cost:
`cor(survey$travel.time, survey$hair.cost)`

Remember you can only calculate correlations between two quantitative variables. If you accidentally ran `cor(survey$travel.time, survey$gender)` then RStudio will print out an error!

How to estimate a least squares regression line

To find the least squares regression line between travel time and hair cut cost:

```
mymodel=lm(hair.cost~travel.time, dat=survey)
mymodel
```

`lm` stands for "linear model". We will learn more about this powerful command later on in the course in Section :How to make inferences about the relationship between two quantitative variables.

The output from `mymodel` contains two numbers in a section titled `Coefficients:` One number is labeled `(Intercept)`, and this is the Y-intercept (which we labeled a in lecture notes, for the line $y = a + bx$). The other number is labeled `survey$travel.time` and this is the slope (which we labeled b in the lecture notes, where $y = a + bx$).

Manipulating numbers using RStudio

How to do simple arithmetic using RStudio

Most standard arithmetic can be done from the RStudio command line as you would expect.

For example, to find $2 \times 3 + 1$: `2*3+1`

To find $\sqrt{20}$: `sqrt(20)`

To find the square root of every integer from 1 to 20: `sqrt(1:20)`. Recall that `1:20` in a shortcut for listing all integers from 1 to 20.

You can also do arithmetic using objects you have previously calculated and stored on RStudio. For example, to find the value that is 2 standard deviations above the mean travel time for the `survey` data:

```
mnTrav=mean(survey$travel.time)
sdTrav=sd(survey$travel.time)
mnTrav + 2*sdTrav
```

Transforming data

Sometimes we want to transform a variable before analysis – whether to try and satisfy assumptions of an analysis method, or to put data onto a different and hopefully a more interpretable scale.

Data transformation is really easy on RStudio. For example, let's say we want to log-transform hair cut cost data before constructing a histogram. Once you've loaded in the `survey` data, type:

```
log.hair=log10(survey$hair.cost)
```

The $\log_{10}$-transformed hair cost data is now stored in the object `log.hair`. To plot a

histogram to see what it looks like, type:
`hist(log.hair)`

Compare this to a histogram of the original data. You can do this on the same display by first asking `RStudio` to construct two graphs per page:
```
par(mfrow=c(2,1))
hist(survey$hair.cost)
hist(log.hair)
```

The `par(...)` is a ridiculously powerful command for fine-tuning and perfecting graphs. You won't be using it much in this course, because `RStudio` is normally quite good with the default options. In case though, `par(mfrow=c(2,1))` informs `RStudio` that you want to plot two graphs in a (2 row $\times$ 1 column) format.

Handy tip: Remember if you want you look at your graph in more detail, you can always press 'Zoom' in the *Plots* tab.

Notice that whereas the hair cut cost data were strongly right-skewed, the log-transformed data are fairly symmetric!

Other types of transformations you may want to use include the square root, `sqrt(...)`, the natural log transform, `log(...)`, and the reciprocal, `1/...`.

Combining variables

Sometimes we want to combine variables together, *e.g.* calculating the average of two variables. This is actually a special case of data transformation (where the transformation is a function of two variables) and it can be done in `RStudio` using the same method used for transforming data above.

To use the `survey` data to find the average of the days of the month on which students' parents were born:
```
day.avg=(survey$day.mum + survey$day.dad) / 2
```

To plot a histogram to see what it looks like, type:
```
hist(day.avg)
```

How does this compare to the distribution of an individual person's birthday?

Counting the number of values satisfying a condition using `RStudio`

To find out how many students take longer than an hour to get to uni, then type the command:
```
sum(survey$travel.time>60)
```

To understand what this command is doing, type
```
survey$travel.time>60
```

Note that `RStudio` returns a variable that contains a series of `TRUE` and `FALSE` values, which has as many entries as the original `survey$travel.time` variable has. A value is

TRUE when the corresponding entry in survey$travel.time is greater than 60, otherwise it is set to FALSE. To find the number of values which are TRUE, we simply sum over this condition command.

Another way to think about it is that RStudio treats TRUE as equal to 1 and FALSE equal to 0. Thus summing over this means you are only summing over students exceeding an hour travel time.

Sometimes you might want to sum the values satisfying some other type of logical condition. Here are a few other types of conditions worth knowing:

- survey$travel.time == 60 means "travel time is *equal to* 60"
- survey$travel.time != 60 means "travel time is *not equal to* 60"
- survey$travel.time < 60 means "travel time is *less than* 60"
- survey$travel.time >= 60 means "travel time is *greater than or equal to* 60"

Probability and experimental design using RStudio

How to generate random success/failure results, where the probability of success is p

Often, when experimenting with the notion of probability, we want to generate random numbers to do a **computer simulation**. This section explains one method of simulating data so that there are two possible responses, "success" and "failure", with the probability of success is equal to p.

To generate k random success or failure responses, each with probability of success p, use the command runif(k)>p.

For example, to generate 10 random numbers which have probability 0.7 of success, and to assign them to the variable x, type:

```
x=runif(10)>0.7
```

Then you can check out the result by typing x. Notice that each element in x is either TRUE (which means "success" or 1) or FALSE (which means "failure" or 0). Briefly, what the runif(10) does is tell RStudio to random generate 10 numbers between 0 and 1. The >0.7 is then exactly the same as what we saw in Section : Counting the number of values satisfying a condition using RStudio.

Let's say our trial involves the number of successful shots from the free throw line at basketball. In this case, it would be nice to be able to label the successes "score" rather than "TRUE" and the failures "miss" rather than "FALSE". To do this, check out:

```
y=rep("miss",10)
y[x==T]="score"
y
```

What does the above do? The `rep("miss",10)` means to repeat the word "miss" ten types. The next line, `y[x==T]="score"` is where the awesomeness lies – basically, it informs RStudio to search out which of the ten elements of `x` are equal to `TRUE`, and then change the corresponding elements in `y` from "miss" to "score". Notice how powerful this command actually is – it is changing the elements of one object (`y`) based on what's happening in another object (`x`)!

Often it is of interest to count the number of successful trials. Along the lines of Section : Counting the number of values satisfying a condition using RStudio, we could type `sum(y=="score")`

Generating random numbers

RStudio has a number of functions designed for random number generation. All start with the letter `r` for 'random', followed by an abbreviation for the distribution you want to generate random numbers from.

Distribution	RStudio function for k random numbers
Uniform (0 to 1)	`runif(k)`
N(mu, sigma)	`rnorm(k,mu,sigma)`
B(n, p)	`rbinom(k,n,p)`

So for example, to generate 10 uniform random numbers between 0 and 1:
`runif(10)`

Or to generate 100 random numbers from $N(3, 2)$:
`rnorm(100,3,2)`

Success and failure data can be thought of as binomial with $n = 1$. So the `rbinom` function can be used to generate success/failure data, as an alternative to the `runif(k)>p` approach suggested in the previous section.

Can you generate 10 random success/failure numbers which have probability 0.7 of success, and assign them to the variable `x`?

How to randomise the order of a set of subjects

The standard computer-based technique of randomizing a set of objects is to first assign each object a random number, then to sort these random numbers. This gives us a random ordering of the experimental subjects, so that we can partition them to treatment groups.

For example, to randomise the assignment of 6 subjects into three groups, each of size 2:

- Give each experimental subject a number, from 1 to 6.
- `ran=runif(6)` will create a vector `ran` that contains 6 random numbers.

- `sort(ran, index.return=T)` will sort the vector of random numbers. The first object returned `$x` is the sorted random numbers, which we aren't interested in. The second object returned `$ix` is a random reordering of the numbers 1:6 (which corresponds to the reordering of `ran` required to sort its values).
- The first two numbers in `$ix` tell us which subjects to assign to the first treatment group, the second two numbers tell us which subjects are assigned to the second treatment group, and the last two are assigned to the last treatment group.

If your subjects have names, then you can use `RStudio` to find a random rearrangement of their names. To do this for six subjects named Andrew, David, Frank, Ian, Jono and Victor, type:

```
names=c("Andrew", "David", "Frank", "Ian", "Jono", "Victor")
ran=runif(6)
sort.ran=sort(ran, index.return=T)
names[sort.ran$ix]
```

If we wish to assign these six people to three treatment groups (two to each group), then the first two names in the list tell us who has been assigned to the first treatment group, the second two names tell us who has been assigned to the second treatment group, and the last two names tell us who has been assigned to the last treatment group.

Handy tip: There's actually an inbuilt function in `RStudio` to do randomization, namely the `sample()` function. For example, check out `sample(names)`. Of course, if you want to know more about the guts of this function, you can always type 'sample' in the *Help* tab on the top right panel or run the command `?sample`.

How to calculate probabilities, quantiles and density curves for a normal distribution using `RStudio`

Instead of using standard normal tables to calculate probabilities and quantiles for the normal distribution, you can use `RStudio` to find exact values, as described below.

Note in the following that there are three key functions: `pnorm`, `qnorm` and `dnorm`. These function names all take their first letter from what the function is doing (they calculate a probability, quantile or density value, respectively) and the remainder of the function name is a description of the type of random variable the function works for (the `norm`al distribution). Remember we saw a similar syntax earlier with `rnorm` in Section :Generating random numbers.

There are many other functions on `RStudio` for calculating similar quantities for other special types of random variables, some of which are described in later sections.

Computing Probabilities for a Standard Normal Random Variable

The function `pnorm` returns cumulative probabilities for the standard normal distribution, that is, for $Z \sim N(0,1)$, `pnorm(z)` returns the value of $P(Z \leq z)$ for any value z i.e., find

the area to left of z.

For example, to find $P(Z \leq 1.96)$, type:
`pnorm(1.96)`

The function `pnorm` can be used for any other normal distribution as well, provided you include the mean and variance when you type your command into RStudio. That is, for $X \sim N(\mu, \sigma)$, `pnorm(z,`μ, σ`)` returns the value of $P(X \leq z)$ for any value z.

For example, to find $P(X \leq 4)$ where $X \sim N(3, 2)$, type:
`pnorm(4,3,2)`

Question: How do we calculate $P(X > 4)$?

Calculating quantiles for the standard normal distribution

The function `qnorm` returns quantiles for the standard normal distribution, that is, for $Z \sim N(0, 1)$, `qnorm(p)` returns the value c so that $P(Z \leq c) = p$ for any value p between 0 and 1 i.e., find that magical number c such that the area to the left of c is equal to p. You can think of quantiles are doing the reverse operation of finding cumulative probabilities.

For example, to find c such that $P(Z \leq c) = 0.975$, type:
`qnorm(0.975)`

This returns about 1.96, which means that the value c that satisfies $P(Z \leq c) = 0.975$ is $c = 1.96$. In other words, 1.96 is the value from the standard normal distribution that has 97.5% of observations falling below it.

Question: How do we calculate $P(Z > c) = 0.975$?

As expected, the function `qnorm` can be used for any other normal distribution as well, provided that you include the mean and variance in your input. That is, for $X \sim N(\mu, \sigma)$, `qnorm(p,`μ, σ`)` returns the value c so that $P(X \leq c) = p$ for any value p between 0 and 1.

For example, to find c such that $P(Z \leq c) = 0.7$ where $X \sim N(3, 2)$, type:
`qnorm(0.7,3,2)`

How to plot the density curve of a normal distribution

The function `dnorm` can be used to find the value of the density curve of any variable $X \sim N(\mu, \sigma)$ at a specified value x. Combining this function with the `plot` function, we can easily construct the density curve of any normal distribution.

For example, plot the density curve of $X \sim N(12, 3)$ for the range of values from 0 to 24:

- To first specify a set of values for which we will find the value of the density curve, between 0 and 24 and increasing by 0.1 each step:
  `x=seq(0,24,0.1)`

  Type `x` to see what the result looks like. Why 0 to 24? Well that's because it is the range where we the majority of the density curve to lie. Recall that 99.7% of

the area under a normal density curve is located ± 3 standard deviations from the mean.

- To find the value of the density curve for $X \sim N(12, 3)$ at each value of `x`:
  `f=dnorm(x,12,3)`

  Note the cleverness of `RStudio` – you can actually calculate the density for all elements of `x` simultaneously!

- To plot the density curve by joining the dots between each point from `x`:
  `plot(x, f, type="l")`

  Note: The type above is the letter, "l" for lines , and NOT the number 1

Do you know how to change this plot so that it has a suitable title? Hint: return to Section : Summarising data graphically using `RStudio`, to find out how to control the title and axis labels added to graphs.)

How to calculate probabilities for the binomial distribution

We can use `RStudio` to understand the binomial distribution, an important discrete random variable.

Evaluating a binomial probability

To find $P(X = k)$ where $X \sim B(n, p)$, use `dbinom(k,n,p)`.

For example, to find $P(X = 7)$ where $X \sim B(15, 0.3)$, type:
`dbinom(7,15,0.3)`

Evaluating a cumulative binomial probability

To find $P(X \leq k)$ where $X \sim B(n, p)$, use `pbinom(k,n,p)`.

For example, to find $P(X \leq 2)$ where $X \sim B(15, 0.3)$, type:
`pbinom(2,15,0.3)`

Question: How do we find $P(X < 2)$? $P(X > 2)$?

How to plot the probability histogram of a binomial distribution

To plot a probability histogram in `RStudio` for $X \sim B(48, 0.25)$, for the range of values of X between 0 and 24:

- First specify the set of values for which we will calculate probabilities i.e., from 0 up to 24, increasing by 1 each step:
  `xbin=0:24`

Remember that this is short hand of for all numbers `c(1,2,3,...,23,24)`. Type `xbin` to see what the result looks like.

- To find the value of the probability function for $X \sim B(48, 0.25)$ at each value of `xbin`:
  `fbin=dbinom(xbin,48,0.25)`
- To plot the probability function, use a barplot:
  `barplot(fbin, names.arg=xbin, space=0)`

  The `space=0` argument ensures no spaces between bars, like a histogram. The `names.arg=xbin` tells `RStudio` what is each is called. In this case, because it's a probability histogram, the 'name' of each bar is then the sequence of integers from 0 to 24. If you further want to **overlay a normal curve** on top of this plot, then try running:
  `lines(xbin+0.5, dnorm(xbin,12,3))`

  What was that `+0.5` for?

There are a few other ways to construct this plot on `RStudio`. You could plot the points alone using `plot(xbin,fbin)`, or you could plot vertical lines dropping down to the X-axis from each point using `plot(xbin,fbin,type="h")`...feel free to experiment

How to calculate probabilities and quantiles for a t distribution using RStudio

Instead of using t tables to estimate probabilities and quantiles for the t distribution, you can use `RStudio` to find exact values, as described below. The functions operate in an analogous manner to `pnorm` and `qnorm`.

Calculating probabilities for the t distribution

The function `pt` returns cumulative probabilities for the t distribution. If $T \sim t(k)$ where k is the degrees of freedom, then we can find $P(T \leq x)$ using `pt(x,k)`.

For example, to find $P(T \leq 2)$ where $T \sim t(4)$, type:
`pt(2,4)`

Note that this returns the probability **to the left**, which is not what Excel does, although it is consistent with what `RStudio` does for other distributions.

To find $P(T > 2)$ where $T \sim t(4)$, recall that $P(T > 2) = 1 - P(T \leq 2)$ and type:
`1-pt(2,4)`

Calculating quantiles for the t distribution

The function `qt` returns quantiles for the t distribution. If $T \sim t(k)$ where k is the degrees of freedom, then we can find the value c that satisfies $P(T \leq c) = p$ for any p between 0

and 1, using `qt(p,k)`.

Let's say we want to find the lower 12% point from the t distribution with 15 degrees of freedom. That is, we want to find c such that $P(T \leq c) = 0.12$ where $T \sim t(15)$. To do this, type:
`qt(0.12,15)`

Now let's say we want to find the upper 0.5% point of the t distribution with 15 degrees of freedom. In this case we need to do a little manipulation to get the answer, in order to turn this expression into a left-tailed probability. If $P(T \geq c) = 0.005$ then $P(T < c) = 1 - 0.005 = 0.995$. So we can find the upper 0.5% point of $t(15)$ using:
`qt(0.995,15)`

A more direct way of doing this is to type:
`qt(1-0.005,15)`

How to calculate probabilities and quantiles for the χ^2 distribution

Instead of using χ^2 tables to estimate probabilities and quantiles for the χ^2 distribution, you can use `RStudio` to find exact values, as described below. Again, the functions operate in an analogous manner to `pnorm` and `qnorm`.

Calculating probabilities for the χ^2 distribution

The function `pchisq` returns cumulative probabilities for the χ^2 distribution. If $X \sim \chi^2(k)$ where k is the degrees of freedom, then we can find $P(X \leq x)$ using `pchisq(x,k)`.

For example, to find $P(X \leq 5.2)$ where $X \sim \chi^2(5)$, type:
`pchisq(5.2,5)`

Note that this returns the probability **to the left**, which is not what Excel does, although it is consistent with what `RStudio` does for other distributions e.g., `pnorm` and `pt`.

To find $P(X > 5.2)$ where $X \sim \chi^2(5)$, we need to note that $P(X > 5.2) = 1 - P(X \leq 5.2)$ and type:
`1-pchisq(5.2,5)`

Calculating quantiles for the χ^2 distribution

The function `qchisq` returns quantiles for the χ^2 distribution. If $X \sim \chi^2(k)$ where k is the degrees of freedom, then we can find the value c that satisfies $P(X \leq c) = a$ for any a between 0 and 1, using `qchisq(a,k)`.

For example, let's say we want to find the lower 1% point from the χ^2 distribution with 10 degrees of freedom. That is, we want to find c such that $P(X \leq c) = 0.01$ where $X \sim \chi^2(10)$. To do this, we just type:
`qchisq(0.01,10)`

Now let's say we want to find the upper 5% point of the χ^2 distribution with 10 degrees of freedom. In this case we need to do a little manipulation to get the answer, in order to turn this expression into a left-tailed probability. If $P(X \geq c) = 0.05$ then $P(X < c) = 1 - 0.05 = 0.95$. So we can find the upper 5% point of $\chi^2(10)$ using:

```
qchisq(0.95,10)
```

A more direct way of doing this is to type

```
qchisq(1-0.05,15)
```

Inference Using RStudio

One-sample inference about μ when σ is known

Consider the following problem:

> (From Moore *et al*) Bottles of cola drink are supposed to contain 300 ml of cola. The distribution of contents is normal with standard deviation 3 ml. The contents of six bottles are
>
> 299.4 297.7 301.0 298.9 300.2 297.0
>
> *Is there evidence that the true mean amount of cola in bottles is less than 300ml?*

First we will type in the data from the Cola dataset (given it only contains 6 observations):

```
Cola=c(299.4, 297.7, 301.0, 298.9, 300.2, 297.0)
```

One-sample z-test for μ

The method for a one-sample z-test on RStudio is very similar to what we did on Excel – it must be done "by hand", with the calculations being done one step at a time. For the Cola data, where $H_0 : \mu = 300, \quad H_a : \mu < 300$:

```
m=mean(Cola)
mu.0=300
sigma=3
n=6
z=(m - mu.0) / ( sigma / sqrt(n) )
```

Typing `z` then returns the observed value of the test statistic, and you should get -0.78928.

To find the P-value for the one-sided test, we want to calculate $P(Z < z)$:

```
pnorm(z)
```

which returns the P-value:

```
[1] 0.2149742
```

How would you modify the above method in order to calculate a P-value if $H_a : \mu \neq 300$? Hint: A two-sided test involves calculating $2P(Z > |z|)$.

Confidence interval for μ when σ is known

Following on from the above working, in order to construct a 95% confidence interval for μ when σ is known:

```
z.star=qnorm(0.975)
margin=z.star * sigma / sqrt(n)
lower.ci=m - margin
upper.ci=m + margin
c(lower.ci, upper.ci)
```

which should return:

```
[1] 296.6329 301.4338
```

Why 0.975? That's because a 95% confidence interval means we want the central 95% of the area, which implies a total of 5% to either side of this. This implies we have 2.5% on the right, and so we want an upper 2.5% point or quantile.

How would you modify the above method in order to construct a 90% confidence interval for μ?

Note: It is actually deliberate that there is NO direct command for performing a z-test or calculating an associated confidence interval in `RStudio` – that's because z-tests are almost always useless because you don't know σ!

One-sample inference about μ when σ is not known

`RStudio` has an in-built function specially designed for making inferences about μ when σ is unknown (which is pretty much always!). We will illustrate this function using the following problem:

> An archaeologist discovers seven fossil skeletons from a previously unknown species of miniature horse. It is reasonable to assume that shoulder heights are normally distributed. Below are the shoulder heights, in centimeters.
>
> 45.3 47.1 44.2 46.8 46.5 45.5 47.6
>
> A present-day species that might be closely related to these fossil skeletons has average shoulder height of 44.1 centimeters.
>
> *Is there evidence that the fossil skeletons are of a different-sized animal?*

One-sample t-test for μ

A one-sample t-test for μ is straightforward on `RStudio`, using the function `t.test`. To do a one-sample t-test for the above problem:

```
shoulder=c(45.3, 47.1, 44.2, 46.8, 46.5, 45.5, 47.6)
t.test(shoulder, mu=44.1)
```

The expression `mu=44.1` in the above command tells RStudio that the null hypothesis is $H_0 : \mu = 44.1$. If no such value is entered, RStudio will assume that a default null hypothesis is $H_0 : \mu = 0$.

The default for `t.test()` is a two-sided tests. However you can ask RStudio to do one-sided t-tests. For a one-sided t-test of H_0 against $H_a : \mu > 44.1$ for the above shoulder height data, you would type:

```
t.test(shoulder, mu=44.1, alternative="greater")
```

Using `alternative="less"` would test H_0 against $H_a : \mu < 44.1$.

Confidence interval for μ when σ is unknown

You can use the `t.test` output to construct a level C confidence interval for μ. You might have noticed in the output from

```
t.test(shoulder, mu=44.1)
```

that there are some additional lines

```
95 percent confidence interval:
45.04226 47.24346
```

which give the lower and upper limits of the 95% confidence interval for μ.

To construct a 98% confidence interval, type:

```
t.test(shoulder, conf.level=0.98)
```

Note: You could include `mu=44.1` in this command if you wanted, but this would ONLY affect the P-value in the output and not the confidence interval.

Double Note: You should not set the `alternative` argument when you are interested in a confidence interval, because this will produce a one-sided confidence interval (something we do not study consider in this course and which is often not that helpful).

For more details on any of the above, see the help for the `t.test` function by typing `?t.test` or searching `t.test` in the *Help* tab.

A (slower) alternative to using `t.test` is to construct the confidence interval "by hand" as below:

```
m=mean(shoulder)
s=sd(shoulder)
n=7
t.star=qt(0.975, n-1)
margin=t.star * s / sqrt(n)
lower.ci=m - margin
upper.ci=m + margin
c(lower.ci, upper.ci)
```

Making inferences about p using RStudio

Now we will consider the case where we have a simple random sample from a categorical variable that has two possible outcomes, "success" and "failure"), and we want to make inferences about p, the probability of "success". We will focus on the following problem:

> Many mammals have a higher likelihood of giving birth to offspring in Spring than at other times of the year. Is this true of humans too?
>
> Out of 381 statistics students, 96 of them were born in the Spring months (September-October).
>
> *Does this provide evidence that people are more likely to be born in Spring?*

We will use RStudio to answer this question by constructing a one-sample test for p and constructing a level C confidence interval for p.

One-sample test for p

Use the following commands to do a one-sample test of p, testing $H_0: \ p = 0.25$ against $H_a : p > 0.25$ for the above problem:

```
n=381
p.hat=96/n
p.0=0.25
se.0=sqrt(p.0*(1-p.0)/n)
z=(p.hat - p.0)/se.0
z
```

which should return as your test statistic:

```
[1] 0.08873565
```

Since we are doing a one-sided test with $H_a : p > 0.25$, then we want $P(Z > z)$:

```
1 - pnorm(z)
```

which should return the following P-value:

```
[1] 0.464646
```

How would you modify the last step from the above in order to do a two-tailed test?

Confidence interval for p

Use the following commands to find a 95% confidence interval for p, the true chance of a student being born in Spring:

```
se=sqrt(p.hat*(1-p.hat)/n)
z.star=qnorm(0.975)
margin=se*z.star
ci.low=p.hat - margin
```

```
ci.high=p.hat + margin
c(ci.low,ci.high)
```

RStudio will then return:

```
[1] 0.2083753 0.2955617
```

This tells us that a 95% confidence interval for p is (0.2083753, 0.2955617).

*How could you change the formula for calculating z^* in order to calculate a **90%** confidence interval?*

Inference about the mean difference from two independent samples

Two-sample inference about means is easy to do on RStudio, as we can again use the `t.test` function. Note that this is the same function we used for one-sample inference about means, but it is used in a slightly different way here. There are two different types of method, depending on how your data are stored.

Two-sample t test for a quantitative and a categorical variable

We will use the `survey` dataset to do a two-sample t-test to see if there is evidence that the mean amount students spend on a hair-cut differs for males and females. Assuming you've imported the `survey` dataset into RStudio (see Section :How to import a text file into RStudio) then type:

```
t.test(survey$hair.cost~survey$gender, var.equal=T)
```

Note that this uses the same function as a one-sample t-test, but RStudio knows to do a two-sample t-test because two variables were specified in the above command – a quantitative variable (`hair.cost`) which contains the measurements, and a categorical variable (`survey`) that says which category each measurement belongs to. Note the use of the '$\sim$' here, which means "against". Recall we used this symbol also when constructing comparative boxplots and scatterplots!

If you don't specify the option `var.equal=T` then RStudio will do a t-test without assuming equal variances, as described in Moore *et al.*

If we wanted to do a one-sided test rather than a two-sided test, we could specify this using `alternative="greater"` or `alternative="less"` as previously.

Two-sample t test for two sets of quantitative measurements

Sometimes, rather than being given a dataset containing the quantitative variable and the categorical variable of interest, we are simply given two sets of measurements, corresponding to the two samples of data. RStudio is able to perform a two-sample t-test in this situation, as described for the following problem from week 9 lectures:

> What is the effect of smoking during pregnancy on children?

> A randomized controlled experiment investigated this by injecting pregnant guinea pigs with 0.5 mg/kg nicotine hydrogen tartrate in saline solution. The control group was injected with the saline solution.
>
> The learning capabilities of the **offspring** of these guinea pigs was then measured using the number of errors that were made trying to find food in a maze.
>
> *Is there a detrimental effect of nicotine on the development of the guinea pig offspring?*

First of all, you will need to import the dataset from the Excel file `smokePregnant.xls`, and save it in `RStudio` as the object `smoke.preg`. Please see Section : How to import data from Excel into `RStudio`, for details on how to do this.

If you now type:
```
smoke.preg
```

then you will notice that the dataset has two columns: `smoke.preg$Treatment` contains the data for treatment guinea pigs, and `smoke.preg$Control` contains the data for the control group. To do a two-sample t-test of $H_0 : \mu_{\text{Treatment}} = \mu_{\text{Control}}$ versus $H_a : \mu_{\text{Treatment}} > \mu_{\text{Control}}$:
```
t.test(smoke.preg$Treatment, smoke.preg$Control, alternative="greater", var.equal=T)
```

Confidence interval for $\mu_1 - \mu_2$

You can use the `t.test` output to construct a confidence interval for the mean difference $(\mu_1 - \mu_2)$. For example, for the guinea pig data stored in `smoke.preg` from above:
```
t.test(smoke.preg$Treatment, smoke.preg$Control, var.equal=T)
```

which will return in the output
```
95 percent confidence interval:
4.460667 37.339333
```

As usual, we can change the confidence level using the `conf.level` option, for example:
```
t.test(smoke.preg$Treatment, smoke.preg$Control, var.equal=T, conf.level=0.90)
```

Note that μ_1 is equivalent to $\mu_{\text{Treatment}}$ and μ_2 is equivalent to μ_{Control}. This is because in the `t.test()` command above, we put `smoke.preg$Treatment` first and `smoke.preg$Control` second.

As previously, we want to make sure we don't use the `alternative` argument of the `t.test` function to specify a one-sided test when we are specifically interested in calculating a confidence interval.

Inference about the mean difference from a paired sample

The `t.test` function on `RStudio` can also be used to analyze paired data. Consider the following problem from week 10 lectures:

Do ravens intentionally fly towards gunshot sounds (to scavenge on the carcass they expect to find)? Crow White (what an appropriate name!) addressed this question by counting raven numbers at a location, firing a gun, then counting raven numbers 10 minutes later. He did this in 12 locations. Results:

location	1	2	3	4	5	6	7	8	9	10	11	12
before	1	0	1	0	0	0	0	5	1	1	2	0
after	2	3	2	2	1	1	2	2	4	2	0	3

Is there evidence that ravens fly towards the location of gunshots?

First import the data into `RStudio` from the `ravens.xls` Excel file. Please see Section : How to import data from Excel into `RStudio`, for details on how to do this. Note the top four rows of the `ravens.xls` need to be deleted before saving as a text file, so that row 1 contains the variable names, and so that the rows immediately below contain the data.

Suppose you imported the dataset using a command like
`ravens=read.csv("ravens.csv", header=T)`

Then, we can view the data by typing
`ravens`

Note that there are three columns, and in particular, `ravens$before` contains the "before" counts, while `ravens$after` contains the "after" counts.

Paired t-test

To do a paired t-test for the raven data, to test $H_0: \mu_{\text{after}} = \mu_{\text{before}}$ against $H_a: \mu_{\text{after}} > \mu_{\text{before}}$:
`t.test(ravens$after, ravens$before, paired=T, alt="greater")`

We have used the function `t.test` to do this test, just as we did for inference about means from one-sample or from two independent samples. Moreover, because two variables were entered rather than one, then `RStudio` knew to do some form of t-test involving two samples.

However, we have now also specified `paired=T`, and so `RStudio` knows to do a paired t-test rather than a two-sample t-test. It is important to remember to include `paired=T`, otherwise `RStudio` will do the wrong type of two-sample test!

How would you modify the above command in order to do a ***two-sided*** *paired t-test using the raven data?*

Confidence interval for the mean difference from a paired sample

To construct a confidence interval for the mean difference from a paired sample, using the `raven` dataset:
`t.test(ravens$after, ravens$before, paired=T)`

This will return in the output:

```
95 percent confidence interval:
-0.1117494 2.2784161
```

As previously, we can change the confidence level using the `conf.level` option.

How to make inferences about the relationship between two categorical variables

To do a χ^2 test for independence of two categorical variables using `RStudio`, you can simply run the `summary` function on a table of frequencies.

For example, to do a χ^2 test for independence of the `gender` and `study.area` variables from the `survey` dataset, first construct a table of frequencies to summarise the data (see Section : How to numerically summarise the relationship between two categorical variables using `RStudio`)

```
tab=table(survey$gender, survey$study.area)
```

Now apply the `summary` function to this table:

```
summary(tab)
```

The value labeled `Chisq =` gives the X^2 test statistic, `df =` tells us the degrees of freedom, and `p-value =` tells us the P-value.

Is there evidence that males and females students have different subject preferences?

Alternatively, if you have already summarized your data in a table and you want to enter your table of frequencies directly into `RStudio` and store them in a `table`, you can do this as described in Section : How to type in a table of frequencies. After creating the table, then do `summary` to conduct the χ^2 test for independence.

How to make inferences about the relationship between two quantitative variables

We will illustrate how to make inferences about the relationship between two quantitative variables using the `concept` dataset. Please make sure you have imported the `concept` dataset into `RStudio` by following the steps outlined in Section : How to graph one quantitative variable versus another quantitative variable using `RStudio`.

The first step is to fit a least squares regression line between GPA and IQ of the students, and store it as an object. We will call it `mymodel`:

```
mymodel=lm(GPA~IQ, dat=concept)
```

`lm` stands for "linear model", and so the above syntax is telling `RStudio` to fit a linear regression line to of GPA against (the '~' symbol is back!) IQ, where these columns are found in `concept`. Note that alternatively, you can have typed

```
mymodel=lm(concept$GPA~concept$IQ)
```

Everything you need to know about your linear regression fit is now stored in the object

`reg`. To find out what sort of information you can find out, try typing
`names(mymodel)`

Information is stored within `mymodel` under each of the labels that are printed out when you type the above line. For example,
`mymodel$coefficients`

will return the estimated slope and intercept of your least squares regression line. By exploring the different objects that are stored in `mymodel`, you could find out all you need to know about a linear regression fit. Below are some tricks to cut straight to the most important bits.

How to obtain linear regression summary output

For summary output from your linear regression, simply type:
`summary(mymodel)`

See if you can answer the following questions, using the summary output:

- Is there evidence of a relationship between GPA and IQ for the students?
- What is the equation of the least squares regression line for predicting a students GPA from their IQ?
- What proportion of variation in GPA is explained by IQ?

Constructing residual plots

The residuals from your fit can be accessed by typing either `mymodel$residuals` or `residuals(mymodel)`. The predicted values from a linear regression fit can be accessed using either `mymodel$fitted` or `fitted(mymodel)`.

And to construct a residuals versus fitted values plot, you could type:
`plot(fitted(mymodel), residuals(mymodel))`

So, for example, to construct a normal quantile plot of residuals, you could type:
`qqnorm(mymodel$residuals)`

To create a scatterplot and add the fitted line, type (See also Section : How to graph one quantitative variable versus another quantitative variable using `RStudio`):
`plot(GPA~IQ, data=concept)`
`abline(mymodel)`

A more automated way of constructing residual plots is to simply use `plot(mymodel)`. This produces four main types of residual plot from regression. To display all four, first split the window into a 2×2 display, then plot your regression object:
`par(mfrow=c(2,2))`
`plot(reg)`

The most important of these plots is the first one (residual vs fits). To construct just this residual plot:

```
plot(reg, which=1)
```

To just produce a normal quantile plot of residuals:

```
plot(reg, which=2)
```

More on linear models

Load the `plantHeightSingleSpp` dataset.

Confidence intervals for key regression parameters

Just use the `confint` function as before. *e.g.*

```
ftLatRain=lm(height lat+rain,data=dat)
confint(ftLatRain)
```

Or to get a confidence interval for the coefficient of latitude only:

```
confint(ftLatRain,"lat")
```

Testing hypotheses in a regression setting

To test for effects of terms in the regression model, use the `summary` function. *e.g.* To test for an effect of latitude after controlling for precipitation:

```
summary(ftLatRain)
```

Is there any evidence of an effect of climate additional to that explained by latitude?

Overall, is there evidence that the regression model explains significant variation in height?

To compare two (nested) regression models, e.g. to test if there is any effect of climate in addition to that explained by latitude, use `anova`:

```
ftLat=lm(height lat, data=dat)
ftLatClim=lm(height lat+rain+temp, data=dat)
anova(ftLat, ftLatClim)
```

Is there any evidence of an effect of climate additional to that explained by latitude?

Checking assumptions

We can construct residual plots in the same manner as described earlier:

```
par(mfrow=c(2,2))
plot(reg)
```

And you can use `which` to focus on a specific type of residual plot.

Constructing partial-residual plots

```
library(car)
ft=lm(height~rain+lat, data=dat)
crPlots(ft, terms=~lat, xlab="Latitude", ylab="Height|rain", grid=F)
```

Checking for collinearity

Using variance inflation factors:

```
library(car)
vif(ftLatClim)
```

Are any of the terms in the model introducing multicollinearity?

ANOVA tasks

Analysis of variance table

Load the `HabitatConfig` dataset and subsample to small patches only:

```
dat=read.csv("HabitatConfig.csv")
dat=dat[dat$Size=="SMALL"]
dat$Dist=factor(dat$Dist)
```

To fit the model and then construct an ANOVA table:

```
ft=aov(log(Total)~Time*Dist, data=dat)
summary(ft)
```

Note you can also do this using the `lm` and `anova` functions, but using `aov` instead gives access to multiple comparisons.

Note also that you don't have to have all categorical x variables to use `aov`, but you do for some of the following functions...

Constructing an interaction plot (categorical x only)

To construct an interaction plot:

```
interaction.plot(dat$Dist, dat$Time, ft$fitted, trace.label="Time (weeks)")
```

Multiple comparisons (categorical x only)

Use the `TukeyHSD` function. If there are multiple terms in the model, the `which` argument lets you tell `R` which terms to do multiple comparisons for, *e.g.*

```
TukeyHSD(ft,which="Dist")
```

You can also try the `pairwise.t.test` function.

To plot the result:

```
tk=TukeyHSD(ft, which="Dist")
plot(tk, las=1)
```

(the argument `las=1` plot labels horizontally on the y axis: works better for long labels)

Including polynomial terms in the model

For the plant height data:

```
ft=lm(height~poly(cbind(rain,temp),degree=2), data=dat)
```

Model selection

Cross-validation approaches

Load the `plantHeightSingleSpp` data and store as `dat`.

Validation (single test/training split)

To get the mean squared error for a linear model with `temp` and `rain` as predictors, using $n^{0.75}$ training observations:

```
n=dim(dat)[1]
indTrain=sample(n, n^0.75) #select a training sample of size n^0.75:
datTrain=dat[indTrain,]
datTest=dat[-indTrain,]
ft1=lm(log(height)~temp+rain, dat=datTrain)
pr1=predict(ft1, newdata=datTest)
mean( (log(datTest$height)-pr1)^2 )
```

This general approach will work for any type of model that works in combination with the `predict` function. Note however that when the response is no longer assumed to be normally distributed, mean squared error is no longer the best criterion to use (*e.g.* log-likelihood is often a safer bet in such settings).

Cross-validation

To repeat validation 50 times, return the mean squared error (averaged over the 50 runs), and the standard error:

```
n=dim(dat)[1]
nValid=50 #to use 50 validation datasets
mse=rep(NA, nValid) #start a vector which will store mse from each validation split
for(iValid in 1:nValid)
{
  #select a training sample of size n^0.75:
  indTrain=sample(n, n^0.75)
  datTrain=dat[indTrain,]
  datTest=dat[-indTrain,]
  ft1=lm(log(height)~temp+rain, dat=datTrain)
  pr1=predict(ft1, newdata=datTest)
  mse[iValid]=mean( (log(datTest$height)-pr1)^2 )
}
print(mean(mse))
print(sd(mse))
```

K-fold cross-validation

To do 5-fold cross-validation and return the mean squared error:

```
library(DAAG)
ft1=lm(log(height)~temp+rain, dat=dat)
cv1=cv.lm(df=dat, ft1, m=5, seed=1) # 5 fold cross-validation
attr(cv1,"ms")
```

To see how much this changes as you choose different random test/training splits, try again using a couple of different values for seed.

Computing AIC or BIC

```
ft1=lm(log(height)~temp+rain, dat=dat)
AIC(ft1)
BIC(ft1)
```

These functions work for most types of fitted model.

Subset selection

To fit a stepwise (forward and backward) selection path using AIC for a linear model:

```
library(MASS)
ft=lm(log(height)~temp+rain+rain.wetm+temp.seas, dat=dat)
step=stepAIC(ft)
step$anova
```

To do forward selection only using BIC:

```
n=dim(dat)[1]
ft=lm(log(height)~1,dat=dat)
forward=stepAIC(ft, direction="forward",
                scope=~temp+rain+rain.wetm+temp.seas, k=log(n))
forward$anova # display results
```

To do backward selection only using AIC:

```
ft=lm(log(height)~temp+rain+rain.wetm+temp.seas, dat=dat)
backward=stepAIC(ft, direction="backward")
backward$anova
```

To do all-subsets selection, then plot BIC of the best two models of each size:

```
library(leaps)
leaps=regsubsets(log(height)~temp+rain+rain.wetm+temp.seas, data=dat, nbest=2)
plot(leaps)
```

The `regsubsets` function works for linear models and not much else. The `stepAIC` functions on the other hand work for most types of model.

LASSO

To fit a full LASSO path, estimate the nuisance parameter using 5-fold cross-validation, and plot predictive performance as a function of the nuisance parameter:

```
X=cbind(temp = dat$temp, rain = dat$rain, rain.wet = dat$rain.wetm,
  seas = dat$temp.seas)
ft.cv=cv.glmnet(X, log(dat$height), k=5)
plot(ft.cv)
```

To then fit to the full dataset the model which minimised mean squared error, and return the slope coefficients:

```
ft=glmnet(X,log(dat$height), lambda=ft.cv$lambda.min)
ft$beta
```

Generalized Linear Models

Use the `glm` function – works in almost exactly the same way as the `lm` function, except:

- You need to specify a `family` argument. Type `?family` for the main options, but most commonly, `family=binomial` for presence/absence data, `family=poisson` for conut data (but check for overdispersion!!)
- `aov`, `crPlots` and `cv.lm` no longer work. But pretty much everything else described previously is OK.

For example, logistic regression of crab presence/absence:

```
ft=glm(I(dat$Crab>0)~Time*Dist, family=binomial, data=dat)
AIC(ft)
summary(ft)
```

Negative binomial regression

If your data are overdispersed compared to the Poisson, a negative binomial is a good option. This is pretty straightforward using the `mvabund` package:

```
library(mvabund)
load("reveg.RData")
Haplotaxida=datRed[,12]
ftMany=manyglm(Haplotaxida~treatment, family="negative.binomial")
```

This uses a log-link. If you want a different link, some are available in the `MASS` package.

Residual plots for GLMs

To compute Dunn-Smyth residuals, refit using the `mvabund` package:

```
library(mvabund)
ftMany=manyglm(I(dat$Crab>0)~Time*Dist, family="binomial", data=dat)
plot(ftMany)
```

Since Australia had such a crushing win in the Ashes this summer you may prefer your plot to look like this:

```
par(bg="gold")
plot(ftMany,  col="darkgreen")
```

Note that there are two differences from the last plot:

- The colours are more appropriate. Assuming they win (80% chance I reckon).
- The residuals are slightly different. There is randomness in their definition, so every time you call the `plot` or `residuals` function, you will get a slightly different result. If you see a pattern, it's worth repeating a few times to check it's really there.

Inference about GLMs – model-based inference

Use the `anova` or `summary` function. For `glm` objects:

```
anova(ft,test="Chisq")
```

will return an *analysis of deviance* table. This is fine for most applications (but problems for small sample size and small counts).

Inference about GLMs – design-based inference

Use the `mvabund` package, applying the `anova` or `summary` function to `manyglm` objects:

```
anova(ftMany)
```

will return an *analysis of deviance* table but with significance determined by bootstrapping probability integral transform residuals (the "PIT-trap").

The `manyglm` function currently only works with certain combinations of families and link functions:

`"binomial"` for logistic regression of presence/absence data.

`"poisson"` for log-linear models of counts assuming data come from the Poisson.

`"negative.binomial"` for log-linear models of counts assuming data come from the negative binomial (over-dispersed compared to the Poisson).

`"gaussian"` for linear models assuming data are normally distributed.

Offsets – accounting for known but different sampling intensities

If you have a variable which is known (or thought) to be proportional to the response in a log-linear model, such as `pitfalls`, you can include it as an offset:

```
ftManyOff=manyglm(Haplotaxida~treatment+offset(log(pitfalls)),
  family = "negative.binomial")
```

Non-linear models – smoothers

Load the plant height dataset.

```
library(mgcv)
ft.gam=gam(log(height)~s(lat), dat=dat)
summary(ft.gam)
```

Is there any evidence of an effect of latitude?

Visualising the smoother

To plot the smoother(s) in your fitted GAM object:

```
plot(ft.gam)
```

With observations on there too (as small, open circles):

```
plot(ft.gam, residuals=T, pch=1, cex=0.4)
```

Number of knots

To specify an upper limit to the number of knots (`k`), to allow for the possibility of an extra-wiggly line:

```
ft.gam=gam(log(height)~s(lat,k=25), dat=dat)
```

Residual plot

You need to construct this manually:

```
plot(residuals(ft.gam)~fitted(ft.gam), xlab="Fitted values", ylab="Residuals")
abline(h=0,col="red")
```

Cyclical predictors

Load the `maunaloa` data, which contains monthly ambient CO_2 measurements from near the top of a frigging huge volcano in the middle of the Pacific.

First coerce data into a vector, format the `Date` variable, and kick out missing CO_2 values:

```
n.rows=length(dat$Year)
datBig=data.frame(year=rep(dat$Year,each=12), co2=as.vector(t(dat[,2:13])))
datBig$month=rep(1:12, n.rows)
datBig$Date=as.Date(paste(datBig$year,datBig$month,1,sep="-"), format="%Y-%m-%d")
datBig$DateNum=as.numeric(datBig$Date)
dat=datBig[is.na(datBig$co2)==F,]
```

Now the `DateNum` variable can be used as a numerical predictor.

For a smoother against Date and a sine curve for seasonality:

```
library(mgcv)
ft=gam(co2~s(DateNum)+sin(month/12*2*pi)+cos(month/12*2*pi), data=dat)
```

To plot the data (line-plot) with the fitted model:

```
plot(dat$co2~dat$Date, type="line", ylab=expression(CO[2]), xlab="Time")
points(predict(ft)~dat$Date, type="line", col="red")
```

Residual plot of periodicity model

```
plot(residuals(ft)~sin(dat$month/12*2*pi), type="line", xlab="Season")
```

(could have used `cos` too, one or the other)

Mixed Models

Linear mixed effects models

Load the `EstuaryInvert` data.

```
library(lme4)
ft=lmer(Total~Mod+(1|Estuary), data=datSmall)
```

Residual plots

```
#Using predicted values conditional on random effects
scatter.smooth(residuals(ft)~fitted(ft))
abline(h=0,col="red")

#Using unconditional predicted values, no random effects used on X-axis
scatter.smooth(residuals(ft)~predict(ft,REform=NA))
abline(h=0,col="red")
```

Quick-and-dirty inference from mixed models

```
summary(ft) #for t-stats which are approximately t-distributed
ftInt=lmer(Total~(1|Estuary), data=datSmall, REML=F)
anova(ftInt, ft) #testing for Mod effect.
```

Note in both cases P-values are only approximate and smaller than they should be – if P-value is large don't bother looking further, if it is small but a bit borderline you can use a parametric bootstrap to get a better estimate.

Also note `anova` won't work when comparing models with and without random effects.

Parametric bootstrap for mixed models

Use the `simulate` function to simulate new values of the response variable, from a given mixed model. Doing this may times and recomputing a test statistic of inference we can test hypothesis (or construct confidence intervals), known as a "parametric bootstrap". For example, to test for an estuary effect, not possible when using the `anova` function:

```
nBoot=1000
lrstat=rep(NA,nBoot)
ftMod=lm(Total~Mod, data=datSmall)
lrObs=2*logLik(ft) - 2*logLik(ftMod) # observed test stat
for(iBoot in 1:nBoot)
 {
  # simulate data
  datSmall$TotalSim=unlist(simulate(ftMod))

  # fit smaller (null) model to simulated data
  bNull=lm(TotalSim~Mod, data=datSmall)

  # fit larger (alternative) model to simulated data
  bAlt=lmer(TotalSim~Mod+(1|Estuary), data=datSmall, REML=F)

  # test stat for simulated data
  lrstat[iBoot]=2*logLik(bAlt) - 2*logLik(bNull)
}
mean(lrstat>lrObs) #P-value for test of Estuary effect
```

This is a computationally intensive procedure, fitting lots of GLMMs repeatedly (after regenerating data lots of times) so it takes a minute or so to run. It's a good idea when using this approach to start with a smaller value of `nBoot` (50, say) to get a rough idea of how long it will take to run (and also an early peek at where results are heading).

Note if using this approach to do a hypothesis test you should **simulate under the null**, in this case, the linear model `ftMod`. P-values are defined as probabilities calculated

assuming the null model is true, so simulated P-values always need to simulate from the null model.

Consider also the `PBmodcomp` from the `pbkrtest` package, if you are comparing between two mixed models specifically.

Confidence intervals

For fixed effects:

```
confint(ft) #CIs for fixed effects and variance components
rft=ranef(ft, postVar=T) #posterior mode and sd for random effects, kind of like CIs
plot(rft)
```